Interpolated N-Grams for Model Based Testing

Paolo Tonella, Roberto Tiella
Fondazione Bruno Kessler
Trento, Italy
{tonella, tiella}@fbk.eu

Cu D. Nguyen
University of Luxembourg
Luxembourg
duy.nguyen@uni.lu

ABSTRACT

Models – in particular finite state machine models – provide an invaluable source of information for the derivation of effective test cases. However, models usually approximate part of the program semantics and capture only some of the relevant dependencies and constraints. As a consequence, some of the test cases that are derived from models are infeasible.

In this paper, we propose a method, based on the computation of the N-gram statistics, to increase the likelihood of deriving feasible test cases from a model. Correspondingly, the level of model coverage is also expected to increase, because infeasible test cases do not contribute to coverage. While N-grams do improve existing test case derivation methods, they show limitations when the N-gram statistics is incomplete, which is expected to necessarily occur as N increases. Interpolated N-grams overcome such limitation and show the highest performance of all test case derivation methods compared in this work.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability, Experimentation

Keywords

Model based testing; test case generation; N-gram statistics

1. INTRODUCTION

In model based testing, models of the system under test are used to identify the relevant system behaviours to be tested [9]. Finite State Machine (FSM) models or their variants have been widely used for test case derivation [2, 16, 17]. Coverage criteria defined upon the model are often adopted to decide how test cases are derived from the model and among them state coverage and transition coverage are the most common ones.

Models can be defined upfront, during the design phase of software development, or they can be inferred from observations of actual executions of the system [7, 3, 12, 13, 23]. Automated model inference has been an intensive area of research in the last few years and the proposed techniques can be roughly grouped into two categories: (1) state abstraction techniques [7, 23], which monitor the application's state and abstract it by means of properly defined abstraction functions; and, (2) event-sequence abstraction [12, 13], which infer a regular grammar capable of recognising the observed event sequences, while also generalising its recognition power beyond the actually observed sequences.

Both manually defined and automatically inferred models express the program semantics in an incomplete way, as a consequence of the abstraction operated during model generation. In fact, a complete and precise specification of the system semantics is usually unaffordable and would require a very complicated and large model, which is against the initial goal of abstracting away the implementation details to get a concise system representation. On the other hand, the approximation necessarily introduced in the model makes some of the test cases derived from the model infeasible. Such test cases violate dependencies and constraints that are not expressed explicitly in the model. Test case infeasibility is one of the major open problems in model based testing [9].

The N-gram statistics expresses the frequency of occurrence of N-tuples of events. We compute the N-gram statistics by monitoring some program executions or by collecting some execution traces. We propose a novel model-based test case derivation method, where the next event to be added to a test sequence is determined by its probability conditioned to the occurrence of the previous N-1 events. By deriving test cases that respect the N-gram statistics, we increase the likelihood of generating feasible test cases. In fact, a given event is added to a test sequence only if it is allowed in the context of the previous N-1 events. Increasing N provides a longer context for test case derivation, but the problem is that the number of N-grams grows very quickly as N increases. Correspondingly, the N-gram statistics becomes largely incomplete at increasing N. To overcome this problem, we interpolate the N-gram statistics, by taking advantage of a longer context, when it exists and by falling back to a shorter context otherwise. In practice, the N-gram statistics is computed as an exponential interpolation from short to long contexts, where longer contexts are given an exponentially higher weight. Experimental results indicate that interpolated N-grams are superior to plain N-grams and to traditional test case derivation methods, both in terms of feasibility and coverage of the generated tests.

The paper is organised as follows: Section 2 provides some basic background on model based test case derivation. Section 3 presents our approach. Section 4 describes the experimental study conducted to validate it. Related works are commented in Section 5, followed by conclusions and future work, in Section 6.

2. BACKGROUND

In model-based testing, models determine the relevant subset of application behaviours to be considered for testing [19] – this step is called *test base generation* or *derivation* (e.g., with FSM models, an output test case may consist of a sequence of events). The adequacy criterion for model-based test case derivation has usually the form of coverage defined on the model. By focusing on specific equivalence classes of behaviours represented by the adequacy criterion of choice, an effective and thorough set of test cases can be obtained from the model. A survey of the state of the art in model based testing has been conducted by Dias Neto et al. [9] and Shafique [20].

One of the most frequently used kinds of model is the FSM model, even though some alternatives do exist (e.g., Briand et al. [4] use UML class and sequence diagrams). A node in a FSM represents a state of the application and it can be determined by, e.g., the values of class attributes (in case of object-oriented applications [11, 25]) or the values of graphical objects (in case of GUI-based applications [27, 15, 1]). FSMs are named *concrete* if each FSM state represents an actual application state or *abstract* if each FSM state represents a set (i.e., an equivalence class) of concrete states (e.g., [15, 8]). A transition in the FSM represents an application event/action (e.g., an input event, a method call, or an event handler invocation) that can change the application state, when executed. Additionally, guards and conditions can enrich the model to capture the context in which events and actions are executed.

By traversing the FSM, sequences of application events are extracted for test case generation, so as to satisfy some coverage criterion of choice [11]. State or transition coverage (every FSM state or transition must be exercised by at least one test case) is often adopted, even if domain-specific (e.g., semantically interacting events [27]) criteria are sometimes preferred.

Different test case derivation strategies can be used to produce a set of test cases that satisfies the chosen adequacy criterion. Graph visit algorithms are widely employed to this aim and among them, random graph visit (RAND), depth first visit (DFV) and breath first visit (BFV) are the most preferred and used. We will consider these three strategies as the baselines for comparison of the approaches proposed in this paper (NGRAM and INTERP, interpolated N-grams).

Figure 1 shows the pseudocode of the random test sequence generation strategy. The procedure *randomVisit* decides whether to add another event to the current test sequence or not in a stochastic way. With probability *RECURSE_PROB*, a randomly selected successor (notation s^e indicates that s is a successor node if event e is triggered) of the current node (n) is added ($p + s^e$) to the current event sequence (p) and *randomVisit* is invoked recursively.

```
proc randomVisit($n : Node, p : Path$) $\equiv$
  do
     if (randProb() $\leq RECURSE\_PROB \wedge$ succ[$n$] $\neq \emptyset$)
       then
          $s^e$ := randChoose(succ[$n$]);
          $p$ := randomVisit($s, p + s^e$);
     fi
     if ($p$ increases adequacy) then addToTestSuite($p$); fi
  od.
$/ * Main * /$
while adequacy criterion not satisfied do
     $p$ := randomVisit($startNode, \langle\rangle$);
od
```

Figure 1: Test sequence derivation by random graph visit

When recursion is not activated, the generated test sequence is added to the test suite, if it increases the test suite adequacy level. Multiple random visits are performed, until the adequacy criterion (e.g., transition coverage) is satisfied.

This algorithm is clearly non deterministic, because of the random choice of the FSM transition to traverse (invocation of *randChoose* with parameter *succ*[n]). The algorithm parameter *RECURSE_PROB* determines the length of the generated event sequences. To produce event sequences with an average length equal to that observed in real executions, *RECURSE_PROB* can be set to *AVG_TRC_LEN / (1 + AVG_TRC_LEN)*, where *AVG_TRC_LEN* is the average length of the monitored/traced event sequences.

Depth-first and breadth-first model traversals share the overall recursive scheme of RAND (see Figure 1). They differ from RAND in recursion condition and successor selection. Both DFV and BFV maintain a queue of not yet visited edges, which plays the role of the selection of the successor node. DFV makes use of a LIFO queue, to ensure in-depth path exploration, while BFV makes use of a FIFO queue, to ensure in-breadth path exploration. The recursion condition used by RAND is replaced by a check for emptiness on the queues of nodes not yet visited: when no path remains to be explored in the LIFO/FIFO queue, DFV/BFV terminate their execution. The pseudocode of DFV and BFV is provided in our workshop paper [24].

The DFV and BFV visit procedures are called multiple times from the start node, until the chosen adequacy criterion is satisfied. By construction, when the adequacy criterion is transition coverage, DFV and BFV are called just once. An adequacy criterion different from transition coverage (e.g., maximum test budget) may require multiple invocations of DFV/BFV. The result of the DFV and BFV procedures is non deterministic, since it depends on the order in which successor nodes are added to the LIFO/FIFO queues used by the two algorithms. In fact, when there are multiple successor nodes, they can be added to the LIFO/FIFO queues in any arbitrary order.

3. APPROACH

N-gram language models are widely used in Natural Language Processing (NLP) [10]. An N-gram language model is a probabilistic model where the probability $P(e|e_1, ..., e_n)$ that a word (an event in our case) e is preceded by a sequence of words (events) $e_1, ..., e_n$ depends only on the last $N-1$ words (with $N-1 < n$):

$$P(e|e_1, ..., e_n) = P(e|e_1, e_2, ..., e_{N-1})$$

Using probabilistic models, knowledge about the N-gram statistics supports *word prediction*. In turn, word prediction is a key component used to address several NLP tasks, such as speech recognition, handwriting recognition, machine translation, spell correction, natural language generation, etc [14, 10]. In fact, NLP algorithms admit usually multiple sentence derivations and N-gram statistics can be used to select the most likely among the possible derivations.

The problem with model based test sequence generation is somewhat similar. Among all possible event sequences that satisfy some adequacy criterion (e.g., transition coverage), only a subset represent *feasible* event sequences, i.e., event sequences that can be actually executed against the application under test. Infeasible sequences involve execution steps whose order is forbidden by the application under test or involve events whose valid inputs prevents the execution of a later subsequence. Avoiding the generation of infeasible event sequences is very similar to avoiding the derivation of unlikely sentences and N-gram statistics can be used in a similar way as in NLP to achieve such purpose. In fact, by generating event sequences that contain N-grams previously observed in real executions, the likelihood that such sequences will in turn be executable is increased.

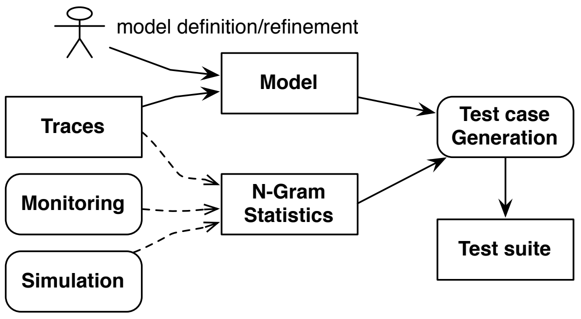

Figure 2: Overall view of the N-gram based test case generation method

Figure 2 shows a high level view of the proposed approach. While graph visit test case generation algorithms (RAND, DFV and BFV) require just one input (i.e., the model), N-gram based test case generation needs two inputs: model and N-gram statistics. The model can be defined manually by the user; it can be inferred automatically from execution traces using state abstraction [8] or event sequence abstraction [3]; or, a mixed approach can be followed, in which the model is first inferred and then it is manually refined by the user.

N-gram statistics consist of the number of occurrences of the N-tuples $\langle e, e_1, e_2, \ldots, e_{N-1}\rangle$ (to be read as e preceded by e_1, preceded by e_2, etc.), which can be turned (by normalisation to 1) into conditioned probabilities $P(e|e_1, e_2, ..., e_{N-1})$. N-gram statistics and the associated conditioned probabilities are the key components of N-gram based test case derivation. Such statistics can be obtained by monitoring some executions of the application under test and collecting the N-tuple occurrence counts. We can monitor the application under real usage or actively run it with a random test generator and keep only successful traces. The statistics can be measured by analysing execution traces or logs. A third alternative is to run a simulator of the user interacting with the application under test, so as to collect a large amount of data to be used for N-gram statistics computation. Depending on the application (domain, technology, deployment, etc.), the most appropriate among these three data collection methods may differ. It should be noticed that monitoring requires to store just the N-gram statistics, while tracing involves storing of full execution traces. Simulation is the least invasive technique, but its applicability might be limited if the creation of a faithful and realistic simulator is not doable for a given application. Independently of the statistics collection method, a key question is when enough data has been collected for reliable N-gram estimate. To this aim, the convergence metrics commonly used in statistics can be resorted to. For instance, the distance between probability estimates before and after adding a new trace can be compared against a convergence threshold ϵ to decide if more observations are needed to converge to a reliable N-gram statistics.

3.1 N-Grams

```
proc ngramVisit($n : Node, p : Path$) $\equiv$
  do
     if (randProb() $\leq RECURSE\_PROB$)
       then
            $s^e$ := ngramChoose(succ[$n$], suffix($p, N-1$));
            $p$ := ngramVisit($s, p + s^e$);
     fi
     if ($p$ increases adequacy) then addToTestSuite($p$); fi
  od.
$/ * Main * /$
while adequacy criterion not satisfied do
     $p$ := ngramVisit($startNode$, $\langle\rangle$);
od
```

Figure 3: Test sequence derivation by N-gram probability

Figure 3 shows the pseudocode of the N-gram test sequence generation strategy (in the following, by NGRAM2, NGRAM3, etc., we indicate that N, the size of the tuples considered in the N-gram statistics, is respectively 2, 3, etc.). The procedure is similar to the random visit shown in Figure 1, the key difference being the way in which the next event to add to the event sequence is chosen. Instead of choosing which successor node to add randomly (i.e., according to a uniform probability distribution), the successor to add is chosen in accordance with the conditioned probabilities of the next events given the last $N-1$ events in the current execution path p = "$\ldots, e_{N-1}, \ldots, e_1$":

$$ngramChoose(S, \langle e_1, \ldots, e_{N-1}\rangle) := s^e \in S$$
$$with\ prob.\ P(e \mid e_1, \ldots, e_{N-1})$$

The conditioned probabilities $P(e \mid e_1, \ldots, e_{N-1})$ are estimated from the frequency of occurrence of the N-tuples $\langle e^{(1)}, e_1, \ldots, e_{N-1}\rangle, \ldots, \langle e^{(k)}, e_1, \ldots, e_{N-1}\rangle$. Specifically, the choice among $e^{(1)}, \ldots, e^{(k)}$ of the next event to add to the event sequence has a probability which is proportional to the frequency of occurrence of the respective tuples $\langle e^{(1)}, e_1, \ldots, e_{N-1}\rangle, \ldots, \langle e^{(k)}, e_1, \ldots, e_{N-1}\rangle$ (an example is presented later, in Figure 4).

When no N-tuple $\langle e, e_1, \ldots, e_{N-1}\rangle$ appears in the available N-gram statistics (e.g., none is present in the execution traces used to compute the N-gram statistics), the next event e is selected randomly among the possible transitions outgoing from the current node n (i.e., $P(e \mid e_1, \ldots, e_{N-1})$ is zero for all successor events e, hence providing no help for the selection operated by *ngramChoose*). In such cases the NGRAM strategy degenerates to the random strategy. This is expected to occur at increased frequency when N takes higher and higher values, since only a small fraction of the possible N-tuples will be represented in the observed traces. We can thus predict that the performance of NGRAM will converge to the performance of RAND as the value of the parameter N increases.

3.2 Interpolated N-Grams

The basic N-gram test sequence derivation algorithm shown in Figure 3 is quite sensitive to the choice of the value of N. If a low value is chosen for N, the N-gram statistics will be quite reliable, being supported by a relatively large number of occurrences of short N-grams. On the other hand, the context captured by a low N might be too limited to capture the feasibility constraints necessary to generate only feasible sequences. When the dependencies between events span a longer interval than N, they will not be captured in the N-gram statistics (with low N) and infeasible sequences may be generated still with high probability.

Choosing a high value for N ensures that a long enough context is captured in the N-gram statistics and that dependencies between events that have a relatively big temporal separation are taken into account, hence increasing the chance of generating only feasible sequences. However, for a given set of traces, the number of instances of N-grams becomes exponentially smaller when compared to the total number of possible N-grams, as N increases. This means that the chance of finding at least one N-tuple $\langle e, e_1, \ldots, e_{N-1}\rangle$ represented in the N-gram statistics decreases exponentially with N. When no such N-tuple is found, the *ngramChoose* procedure of the NGRAM algorithm performs a uniformly random selection, becoming indistinguishable from the random visit algorithm RAND.

Interpolated N-gram statistics [10] are computed by taking advantage of long N-tuples, when these exist (i.e., they have non zero frequency of occurrence), and by falling back to a smaller value of N when N-gram statistics are not available for larger N. The idea is to interpolate the conditioned probabilities of the next event e given the preceding k events, with k varying between 1 and some big value $N-1$ ($P(e|e_1, e_2, ..., e_k)$, with $k \in [1, \ldots, N-1]$). For some k, probabilities may still be zero, when no N-gram statistics is recorded for the tuple $\langle e, e_1, e_2, ..., e_k\rangle$, but the interpolated value, computed by varying k between 1 and $N-1$, is very unlikely to be zero.

Interpolated probability values are computed as follows. For each k between 1 and $N-1$, conditioned probabilities $P(e|e_1, e_2, ..., e_k)$ are computed (some may be zero) and are then multiplied by a weight $w(k)$, which increases with k. Specifically, we have chosen an exponentially increasing function for the weight: $w(k) = 2^k$, so as to give substantially more importance to longer contexts, when these exist. The result is a weighted conditioned probability number (formally, no longer a probability) W_k:

$$W_k(e|e_1, e_2, ..., e_k) = 2^k P(e|e_1, e_2, ..., e_k) \quad (1)$$

Weighted probability numbers are then summed up (to interpolate their values) and normalised (to respect the constraint that they must sum up to 1, as required by probability distributions), resulting in the interpolated conditioned probabilities P^* of the next event e given previous events $\langle e_1, \ldots, e_{N-1}\rangle$:

$$S^*(e|e_1, e_2, ..., e_{N-1}) = \sum_{k=1}^{N-1} W_k(e|e_1, e_2, ..., e_k) \quad (2)$$

$$P^*(e|e_1, e_2, ..., e_{N-1}) = \alpha S^*(e|e_1, e_2, ..., e_{N-1}) \quad (3)$$

where α is a normalisation factor computed by summing the values S^* associated with all possible successor events e of the current FSM node n, given the previous execution path $p =$ "$\ldots, e_{N-1}, \ldots, e_1$":

$$\alpha = 1/ \sum_{e \in \, succEvents(n)} S^*(e|e_1, e_2, ..., e_{N-1}) \quad (4)$$

By assigning an exponentially increasing weight to longer tuples of previous events, N-grams with big values of N are given a large weight in the computation of the probability of the next event in the test sequence. However, when no such long N-gram exists (i.e., $P(e|e_1, e_2, ..., e_k)$ is zero for large k), the N-gram statistics available for smaller k is used (instead of resorting to a uniform selection, as done by the NGRAM algorithm). In fact, if W_k is zero for a large k, because the associated N-gram probability is zero, only the values of W_k computed for lower k will contribute to the computation of S^* (see equation (2)) and of the interpolated probabilities P^* (see equation (3)).

The interpolated N-gram test sequence derivation algorithm (INTERP) is the same as the NGRAM algorithm shown in Figure 3, the only difference being the probabilities used by *ngramChoose*. Given the interpolated N-gram statistics, the probability of the next event to add to the test sequence will be P^* instead of P, so that the chance of resorting to a random selection of the next event (because P is zero for the current path suffix of length $N-1$) is drastically reduced. In the following, we indicate by INTERP3, INTERP4, …, INTERP10 the variants of the interpolated N-gram algorithm with $N = 3, \ldots, 10$ (INTERP2 is the same as NGRAM2, hence it will not be considered).

Figure 4 shows an example of computation of the interpolated N-gram probabilities P^*. Given the N-gram statistics at the top-left, with the occurrence counts provided in the second column, let us consider two execution paths p_1 and p_2 (see top-right in the figure), both leading to the FSM state n which has two successor events e, d. Algorithm NGRAM2 will compute the probabilities of the next events e, d based on the occurrence counts of the bigrams $\langle c, e\rangle$ and $\langle c, d\rangle$ (respectively, 6 and 9), since only the last event in the executions paths p_1 and p_2 is used by NGRAM2. The resulting probabilities for both paths are 0.4 and 0.6, for e and d respectively. When applying NGRAM3, the existence of some

Traces	Count
a, b, c, d	9
a, b, c, e	1
h, g, c, e	5
x, y, w, d	5

$p_1 = \langle a, b, c\rangle$
$p_2 = \langle x, y, c\rangle$
$succEvents(n) = \{e, d\}$

	NGRAM2		NGRAM3		INTERP3	
	e	d	e	d	e	d
$a, b, c, ?$	0.4	0.6	0.1	0.9	0.2	0.8
$x, y, c, ?$	0.4	0.6	0	0	0.4	0.6

Figure 4: Example of conditioned probabilities computed by NGRAM and INTERP

occurrences of the trigrams $\langle b, c, e\rangle$ and $\langle b, c, d\rangle$ allows for the computation of the conditioned probabilities for the execution path p_1 (see Figure 4, columns under NGRAM3), while no trigram $\langle y, c, e\rangle$ and $\langle y, c, d\rangle$ is present, resulting in zero probabilities for p_2 (which means the algorithm will behave as RAND and will select between d and e with equal probability 0.5).

When INTERP3 is applied, a higher weight (equal to 4) is given to the probabilities computed by NGRAM3 for p_1. Half of this weight (i.e., 2) is given to the probabilities of NGRAM2. The weighted sum of the probabilities is 1.2 (4*0.1+2*0.4) for e and 4.8 (4*0.9+2*0.6) for d, which, once normalised, give the probabilities shown in Figure 4 under INTERP3. On the other hand, no probability is available under NGRAM3 for path p_2 and correspondingly the probabilities computed by INTERP3 are the same as those of NGRAM2.

This example shows some nice properties of interpolated N-grams. When N-grams of variable length are available for a given execution path (e.g., p_1), the longer N-grams are given a higher weight (correspondingly, the resulting probabilities, under INTERP3, are closer to NGRAM3 than to NGRAM2). When only shorter N-grams are available, instead of resorting to a uniform random selection of the next event, INTERP3 makes use of the shorter N-gram statistics which is available (for p_2, this is the NGRAM2 statistics).

4. EXPERIMENTAL RESULTS

We have conducted a set of experiments on five subject applications to answer the following research questions:

- **RQ1 (Feasibility):** How many feasible test sequences are generated by the interpolated N-gram, the non-interpolated N-gram, and by the baseline graph visit strategies?
- **RQ2 (Coverage):** What level of transition coverage is achieved by the interpolated N-gram, the non-interpolated N-gram, and by the baseline graph visit strategies?
- **RQ3 (Test suite size):** How many test sequences are generated by the interpolated N-gram, the non-interpolated N-gram, and by the baseline graph visit strategies?
- **RQ4 (Test case length):** What is the length of the test sequences generated by the interpolated N-gram, the non-interpolated N-gram, and by the baseline graph visit strategies?

The first two research questions are key to validate the proposed approach. We conjecture that interpolated N-gram based test sequence generation will produce less infeasible test sequences (hence, higher coverage) than non-interpolated N-gram and graph visit model traversal. With these two research questions we want to assess whether our conjecture is confirmed or not by the experimental data.

The last two research questions deal with some interesting properties of the automatically generated test suites, namely, the number of test sequences they contain and their length. To make the comparison fair, we adopt the same adequacy criterion with all alternative test sequence generation strategies: *transition coverage*, the most widely used coverage criterion in model based testing. All test suites produced by the various strategies will be transition coverage adequate and will contain only test cases that contribute to increasing such adequacy level (see algorithms in Figures 1, 3). It is the presence of infeasible test sequences that might diminish the actual level of transition coverage from 100% (reached by design, under the assumption of full feasibility) to some lower level, measured to answer RQ2.

4.1 Metrics

To address the four research questions listed above, we have collected the following metrics:

- **FEAS (RQ1):** Ratio between feasible test sequences and total number of test sequences generated by each test strategy.
- **COV (RQ2):** Ratio between covered transitions and total number of transitions in the model.
- **SZ (RQ3):** Number of test sequences in the test suite.
- **LEN (RQ4):** Average number of events in the test sequences added to each test suite.

While metrics COV, SZ and LEN can be easily measured automatically, using tools, metrics FEAS requires human judgment, since it is not possible to automatically decide if a test sequence is feasible. We manually defined a set of constraints for the subject applications to characterise the event sequences that can be legally submitted to and executed by the system under test. Such constraints are needed in this experiment to measure FEAS, but are not necessary when applying the proposed technique in a real setting.

4.2 Subjects

The five subject applications used in our empirical study have been selected by looking for open source web applications used in previous web testing research, with a rich and stateful client, which makes them appropriate for state based modelling. The features of the five subjects are listed in Table 1. Their source code includes PHP, Java, JavaScript, ActionScript, JSP, XML and CSS files, whose approximate size varies between 3.5 and 75 kLOC (thousands Lines Of Code, as reported by the UNIX utility `wc`).

Flexstore[1] is an online shopping application developed by Adobe and made available from the company's web site to

[1]http://www.adobe.com/devnet/flex/samples/flex_store_v2.html

Subject	**LOC**	**Nodes**	**Trans**	**Traces**
Flexstore	3.5k	9	42	100
Cyclos	75k	7	18	999
TheOrganizer	4k	19	68	10,000
TaskFreak	35k	9	46	10,000
HitList	5.2k	10	24	5

Table 1: Features of the subject applications

demonstrate the capabilities of their testing framework. Its client-side is developed in Flex (ActionScript) and run by the Flash plug-in. The application allows the user to browse a catalog of mobile phones and to focus on a subset of models by means of filters, such as price range, camera, tri-band, and video availability. The customer can select one or more models to perform comparisons among features. Eventually, customers can put one or more phones in their shopping cart.

Cyclos[2] is a popular open source Java/Servlet Web Application, supporting e-commerce and online payment. The client side runs Ajax JavaScript code. Its main features include: banking (e.g., payments, loans, brokering), e-commerce (e.g., advertising, member payments) and many others (e.g. access control, management). Cyclos is a quite large system. In our experiment we focused only on the payment functionality of Cyclos.

TheOrganizer[3] is a web application that supports management and organisation of the activities in a personal agenda. The server is written in Java (using Servlets, J2EE and Spring JDBC). The client is based on JavaScript/Ajax. Once authenticated, the user can add, remove and edit tasks, appointments, contacts and notes.

TaskFreak[4] is a project management web application. It is written in PHP and its client is based on the Ajax technology. The application supports the full life cycle of a project, including its initial creation, the definition of its tasks (including deadlines, priority and context) and their assignment to users.

HitList[5] is an Ajax-rich PHP based contact manager. It supports creation and editing of contact lists, including advanced search and mark (e.g., star/un-star) functionalities.

All subjects are being or have been used in previous empirical studies on web testing. Flexstore is being used as a case study of the European Union FP7 project FITTEST[6]; Cyclos has been used by Nguyen et al. [17] in their work on the combination of combinatorial and model-based testing; TheOrganizer, TaskFreak and HitList have been used by Mesbah et al. [16] in their work on Ajax testing.

We have obtained execution traces for the subject applications either by manually/randomly navigating and exercising the various application features or by running a carefully crafted user simulator (see Figure 2). Specifically, manual navigation was used with Flexstore and HitList; a simulator was run with TheOrganizer and TaskFreak; and finally random executions were used with Cyclos (among them, only traces of successful, non-error runs were kept). Correspondingly, a comparatively low number of traces was collected for Flexstore and HitList with respect to the other three applications (see Table 1), which allowed us to investigate the robustness of the proposed approach when the number of available traces varies a lot.

Manual navigation was conducted according to a high level functional coverage criterion: all functionalities provided in the top-level application menus have been executed with input data that a user would be typically expected to type. Simulators have been written (by one of the authors) after thorough code inspection and dynamic analysis, so as to properly include all validation checks and dependencies enforced by the original applications. Probabilities of user interaction along specific navigation paths have been estimated from real navigations. The five applications have been instrumented so as to support trace collection.

We have obtained a FSM model of each application by applying state-based abstraction to the execution traces and by manually refining the resulting models (see Figure 2). Specifically, we have used the model inference component [15] of the FITTEST Integrated Testing Environment (ITE). Since this tool is based on dynamic analysis, which is by definition incomplete, a manual refinement phase (conducted by one of the authors) was included, to make sure that the model is a faithful and complete abstraction of the modelled application. Examples of manual refinement actions include the addition of missing transitions or the split/merge of different/equivalent states. Refinement was based on thorough code inspection and on application execution under scenarios specifically selected for model validation.

The size of the client-side models of these five applications ranges between 9 and 19 nodes; 18 and 68 transitions (see Table 1). In our experience, these numbers are quite representative of typical client side models of web applications. The number of paths in these models is unbounded, because of loops, and the derivation of effective test sequences from the models is a non trivial task, if done manually, because of the combinatorial number of possible paths and of the feasibility constraints enforced by the running application (both on the server and on the client side).

`State variable`: *type*	**Abstract values**
`dateSelector`: *String*	Empty, NonEmpty
`dayAtAGlance`: *String*	Empty, NonEmpty
`notes`: *String*	Empty, NonEmpty
`tasks`: *String*	Empty, NonEmpty
`contacts`: *String*	Empty, NonEmpty
`appointments`: *String*	Empty, NonEmpty
`formAttribs`: *Number*	Zero, GreaterThanZero
`formTdCounts`: *Number*	Zero, GreaterThanZero

Table 2: Portion of the state abstraction function used to infer the model of TheOrganizer

Table 2 shows a portion of the state abstraction function used to automatically infer the model of TheOrganizer from its execution traces. In this abstraction, string variables (`dateSelector`, `dayAtAGlance`, etc.) are assigned to two equivalence classes, *Empty* and *NonEmpty*. This means that the exact content of the strings is not taken into account by this abstraction function and the only information being used is the string length (being zero or greater than zero). In a similar way, numeric variables (e.g., `formTdCounts`) are

[2]http://project.cyclos.org
[3]http://www.apress.com/9781590596951
[4]http://www.taskfreak.com/
[5]http://code.google.com/p/hit-list/
[6]http://crest.cs.ucl.ac.uk/ fittest/

grouped into two classes, depending on the stored value being zero or greater than zero.

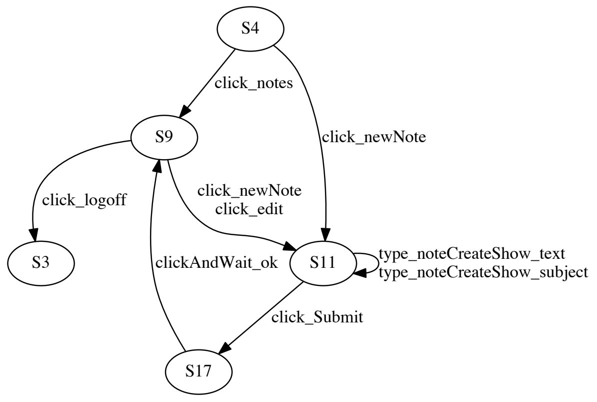

Figure 5: Portion of the FSM model of TheOrganizer

A portion of the model inferred from the execution traces of TheOrganizer is shown in Figure 5. From state S4 it is possible to either click on `notes` and navigate to a page that shows all notes stored so far in the application, or it is possible to create a new note. In state S9, the application displays all previously inserted notes. From this state it is possible to edit an existing note or to add a new note. Both actions lead to state S11, where the subject and the text of the note can be edited/inserted. Note submission leads to state S17 and then back to S9. Logoff is possible in state S9.

The small portion of FSM model shown in Figure 5 contains some examples of infeasible paths. If state S11 is entered after clicking on `newNote`, the user interactions to type the new note *text* and *subject* are mandatory before the new note can be successfully submitted to the application. If not done, the application does not leave state S11 and an error message is displayed to the user. On the contrary, if this state is entered after clicking on `edit`, the form fields containing the note *subject* and *text* are already not empty, which allows the user to perform a submission, moving to state S17, even without typing anything into the *subject* or *text* field of the form (or possibly typing into just one of the two fields). This constraint is not represented explicitly in the model, hence traditional test sequence generators (DFV, BFV, RAND) cannot take it into account. The proposed approach (NGRAM and INTERP) will approximate such constraint through the N-gram statistics and will be more likely to respect it during test sequence derivation.

4.3 Procedure

The test sequence generation algorithms DFV, BFV, RAND, NGRAM and INTERP have been applied to the models of the five subject applications. NGRAM and INTERP have been applied with increasing values of N, until the algorithm performance started to decline or reached a plateau. With NGRAM this happens typically at $N = 4$; with INTERP at $N = 10$. The N-gram statistics was computed using the same traces used for model inference. Since all algorithms are non deterministic, each of them has been run 100 times. Results are averaged over these 100 runs. In all runs, each algorithm was executed with transition coverage set as the adequacy criterion to satisfy.

Metrics SZ and LEN are obtained by measuring the number of test cases and the number of events per test case in each test suite. Metrics COV has been obtained by means of a tool that visits the FSM models of the subject applications based on the feasible input sequences in each test suite, keeping track of the covered transitions. Metrics FEAS has been measured based on a set of manually defined constraints that are satisfied only by valid sequences of actions, that can be successfully executed on the subject applications. Test cases which contain event sequences violating such constraints have been marked as infeasible and have been subtracted from the count of the feasible sequences. They also do not contribute to metrics COV.

An example of constraint that holds for TheOrganizer has been provided in the previous section. Another example of constraint, manually defined for Cyclos, is:

$$[\mathit{amount} > 0 \wedge$$
$$(\mathit{immediate} \vee \mathit{single} \vee (\mathit{multiple} \wedge \mathit{count} > 0))]$$
$$\mathit{click\_submitButton}$$

It is possible to submit a payment if the amount is greater than zero and the payment is immediate or single. Multiple payments require in addition that the recurrence count is greater than zero.

4.4 Results

All experimental data (execution traces, models, test suites) are available to researchers at http://selab.fbk.eu/icse2014.zip, to support replication of the study and comparison with alternative approaches. The tool Ngram-MBT, to derive the test suites from a model using DFV, BFV, RAND, NGRAM and INTERP, is also available online, at: http://se.fbk.eu/technologies.

The values of the four metrics collected in the experiments are reported for the five subject applications in Tables 3, 4. The highest values of FEAS and COV are shown in boldface. They have always been obtained by algorithm INTERP (in one case, Cyclos, NGRAM2 produced the same values). Such maximum FEAS and COV values have been compared for statistical significance against the highest values (shown underlined in the tables) produced by any of the graph visit algorithms (DFV, BFV or RAND). We used the non-parametric Wilcoxon statistical test to assess whether the difference between the means is statistically significant, setting the significance threshold α to 0.01.

On *Flexstore*, INTERP stabilises around the highest FEAS performance when N is 7 or more, while a very high COV value is reached early, even at the initial value of $N = 3$. RAND and DFV have very poor performance. BFV is better, but still it is not comparable with INTERP. NGRAM has also inferior performance. Moreover, such performance is quite sensitive to the choice of N. NGRAM's performance is maximum at $N = 3$ and it starts to decline toward RAND with $N = 4$. Excluding BFV, which produces test suites with many, short test cases, the other test suites contain between 4 and 14 test cases, with a test case length between 25 and 57 events.

On *Cyclos*, NGRAM2 and INTERP (for any value of N) have the best performance, both in terms of FEAS and

Flexstore				
	FEAS	**COV**	**SZ**	**LEN**
DFV	0.02	0.11	4.74	25.34
BFV	0.72	0.65	34.00	3.38
RAND	0.04	0.27	4.26	55.01
NGRAM2	0.39	0.78	4.81	47.73
NGRAM3	0.57	0.81	4.96	49.66
NGRAM4	0.46	0.71	5.07	45.48
INTERP3	0.43	0.86	14.33	54.15
INTERP4	0.58	0.87	13.25	53.41
INTERP5	0.65	0.88	12.28	51.27
INTERP6	0.73	0.88	10.92	55.95
INTERP7	0.79	0.89	9.76	53.63
INTERP8	**0.80**	**0.89**	9.10	55.01
INTERP9	0.80	0.88	8.39	57.18
INTERP10	0.80	0.87	8.03	56.50

Cyclos				
	FEAS	**COV**	**SZ**	**LEN**
DFV	0.53	0.74	5.61	6.22
BFV	0.64	0.72	14.00	2.86
RAND	0.32	0.73	7.36	5.65
NGRAM2	**1.00**	**0.94**	5.61	6.98
NGRAM3	0.36	0.80	7.11	5.39
NGRAM4	0.36	0.78	7.22	5.64
INTERP3	**1.00**	**0.94**	5.73	7.02
INTERP4	**1.00**	**0.94**	5.79	6.92
INTERP5	**1.00**	**0.94**	5.63	6.95
INTERP6	**1.00**	**0.94**	5.71	6.94

TheOrganizer				
	FEAS	**COV**	**SZ**	**LEN**
DFV	0.22	0.23	20.61	11.09
BFV	0.16	0.13	51.00	3.82
RAND	0.00	0.00	18.84	16.48
NGRAM2	0.76	0.95	16.33	16.12
NGRAM3	0.55	0.81	15.81	15.77
NGRAM4	0.38	0.66	15.63	16.39
INTERP3	0.88	0.98	16.18	16.15
INTERP4	0.90	0.98	15.61	17.18
INTERP5	0.93	0.99	15.33	17.67
INTERP6	0.94	0.98	15.27	18.28
INTERP7	0.96	0.99	15.43	17.64
INTERP8	**0.98**	**1.00**	15.19	18.58
INTERP9	0.97	0.99	15.28	18.39
INTERP10	0.97	0.99	15.10	18.67

Table 3: Results obtained for the subject applications: Flexstore, Cyclos, TheOrganizer

COV. NGRAM2 and INTERP achieve the maximum possible FEAS score (1.0), the reason being that in this application bi-grams subsume all feasibility constraints. Coverage reaches also the maximum possible value, since two transitions in the model are infeasible, which makes the maximum achievable coverage equal to 0.94. With tri-grams (NGRAM3) several bi-gram constraints are still met, but there are cases where no tri-gram is applicable (because there is no such triple in the collected execution traces), which results in a random selection of the next event. This explains the lower performance of NGRAM3. In contrast, INTERP interpolates bi-grams for all values of N, resulting in a stable FEAS and COV performance, which remains consistently the highest. Graph visit methods have substantially lower performance. If we exclude BFV, the test suite size and the test case length do not show high variability in this application, being roughly between 5 and 7 (both SZ and LEN).

TaskFreak				
	FEAS	**COV**	**SZ**	**LEN**
DFV	0.72	0.82	7.47	15.20
BFV	0.98	0.98	38.00	3.11
RAND	0.50	0.92	6.99	33.26
NGRAM2	0.88	0.99	7.38	39.68
NGRAM3	0.90	1.00	7.43	39.64
NGRAM4	0.90	0.99	7.73	34.15
INTERP3	0.88	0.99	7.60	39.32
INTERP4	0.91	1.00	7.39	40.05
INTERP5	0.94	1.00	7.28	38.31
INTERP6	0.96	1.00	7.25	40.82
INTERP7	**0.99**	**1.00**	7.07	40.58
INTERP8	0.98	1.00	7.44	40.45
INTERP9	**0.99**	**1.00**	7.22	39.46
INTERP10	**0.99**	**1.00**	7.13	39.80

HitList				
	FEAS	**COV**	**SZ**	**LEN**
DFV	0.20	0.28	6.66	11.49
BFV	0.07	0.12	15.00	5.40
RAND	0.07	0.40	3.61	32.37
NGRAM2	0.46	0.85	3.84	31.78
NGRAM3	0.38	0.90	3.29	34.78
NGRAM4	0.19	0.60	3.38	31.39
INTERP3	0.68	0.93	4.11	24.85
INTERP4	0.70	**0.97**	3.98	24.94
INTERP5	0.75	0.96	3.99	24.85
INTERP6	0.66	0.95	3.65	26.58
INTERP7	**0.76**	**0.97**	3.78	24.08
INTERP8	0.70	0.95	3.82	25.10
INTERP9	0.75	0.95	3.82	23.49
INTERP10	0.75	0.94	3.67	25.00

Table 4: Results obtained for the subject applications: TaskFreak, HitList

On *TheOrganizer*, there is a dramatic gap between graph visit strategies and N-gram based algorithms, with the former exhibiting extremely poor performance as compared to the latter. The reason is that in this application a specific authentication protocol must be followed initially to leave the start state and to exercise the application in depth. At the same time, such protocol is not enforced explicitly in the model. As a result, test sequences that do not take into account any interaction ordering constraint remain stuck in the initial application state. Thanks to the availability of the N-gram statistics, INTERP can easily escape such trap state and can exercise the subject application in depth, achieving high feasibility and coverage. For INTERP, metrics COV is high even at $N = 3$, while FEAS tends to increase at a quite constant rate while N increases. NGRAM2 performs also pretty well, but its performance degrades quickly at increasing N, making it quite difficult to select the right N for a given application (in fact, $N = 3$ works well for Flexstore, while for Cyclos and TheOrganizer $N = 2$ is much better). Test suite size and test case length (BFV excluded) are respectively between 15 and 20, and between 11 and 18.

On *TaskFreak*, INTERP achieves the highest FEAS and COV values, but BFV is not far from such performance. The absolute difference between INTERP7 and BFV is small (0.01 for FEAS and 0.02 for COV). However, such small difference is statistically significant (at level 0.01) according to the Wilcoxon test. This means that such small difference is not due to fluctuations associated with the non deterministic nature of the algorithms. On the contrary, INTERP has consistently a slightly higher performance. Even though

BFV might be considered perfectly acceptable in this case, it exhibits so much variability across applications that its use cannot be recommended in general. On the contrary, this does not happen with INTERP. On this application, the performance of NGRAM is quite high, but still substantially lower than INTERP. If we exclude BFV, there is not much variability of the test suite size SZ (between 6 and 7), while the test case length has a minimum with DFV (15) and then ranges between 33 and 40.

On *HitList*, the highest performance is reached by INTERP7, with a large margin over the best graph visit performance (DFV for FEAS and RAND for COV). NGRAM2 has also good performance in terms of coverage, but as N varies the performance of NGRAM changes substantially, while that of INTERP remains stable and predictable. It is interesting to notice that despite the extremely low number of traces (5) used to learn the N-gram statistics, the performance of NGRAM and INTERP are definitely superior to those of the other algorithms, showing that the N-gram based approach is robust also with respect to the number of traces used to compute the N-gram statistics. Moreover, despite the low number of traces used to compute the N-gram statistics, most of the test cases generated by NGRAM (99.59%) and by INTERP (99.75%) differ from the event sequences in the initial five traces (NGRAM/INTERP regenerate just one of the five initial test sequences, among the 976/1631 test sequences generated in total). Excluding BFV and DFV, the test suite size is around 3-4 test cases, each with a length between 23 and 32 events.

All differences between boldface and underlined values in Tables 3, 4 have been found to be statistically significant al level $\alpha = 0.01$ according to the Wilcoxon two-sided non-parametric statistical test.

Based on the results presented above, we can positively answer **RQ1** and **RQ2**: *interpolated N-gram based test sequence generation produces a higher proportion of feasible event sequences and achieves higher coverage than plain N-gram and graph visit approaches.*

In terms of test suite size and test case length (**RQ3**, **RQ4**), we can notice that BFV produces generally test suites with a lot of very short test cases (see Tables 3, 4). The other methods are not very different from each other. INTERP produced slightly longer and more numerous test cases than the other techniques on Flexstore, while on the other subjects there is no remarkable difference between INTERP, NGRAM and RAND in terms of test suite size and test case length.

In summary, the test sequences generated by means of interpolated N-grams achieve the highest feasibility and coverage scores, outperforming all alternative test case derivation techniques. Moreover, the performance of interpolated N-grams is stable across applications and for various choices of the parameter N, which is not true for the graph visit and plain N-grams algorithms. Interpolated N-grams have stable performance also when the number of traces available is reduced dramatically. The interpolated N-grams test suite size and test case length are in line with those of the other techniques and look generally acceptable for the tester. The extra cost associated with the N-gram method is limited to the collection of the data necessary to reliably compute the N-gram statistics. This cost is application dependent, but it was always acceptable in the five cases considered in this experiment.

The interpolated N-grams approach has stable performance (across subject applications, number of training traces and value of parameter N) and achieves the highest feasibility and coverage scores.

4.5 Qualitative analysis: TheOrganizer

Let us consider the transition `"S11 -> S17 [click_Submit]"` in the model of TheOrganizer (see Figure 5). To cover this transition, note *text* and *subject* must be filled in before submitting the form, if the note is a new one, while there is no pre-condition to be satisfied for edited notes.

If we look at the NGRAM2 statistics, we can notice that event `click_newNote` is never followed directly by `click_Submit`. Hence, the probability of generating a test case that creates a new note and submits it without filling in *text* or *subject* is zero. On the contrary, graph visit algorithms may (non deterministically) choose such event sequence. Actually, in the NGRAM2 statistics, event `type_noteCreateShow_text` follows `click_newNote` with probability one, so it will be always chosen as successor of `click_newNote`.

In the NGRAM2 statistics, event `type_noteCreateShow_text` is followed by `type_noteCreateShow_subject` with probability 79%, while it is followed by `click_Submit` with probability 21%. In fact, the NGRAM2 statistics does not have enough context to distinguish the case where `type_noteCreateShow_text` is preceded by `click_newNote` from the case where it is preceded by `click_edit`. To capture such longer dependency, the NGRAM3 statistics is needed.

In the NGRAM3 statistics the two events `click_newNote`, `type_noteCreateShow_text` are never followed by `click_Submit`, hence event `click_Submit` has zero probability in the NGRAM3 test sequences, if preceded by *new note* creation and *text* typing (without *subject* typing). In this case, a context of length two (i.e., the 3-tuples statistics) is enough to choose only feasible successors of the current state. In other cases (e.g., *new account* creation), a longer context is needed.

Let us consider the length 3 context (i.e., the NGRAM4 statistics) needed to handle correctly the feasible sequences involved in *new account* creation. In the NGRAM4 statistics, the prefix `"clickAndWait_image, clickAndWait_image, click_newNote"` has zero occurrence count. As a consequence, when this path prefix occurs, the next event being generated by NGRAM4 is chosen with uniform probability among the three possible successors (see state S11 in Figure 5). This means that there is a 33.3% chance of generating an infeasible sequence (by choosing `click_Submit` as successor event). On the contrary, when INTERP4 is applied, since the current (length 3) path prefix has zero probability, it does not contribute to the selection of the successor events. Both the length 2 path prefix `"clickAndWait_image, click_newNote"` and the length 1 path prefix `"click_newNote"` are followed by `type_noteCreateShow_text` with probability 1. Correspondingly, with this path prefix INTERP4 generates only feasible event sequences (i.e., INTERP4 chooses `type_noteCreateShow_text` with probability 1), while NGRAM4 has 33.3% probability of generating an infeasible event sequence.

4.6 Threats to validity

The main threats that affect the validity of the empirical study described above are the authors' bias and the external validity threats.

Authors' bias: The authors have been involved in a number of tasks carried out to complete the experiment and the way in which such tasks were actually conducted might have influenced the results. Specifically, the authors have collected the traces used in the experiment, have refined the models inferred from the traces and have manually defined the feasibility constraints. Some trace collection was carried out either manually or through a simulator. When done manually, it was executed with functional coverage in mind, so as to have a clear goal to achieve, in terms of what execution scenarios to exercise and when to stop tracing. When done through simulation, the simulator code was carefully developed so as to take into account all validity checks implemented in the real applications. Realistic probabilities (obtained from observations of real executions) are used in the simulator to decide which interaction the simulated user will activate in each application state. Manual model refinement was guided by the presence of mismatches between the modelled behaviours and the actual ones, as apparent from code inspection and execution. Feasibility constraints have been obtained through code inspection and by executing the application to see what validation checks are performed at runtime. Since the connection between the manual activities conducted by the authors and the performance of NGRAM and INTERP is quite loose and difficult to predict a-priori, we think that our expectations on the performance of the proposed approach had no relevant influence on the manual activities. However, we recognise that replication in a setting where the authors are not directly involved in any activity would be beneficial to confirm our findings.

External validity: We have validated our approach on *five* web-based applications, for which the client side behaviour was abstracted into a model used for test sequence derivation. Moreover, the model was first inferred from execution traces and then refined manually. We expect similar results to hold for applications in the same domain (web) for which a similar modelling is carried out, although we recognise that five is a small number and additional subjects would be needed to corroborate our findings, even in this specific domain. Generalisation beyond the domain of web applications and beyond the kind of models that we used should be done with extreme care. For that, additional experiments are definitely required.

5. RELATED WORKS

Model based testing is a research area that amounts to a huge literature, which was surveyed, among others, in the paper by Dias Neto et al. [9]. It has been applied in a number of different domains and contexts [11, 27, 6, 18, 21, 5, 15].

It is possible to distinguish between design time models and inferred models. In case of a design-time model, abstract test cases can be produced to check how the expected behaviours (coming from the model) have been implemented in the application [6, 18]. Hence, the role of behaviour specifications is to provide abstract test cases, test oracles, and a measure of the test adequacy [21]. Instead, the role of an inferred model is mainly to provide abstract test cases (e.g., [5, 6, 15]), to be instantiated in executable test cases by adding inputs and, eventually, test oracles (e.g., [15]).

In both cases, instantiation of abstract test cases derived from models is an expensive activity and the presence of infeasible test cases demands for substantial effort and knowledge from the tester's side [9], who is required to recognise infeasible sequences and to filter them out or replace them (if the target level of adequacy is to be preserved).

While approaches exist which try to balance the degree of over/under approximation of inferred models [22] and metrics have been proposed to evaluate the generalisation capability of inferred models [26], to the best of our knowledge, this is the first work which tries to address the problem of test case infeasibility, by adopting a test case derivation strategy (based on the N-gram statistics) that aims at increasing the likelihood of feasibility and correspondingly the level of coverage actually achieved. This paper extends our previous workshop paper [24] with a novel algorithm (INTERP), which has been shown to outperform the previous one (NGRAM) and to exhibit a more stable and predictable (high) performance level. We also extended the empirical evaluation done for the workshop paper with three additional subjects.

6. CONCLUSIONS AND FUTURE WORK

We have proposed a technique for model based test case derivation which takes advantage of the interpolated N-gram statistics to address the infeasibility problem of model based testing. By selecting the next events for a test sequence according to the N-gram statistics we increase the likelihood of generating a feasible sequence. Correspondingly, we also increase the level of model coverage achieved. Interpolation of the N-gram statistics is required as N increases to compensate for missing N-tuples at large values of N.

We have implemented the proposed technique in the tool Ngram-MBT, which is publicly available as open source code. The experimental results, including models, traces and test suites, obtained on five public domain applications are also publicly available for future empirical research in the field. Results indicate that test cases generated according to the interpolated N-gram statistics outperform plain N-grams and graph visit methods for model based test case derivation. Moreover, the interpolated N-grams method has consistently high performance across different applications and does not depend critically on the choice of the parameter N. This method is also robust with respect to the number of traces available.

Our future work will be devoted to further experimentation of the proposed method on additional subjects and with alternative interpolation schemes. In particular, applicability to domains different from web applications and to models not necessarily inferred from execution traces (using state abstraction) would increase the generality of the obtained results and would corroborate our findings.

Acknowledgements

This work has been funded by the European Union FP7 project FITTEST (grant agreement n. 257574).

7. REFERENCES

[1] A. Andrews, J. Offutt, and R. Alexander. Testing Web Applications by Modeling with FSMs. *Software and System Modeling, Vol 4, n. 3*, pages 326–345, 2005.

[2] Anneliese Amschler Andrews, Jeff Offutt, and Roger T. Alexander. Testing web applications by modeling with fsms. *Software and System Modeling*, 4(3):326–345, 2005.

[3] A. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. on Computers*, 21(6), 1972.

[4] Lionel C. Briand and Yvan Labiche. A UML-based approach to system testing. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 194–208. Springer-Verlag, 2001.

[5] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Pasareanu, Robby, and Hongjun Zheng. Bandera: Extracting finite-state models from java source code. In *Proceedings of the International Conference on Software Engineering*, pages 439–448, 2000.

[6] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Sebastian Hack, and Andreas Zeller. Generating test cases for specification mining. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 85–96. ACM, 2010.

[7] Valentin Dallmeier, Christian Lindig, Andrzej Wasylkowski, and Andreas Zeller. Mining object behavior with ADABU. In *Proceedings of the 2006 international workshop on Dynamic systems analysis*, pages 17–24, 2006.

[8] Valentin Dallmeier, Christian Lindig, Andrzej Wasylkowski, and Andreas Zeller. Mining object behavior with ADABU. In *Proc. of the International Workshop on Dynamic Analysis (WODA)*, pages 17–24, Shangai, China, May 2006.

[9] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches: a systematic review. In *Proc. of the International Workshop on Empirical Assessment of Software Engineering Languages and Technologies (co-located with ASE'07)*, WEASELTech '07, pages 31–36. ACM, 2007.

[10] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing*. Pearson Prentice Hall, 2007.

[11] Y.G. Kim, H.S. Hong, D.H. Bae, and S.D. Cha. Test cases generation from UML state diagrams. *Software, IEE Proceedings -*, 146(4):187 –192, aug 1999.

[12] Ivo Krka, Yuriy Brun, Daniel Popescu, Joshua Garcia, and Nenad Medvidovic. Using dynamic execution traces and program invariants to enhance behavioral model inference. In *ICSE (2)*, pages 179–182, 2010.

[13] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. Automatic generation of software behavioral models. In *30th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, May 2008.

[14] Chris Manning and Hinrich Sch utze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[15] Alessandro Marchetto, Paolo Tonella, and Filippo Ricca. State-based testing of ajax web applications. In *Proc. of IEEE International Conference on Software Testing (ICST)*, pages 121–131, April 2008.

[16] A. Mesbah, A. van Deursen, and D. Roest. Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering*, 38(1):35–53, 2012.

[17] Cu D. Nguyen, Alessandro Marchetto, and Paolo Tonella. Combining model-based and combinatorial testing for effective test case generation. In *Proc of the ACM International Symposium on Software Testing and Analysis (ISSTA)*, pages 100–110, 2012.

[18] Jeff Offutt and Aynur Abdurazik. Generating tests from UML specifications. In Robert France and Bernhard Rumpe, editors, *UML 99 : The Unified Modeling Language*, volume 1723 of *Lecture Notes in Computer Science*, pages 76–76. Springer Berlin / Heidelberg, 1999.

[19] Mauro Pezzè and Michal Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2007.

[20] Muhammad Shafique and Yvan Labiche. A systematic review of model based testing tool support. Technical Report Technical Report SCE-10-04, Carleton University, Canada, May 2010.

[21] Phil Stocks and David Carrington. A framework for specification-based testing. *IEEE Trans. Softw. Eng.*, 22:777–793, November 1996.

[22] Paolo Tonella, Alessandro Marchetto, Cu D. Nguyen, Yue Jia, Kiran Lakhotia, and Mark Harman. Finding the optimal balance between over and under approximation of models inferred from execution logs. In *Proc of the Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 21–30, 2012.

[23] Paolo Tonella, Cu D. Nguyen, Alessandro Marchetto, Kiran Lakhotia, and Mark Harman. Automated generation of state abstraction functions using data invariant inference. In *Proceedings of the 8th International Workshop on Automation of Software Test (AST)*, 2013.

[24] Paolo Tonella, Roberto Tiella, and Cu D. Nguyen. N-gram based test sequence generation from finite state models. In *Proceedings of the 1st Future Internet Workshop (FITTEST)*, 2013.

[25] C. D. Turner and D. J. Robson. The state-based testing of object-oriented programs. In *Proc. of the Conference on Software Maintenance (ICSM)*, pages 302–310, Montreal, Canada, September 1993. IEEE Computer Society.

[26] N. Walkinshaw, K. Bogdanov, C. Damas, B. Lambeau, and P. Dupont. A framework for the competitive evaluation of model inference techniques. In *Proceedings of the International Workshop on Model Inference in Testing (MIIT)*, 2010.

[27] Xun Yuan and Atif M. Memon. Using GUI run-time state as feedback to generate test cases. In *Proc. the International Conference on Software Engineering (ICSE)*, pages 396–405, Washington, DC, USA, May 23–25, 2007. IEEE Computer Society.