Improved Usage Model for Web Application Reliability Testing

Gregor v. Bochmann, Guy-Vincent Jourdan, Bo Wan

School of Information Technology & Engineering, University of Ottawa, Ottawa, Canada
{ bochmann, gvj, bwan080}@ site.uottawa.ca

Abstract. Testing the reliability of an application usually requires a good usage model that accurately captures the likely sequences of inputs that the application will receive from the environment. The models being used in the literature are mostly based on Markov chains. They are used to generate test cases that are statistically close to what the application is expected to receive when in production. In this paper, we study the specific case of web applications. We present a model that is created directly from the log file of the application. This model is also based on Markov chains and has two components: one component, based on a modified tree, captures the most frequent behavior, while the other component is another Markov chain that captures infrequent behaviors. The result is a statistically correct model that exhibits clearly what most users do on the site. We present an experimental study on the log of a real web site and discuss strength and weakness of the model for reliability testing.

Keywords: Web applications, Usage models, Reliability testing, Markov chains

1 Introduction

Many formal testing techniques are directed towards what is sometimes called "debug techniques": the goal is to fulfill some given criteria (branch coverage, all-uses coverage, all paths, all code and many others), or uncover every fault[1] using some restricted fault models (checking experiments). However, in practice, non-trivial applications are simply not expected to ever be failure-free, thus the purpose of a realistic testing campaign cannot be to find all the faults. Given that only some of the failures will be uncovered, it only makes sense to question which ones will be found by a testing method. Note that in a realistic setting, failures are always ranked by importance.

[1] In this document, a "fault" in the application source code leads to a "failure" at execution time. A "test sequence" is a sequence of interactions between the testing environment (e.g. the tester) and the tested application. "Test input data" is the data that is input during the execution of the test sequence. A test sequence, valued with test input data, is a "test case". A test case may uncover a failure, due to one (or more) fault.

In this paper, we are interested in testing the reliability of an application. For a material system, reliability is usually defined by the expected time of operation after which the system will fail. In the case of a software system, it can be defined by the expected number of usages before it will fail. A usage, in this context, may be a request provided by the environment, or a complete usage session, for instance in the case of an application with an interface to a human user. Clearly, the occurrence of a failure of a software system is dependent on the input provided. In order to test the reliability of an application, it is therefore important to apply inputs that reflect the behavior of the environment of the application in the normal operating conditions. This is sometimes called "operational testing". In this context, it is important to test first those behavior patterns that occur most frequently under normal operating conditions. This idea has been applied with great success on certain large software projects: Google was for example able to deliver an internet browser, Chrome, that was remarkably reliable from its first release, not necessarily because it was tested against more web pages than the other browsers, but because it was tested against the web pages that Google knew people were most looking at[2].

There are essentially two methods for obtaining a realistic model of the behavior of the environment of the application to be tested for reliability:

1. Environment model based on the application model: If the functional behavior requirements of the application are given in the form of an abstract model, for instance in the form of a UML state machine model, this model can be easily transformed into a model of the environment by exchanging input and output interactions. However, for obtaining an environment model useful for reliability testing, this functional model must be enhanced with statistical performance information about the frequency of the different inputs applied to the application in the different states of the environment. This may be formalized in terms of a Markov model based on the states of the abstract functional application model.
2. Environment model extracted from observed execution traces in a realistic environment: Independently of any model of the application that may be available, a model of the dynamic behavior of the environment may be extracted from the observation of a large number of execution traces that have occurred in a realistic setting.

In this paper, we pursue the second approach. We assume that the application to be tested is a Web application. We make the assumption that after each input by the user, the response from the web server provides information about the functional state of the application. In the case of traditional web applications, the state information is given by the URL of the page that is returned. In the case of Rich Internet Applications (RIA), we consider that the content of the returned web page, that is the DOM of this page, represents the application state, assuming that there is no hidden state information stored in the server.

In previous work on reliability testing, the user model is usually either given in the form of a tree of possible execution sequences with associated probabilities for each branching point [1][2], or in the form of a Markov model [3],[4],[5],[6]. We show in

[2] See http://www.google.com/googlebooks/chrome/, page 10 for a graphical illustration.

this paper how one can extract, from a given set of execution sequences, a user model that is a combination of an execution tree and a traditional Markov model. We first construct the execution tree from the given set of execution sequences. The upper branches of the tree have usually been executed a large number of times which means that good statistical information is available for the branching probabilities. This part of our model is called the “upper tree”. For the lower branches of the tree, however, there are usually only one or a few executions that have been observed; therefore the statistical information about branching probabilities is very weak. We therefore remove these lower branches and combine them into a Markov model for which the branching probabilities are obtained from the union of all the lower branches. This part of our model is called “lower Markov model”. Our resulting user model is therefore a Markov model which contains two parts, the “upper tree” and the “lower Markov model”.

The “lower Markov model” is a traditional (first-order) Markov model where each state of the model corresponds to a state of the application. However, the “upper tree” is a higher-order Markov model which may contain several different Markov states corresponding to the same application state; this is the case when the behavior of the user does not only depend on the current application state, but also on the path which was taken to get to this state. In contrast to other statistical modeling methods starting out with observed execution sequences that can only model dependencies on previous states up to a limited number of interactions [7],[8] our “upper tree” can model dependencies on previous application state for arbitrarily long state sequences.

This paper is structured as follows. Section 2 presents an overview of Markov usage models and their application in reliability testing, followed by a brief review of previous work on Markov usage models in Web applications. Section 3 presents a detailed description of our hybrid Markov usage model. In Section 4, we present the results of experiments conducted with real data. In Section 5, we give our conclusions and present our plans for future research.

2 Review Markov Usage Model

Markov models are commonly used to model usage patterns and to establish reliability estimations because they are compact, simple to understand and based on well-established theory. K. Goseva-Popstojanova and K. S. Trivedi used Markov renewal processes to estimate software reliability [9],[10], but they did not use the usage model. Markov chains have been used extensively over the past two decades in the domain of statistical usage testing for software. In 1994, Whittaker and Thomason [3] explained how to use a Markov-chain-based model of software usage to perform statistical testing. Random walks on the model are performed to generate test sequences. These test sequences are applied to the implementation and the test experiment is run until enough data is gathered for the implementation under test. More recently, MaTeLo, an industrial tool also used Markov chains to model usage profiles, generate test cases, debug and estimate software reliability [4], [5], [11].

In Web applications reliability testing, a Markov chain model can be constructed from log files. When users visit a Web site, Web servers record their interactions with the Web site in a log file. The log file usually contains data such as the user's IP address, viewing time, required page URL, status, and browser agent. After some massaging of the data, it is possible to infer from the log files a reliable set of user sessions (see e.g. [12],[13],[14] for detailed explanations of how to obtain sessions from Web log files). Figure 1(a) illustrates the principle of building a Markov chain from log files. In this example, the Web pages being visited (identified in practice by their URLs) belong to the set {1, 2, 3, 4, 5}. In the following, we call such pages "application states". From the log files, visiting sessions have been reconstructed. Artificial starting state S and terminating state T are added to theses sessions for simplicity. Some of the reconstructed sessions may be identical if several users have followed the same sequence of pages on the Web site. We combine such sessions going through the same sequence of application states, to obtain what we call "application state sequences". In the figure, the column Nb shows how many times each application state sequence was followed. Figure 1.b presents the traditional Markov chain model for these sessions. In the Markov chain, the edges are labeled with probabilities representing the distribution of the user's choice for the next state from the current one.

Application state sequences	Nb.
S-1-3-5-T	4
S-3-4-T	2
S-2-3-4-T	1
S-2-3-5-T	3

(a)

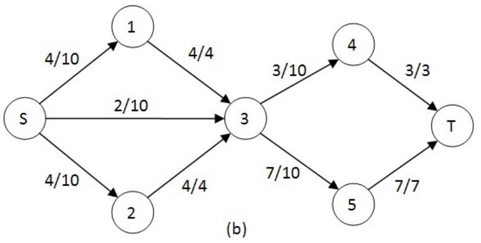

Fig. 1. An example of a traditional Markov Chain model: (a) a collection of application state sequences and (b) the corresponding traditional Markov chain model

Traditional Markov models are simple and compact, but they have also limitations when used to model usage profiles. In Web applications, a traditional Markov model, sometimes called *first-order* Markov model, captures the page-to-page transition probabilities: *p(x2|x1)* where *x1* denotes the current page and *x2* denotes one of pages reachable from *x1*. Such low order Markov models cannot capture behavior where the choice of the next page to be visited depends on "history", that is, on how the current application state was reached. For example, in an e-commerce site, after adding an item to the shopping card, the user would typically either "proceed to checkout" or "continue shopping". The probability of doing one or the other is certainly not identical after adding one item, after adding two items etc. Another example is shown in Figure 1: In the snippet of a traditional Markov usage model (b), we see that there are three ways to reach state 3 (from state 1, from state 2 and from state S), and that from state 3, there is 30% chance to go to state 4, and 70% chances to go the state 5. However, looking at the provided application state sequences, we can see that users reaching state 3 from state 1 never go to state 4 afterwards. What is shown in the traditional Markov chain is misleading. Since it is reasonable that most Web applications involve such history-dependent behavior, accurate models of user behavior cannot be obtained with first order Markov chains [15]. The same problem is also discussed by Deshpande and Karypis [16]. Thus, a good usage model requires higher-order Markov chains.

A higher-order Markov model has already been explored by Borges and Levene in 2000 to extract user navigation patterns by using a Hypertext Probabilistic Grammar model structure (HPG) and N-grams [7]. In their work, an N-gram captures user behavior over a subset of N consecutive pages. They assume that only the N-1 previous pages have a direct effect on the probability of the next page selected. To capture this, they reuse the concept of "gram" taken from the domain of probability language learning [17]. Consider, for example, a web site composed of six states {*A1, A2, A3, A4, A5, A6*}. The observed application state sequences are given in Table 1 (Nb denotes the number of occurrences of each sequence).

Table 1. A collection of application state sequences

Application State Sequences	Nb
A1-A2-A3	3
A1-A2-A4	1
A5-A2-A4	3
A5-A2-A6	1

A bigram model is established using first-order probabilities. That is, the probability of the next choice depends only on the current position and is given by the frequency of the bigram divided by the overall frequency of all bigrams with the same current position. In the example of Table 1, if we are interested in the probabilities of choices from application state A2, we have to consider bigrams (sequences including two application states) that start with state A2. This includes the following: Segment A2-A3 has a frequency of 3, and other bigrams with A2 in their current position include the segments A2-A4 and A2-A6 whose frequency are 4 and 1, respectively; therefore, *p(A3|A2)=3/(3+4+1)=3/8*. It is not difficult to see that the 2-gram model is a first-order Markov chain, the traditional Markov usage model. The second-order model is obtained by computing the relative frequencies of all trigrams, and higher orders can be computed in a similar way. Figure 2 shows the 3-gram model corresponding the sessions in Table 1.

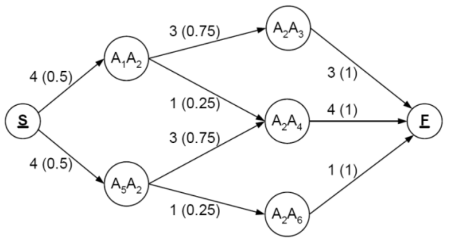

Fig. 2. 3-gram model corresponding to the sessions given in Table 1

Subsequently, the same authors showed in 2004 how to use higher-order Markov models in order to infer web usage from log files [8]. In this paper, they propose to duplicate states for which the first-order probabilities induced by their out-links diverge significantly from the corresponding second-order probabilities. Take Table 1 again as example. Consider state 2 and its one-order probability *p(A3|A2)=3/8*, and its two-order probability *p(A3|A1A2)=3/4*. The large difference between *p(A3|A2)* and *p(A3|A1A2)* indicates that coming from state *A1* to state *A2* is a significant factor on the decision to visit *A3* immediately afterwards. To capture this significant effect, they split state *A2* as illustrated in figure 3. A user-defined threshold defines how much the first and second order probabilities must differ to force a state splitting. A k-means clustering algorithm is used to decide how to distribute a state's in-links between the split states.

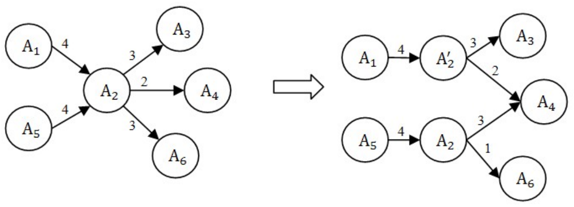

Fig. 3. An example of the cloning operation in dynamic clustering modeling

All these approaches focus on the last N-1 pages and will thus ignore effects involving earlier pages visited. In addition, the fixed-order Markov model also has some limitations in accuracy as pointed out by Jespersen, Pedersen and Thorhauge [18].

3 Hybrid Tree-Like Markov Usage Model

In this section, we introduce a new method to infer a probabilistic behavioral model from a collection of sessions extracted from the logs of a Web application. The model draws from both a traditional Markov chain usage model and a tree of application state sequences which is introduced in the next Section. The new usage model contains a modified tree of state sequences that captures the most frequent behaviors, and a traditional Markov chain model recording infrequent behavior.

Table 2. A collections of Application state sequences

Application State Sequence	Nb
S-1-1-3-5-T	1
S-1-3-2-1-2-4-T	4
S-1-3-2-2-4-T	9
S-2-3-4-2-2-4-T	4
S-2-3-4-4-T	21
S-2-3-4-2-3-4-T	14
S-3-3-4-2-4-T	23
S-3-3-4-2-T	4
S-3-3-4-4-T	33
S-3-2-2-3-4-T	4
S-3-2-2-5-T	4
S-3-2-4-5-T	4
S-3-2-4-3-5-T	4
S-3-T	2

The usage model is built from a collection of user sessions, to which we add a common starting and terminating state. Again, different users can go over the same application states during their sessions. We group sessions in "application state sequences", as discussed above. Table 2 shows an example, along with the number of times each state sequence occurs in the log files.

3.1 Building the Tree of Sequences

The tree of sequences (TS for short) is constructed from the given state sequences by combining their longest prefix. Each node in the tree, called model state, corresponds to an application state (e.g. the URL of the current page), but each application state has in general several corresponding model states. The tree captures the application state sequence that was followed to reach a given model state; this "history" corresponds to the path from the root of the tree to the model state. This is unlike the traditional Markov chain usage model where there is a one-to-one mapping between application states and model states. In addition, each edge of the tree is labeled with the number of application state sequences that go over this particular branch. Figure 4 shows the TS model built from the state sequences listed in Table 2. One major advantage of the TS model is that it can be used to see the conditional distribution of the next state choice based on the full history. For example, the probability of choosing state 2 from state 4 after the state sequence 2-3-4 is *p(2|2-3-4)=18/39* while the probability of choosing state 4 under the same circumstances is *p(4|2-3-4)=21/39.*

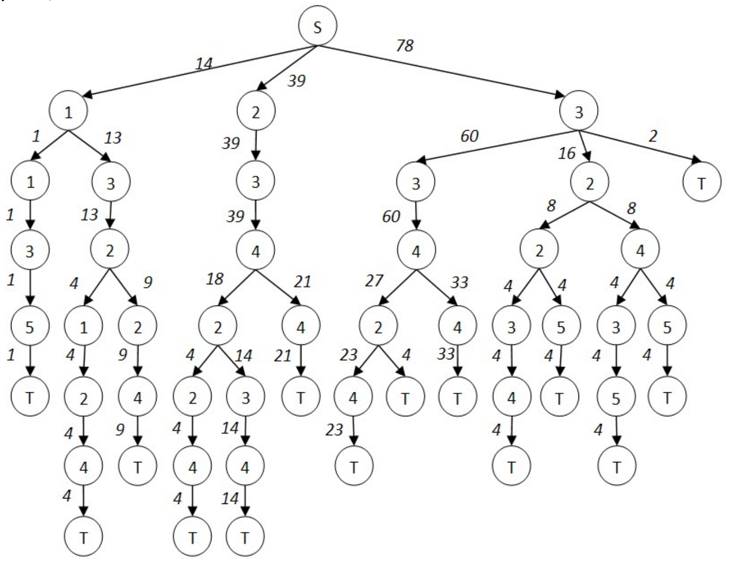

Fig. 4. The tree of sequences captured from Table 2

Despite its strengths, the TS model has many weaknesses. One major disadvantage is the fact that the probabilities calculated for each transition might not be reliable if the state sequences have not been followed very often. Since the tree tends to be very wide there are many such sequences. And long sequences tend to be less and less representative the longer they go (that is, many users may have followed a prefix of the sequence, but few have followed it to the end). In addition, we mention that (a) it tends to be fairly large, (b) it is not adequate for creating new test cases, and (c) it does not pinpoint common user behavior across different leading state sequences.

3.2 Frequency-pruned TS model

To overcome some of the problems of the TS model, we first apply a simple technique that we call "frequency pruning". This is based on the observation that model states that occur with low frequency in the application state sequences do not carry reliable information from a statistical point of view. We note that, for such states, the estimation of the conditional probabilities will not be reliable [16]. Consequently, these low frequency branches can be eliminated from the tree without affecting much the accuracy of the model. However, just applying such pruning would impact the coverage that can be inferred from the model, since it removes some low frequency but still very real branches. To avoid this problem, we do not discard these pruned branches, instead we include the corresponding state sequences in the "lower Markov model" (introduced below), which is a traditional Markov usage model.

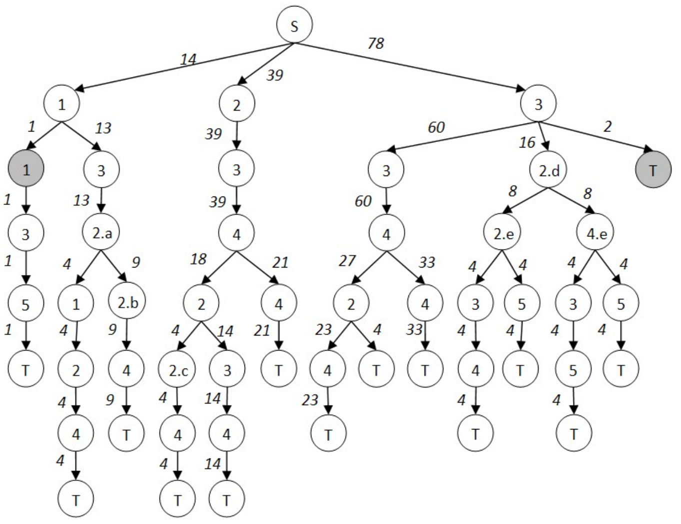

Fig. 5. Frequency-pruned TS Model

The amount of pruning in the TS model is controlled by a parameter, called "frequency threshold" θ. When the calculated conditional probability of a branch is lower than the frequency threshold, the branch is cut. Figure 5 shows the result of pruning of the TS model of figure 4, with θ set at 10%. For example, we have $p(1|1)=1/14 < \theta$, therefore the branch 1-3-5-T is cut from the tree and will be used when building the "lower Markov model". The grey nodes in figure 5 represent access point from the TS model to this Markov model.

3.3 Hybrid Markov Usage model

Our goal is to strike the right balance between the traditional Markov chain model and the TS model. We want to have separate model states for the instances of applications states when the user behavior is statistically different, and we want to merge them into a single model state when the user behavior cannot be statistically distinguished (be it that the users behaves identically or that we do not have enough information to make the difference). Working from the two previous models, one could start from the traditional Markov usage model and "split" states for which a statistically different behavior can be found depending on the history, or one could start from the TS model and "merge" states that are instances of the same application state for which no significant behavior difference can be found (or even iterate on merges and splits).

In this paper, we work from the frequency-pruned TS model and merge states. The goal is thus to look at different model states which represent the same application state and decide whether the recorded user behavior is statistically significantly different. If so, the states must be kept apart, and otherwise the states are candidates to be merged.

3.4 Independence Testing and State Merging

As shown in the example of Figure 7, the TS model contains in general many model states that correspond to the same application state. For instance, the states 4.a, 4.b and 4.c all correspond to the application state 4. To simplify the user behavior model, we would like to combine such states in order to reduce the number of states in the model. However, this should only be done if the user behavior is the same (or very similar) in the different merged states. We therefore have to answer the following question for any pair of model states corresponding to the same application state: Is the recorded user behavior statistically significantly different on these two states? In other words, is the users' behavior dependant on how they have reached this application state? – If the answer is yes, then the model states should be kept separated, otherwise they should be merged. An example is shown Figure 6 (a) where the statistical user behavior is nearly identical in the two states 1.a and 1.b, and these two states could therefore be merged leading to Figure 6 (b).

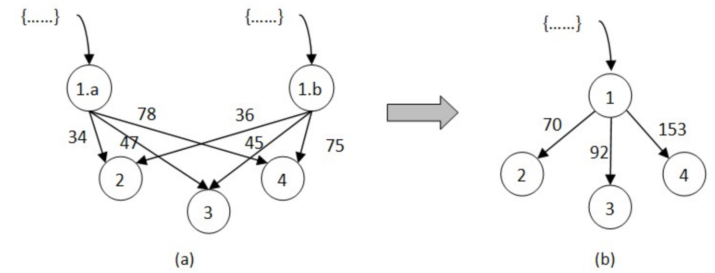

Fig. 6. An example of merging two model states

We note that the model states that are the successors of the states to be merged must be identical for the two states to be merged, as shown by the example of Figure

6. This implies that the merging operations must be applied from the bottom-up through the original tree-like TS model. We note that the terminal states labeled T can be merged (because they have the same user behavior). However, the model states that precede the final state, and many of the other preceding states, have only few occurrences in the application state sequences inferred from the log files. Therefore the statistical significance of these occurrences is not so strong; therefore a decision for merging is difficult to make.

There are a number of statistical methods to answer to these questions. We will use in the following the so-called "test of independence", itself based on the chi-square test (see for instance [19]). However, we note that the test of independence gives only reliable answers when there is enough statistical information. The so-called "Cochran criterion" states that in order to apply the test of independence, at most 20% of the possible alternatives should have fewer than six instances in the sample set. As discussed above, many of the model states in the lower part of the TS model will fail the Cochran criterion and thus cannot be used for the test of independence since they do not carry enough information to be statistically significant.

We therefore propose to merge into a single model state all TS model states that correspond to a given application state and do not satisfy the Cochran criterion. These merged states form what we called "lower Markov model" in the Introduction. For our running example of Figure 5, we obtain after the application of the Cochran criterion the model of Figure 7. The grey nodes form the "lower Markov model". For example, the model states 2.a, 2.b, 2.c, etc of Figure 5 were merged into the state 2.e of Figure 7. For state 2.d in Figure 5, for instance, there are two choices to go to state 2.e or to state 4.e with the frequencies of 8 and 8 respectively. They are represented in Figure 7 as state 2.d to state 2.e or state 4.e with frequencies 8 and 8, respectively.

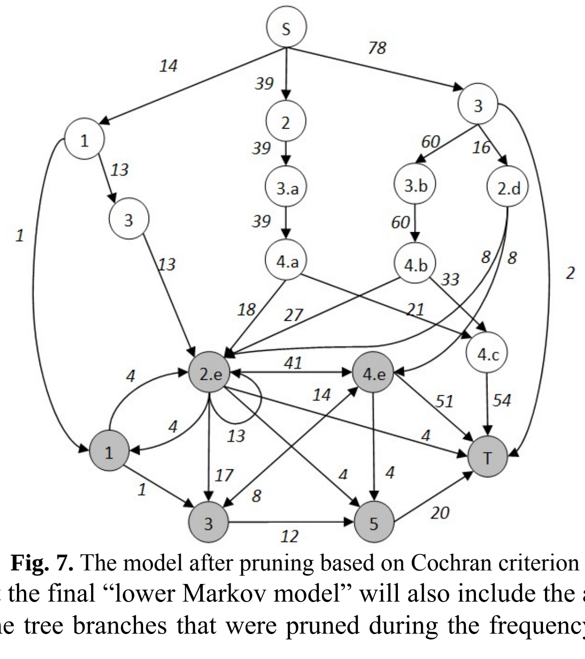

Fig. 7. The model after pruning based on Cochran criterion

We note that the final "lower Markov model" will also include the application state sequences of the tree branches that were pruned during the frequency-pruning phase described in Section 3.2. The frequency-pruning steps is applied first to avoid running into situations in which a model state traversed by a large number of application state transitions still fails the Cochran criterion because a few very infrequent behavior have been observed there (for example because of sessions created by web crawler).

After applying the Cochran criterion and constructing the "lower Markov model", we check the remaining model states in the "upper tree" for the possibility of merging by applying the chi-square-based independence test in a bottom-to-top order. We apply this test pairwise, even if there are more than two model states corresponding to the same application state.

The value of chi-square indicates how good a fit we have between the frequency of occurrence of observations in an observation sample and the expected frequencies obtained from the hypothesized distribution. Assuming that we have k possible observations and have observed $o_i(i = 1, \dots k)$ occurrences of observation i while the expected frequencies are $e_i(i = 1, \dots k)$, the value of chi-square is obtained by Formula (1)

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \qquad (1)$$

χ^2 is a value of a random variable whose sampling distribution is approximated very closely by the chi-square distribution for (k-1) degrees of freedom [19].

Let us consider the example shown in Table 3 below. It shows the observed choices to application states 2 and 4 from the model states 4.a and 4.b (see Figure 7). If we assume that these two model states can be merged, that is, the branching probabilities to states 2.c and 4.c is almost identical, we can calculate these branching probabilities by considering the union of all observed sessions going through states 4.a and 4.b (see last row in the table). This leads to the expected number of choices indicated in the last two columns of the table. For instance, the probability of choosing application state 2 is 45/99, and therefore the expected number of choices of state 2 from model state 4.a is 45/99 * 39 = 17.73.

Table 3. Example of chi-square calculation

	Observed occurrences			Expected occurrences	
Next State	4.a	4.b	total	4.a	4.b
2	18	27	45	17.73	27.27
4	21	33	54	21.27	32.73
total	39	60	99	39	60

Then we use the numbers in the table to calculate χ^2 according to formula (1) for model states 4.a and 4.b and take the average. The result is the χ^2 value that we can use to determine whether our hypothesis is valid for a given confidence level, using a table of the chi-square distribution for one degree of freedom. In the case of our example, we get a χ^2 value of 0.0124. Since this value is smaller than $\chi^2_{0.05}$= 3.841 we can say with confidence level of 95% that the model states 4.a and 4.b represent the same user behavior, and the states can be merged, as shown in Figure 8.

We apply such merging tests to all pairs of model states that correspond to the same application state and that have outgoing transitions to the same set of model states[3]. This is done from the bottom of the "upper tree" towards its root. The states at the bottom have transitions that lead to states of the "lower Markov model", which are already merged, thus the test can always be applied to these bottom states. If these bottom states are merged, then the test can be applied to their parents and this keeps going until the test fails. As already explained, one example of two states that can be merged is states 4.a and 4.b (see Figure 7 and Figure 8). Once they are merged, some states higher in the tree may become candidates for merging. In this example, the parent nodes of 4.a and 4.b, namely nodes 3.a and 3.b, respectively, can also be merged (because they have only one successor which is the same). Once all candidates for merging have either been merged or are determined not to satisfy the merging condition, then we obtain our final performance model, as shown for our example in Figure 8.

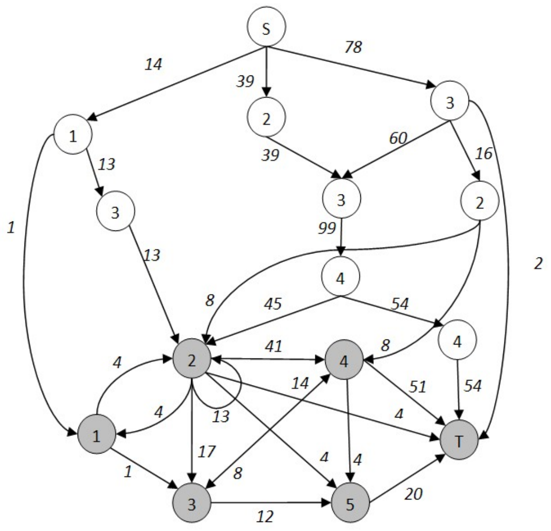

Fig.8. Hybrid Markov model constructed by sessions in table 2

4 Experiment

We experimented our approach on a web site called Bigenet (http://www.bigenet.org). Bigenet is a genealogy web site allowing access to numerous registers – birth, baptism, marriage, death and burials – in France. Two international exchange students, Christophe Günst and Marie-Aurélie Fund, developed a tool which is able to generate a list of visiting sessions from the access log files of the web server and the functional model of the application. The tool follows the approach presented in [12]. We had at our disposal the access log files for the period from September 2009 to September 2010. Table 4 presents a summary of the characteristics of the visiting sessions during this period.

[3] If some model states are reached by only one of the two states being tested, we assume that the other state also reaches to the same states but with a probability 0.

Table 4. Summary Statstic for the data set from Bigenet.

Characteristics	Bigenet
Num. of Application States	30
Num. of Request	900689
Num. of Sessions	108346
Num. of Application State Sequences	27778
Ave. Session length	8.3132
Max. Session length	301

We have developed a second tool that implements the model construction approach described in Section 3. It creates the TS model from the list of application state sequences inferred from the reconstructed sessions, and then performs the pruning, Cochran merging and state merging based on independence tests. The TS model constructed from the whole year of visiting sessions is very large, containing 348391 nodes in the tree. Figure 9 shows the TS model based on 1000 visiting sessions.

Table 5. Summary of experimental results

	All	Set-1	Set-2	Set-3	Set-4
Num. of states in TS	348391	38652	38463	38336	42039
Num. of states after frequency pruning	81364	12493	11796	12438	13066
Num. of states in "lower Markov model"	30	30	29	30	29
Num. of states in "upper tree" before merging	426	78	79	76	82
Num. of states in "upper tree" after merging	337	65	68	65	68
Num. of independence tests applied	108	17	15	15	18
Num. of mergings performed	89	13	11	11	14
Execution time without optimization[4]	3937ms	313ms	328ms	297ms	328ms

Table 5 shows the results of our analysis of these 108346 user sessions (see column labeled "All"). The following parameters were used during our analysis: (a) the pruning threshold was 5%; (b) the confidence level for the independence test was 95%.

We note that frequency-pruning and Cochran criteria leads to a user performance model that has a very much reduced "upper tree" and a "lower Markov model" that corresponds to the states of the application which in this case contains 30 states. The merging of non-independent states in the "upper tree" leads in our case to a further reduction of 20% of the model states. Most of the applied tests for merging succeeded. The "upper tree" is shown Figure 10.

[4] We coded all algorithms in NetBeans 6.9.1 and performed experiments on a 2.53GHz Intel Core 2 P8700 laptop computer with 2.93 GB RAM.

Table 5 also shows similar results for several smaller sets of user sessions that were selected randomly from the original visiting sessions. Each subset has 10000 sessions. The results obtained for the different subsets of sessions are very similar to one another. The number of model states in the "upper tree" is smaller than in the case of all sessions, since the Cochran criterion removes more states from the TS model because of the lower number of observations. As can be seen, the model that we obtain is quite stable with different sets of sessions. The number of states in the obtained user models varies little for the different subsets of sessions. Since the "upper tree" part of the model is the most interesting, we show in Figure 11 the "upper trees" for the sets of sessions Set-1 and Set-2. One can also see that these trees closely resemble the upper part of the "upper tree" for all sessions, as shown in Figure 10. Due to the difficulty of defining a feature space and measurement method, we do not discuss the similarity between the re-constructed web sessions and the real data in this paper.

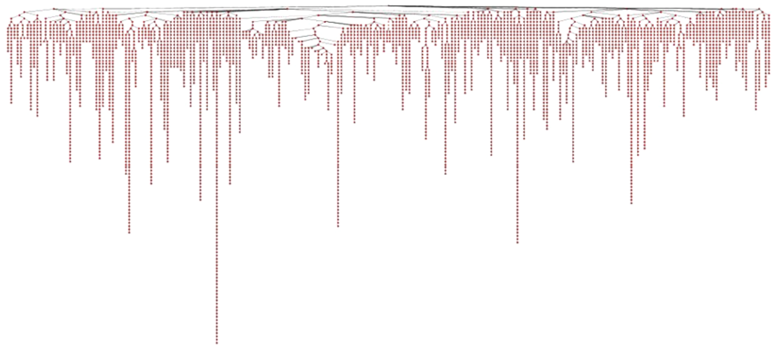

Fig. 9. The TS model created by 1000 visiting sessions.

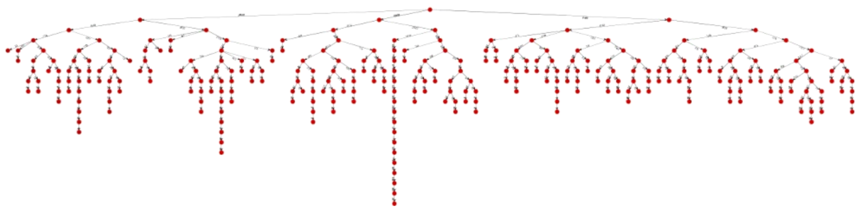

Fig.10. The upper tree part of usage model, down from originally 348,391 states.

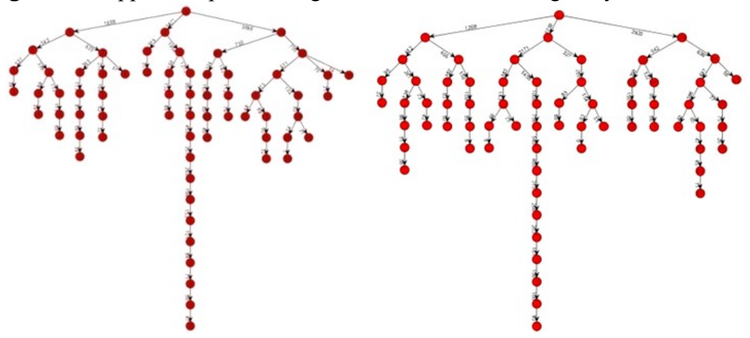

Fig. 11. Upper-tree part of usage models generated from Set-1 and Set-2

5 Conclusion and Future Work

In this paper, we have presented a method that can be used to create an accurate statistical usage model for Web applications. This model is created from the application log file, and can be used for reliability testing. Our method uses a tree structure to preserve statistically significant information on user behavior, as gathered from the log files. The initially very large tree is reduced in three steps: first, frequency pruning removes the branches that are almost never followed. Then, a test called Cochran criterion is used to remove states that do not carry reliable statistical information. States removed during these two steps are merged into a traditional Markov chain model (the "lower Markov chain") that captures infrequent behaviors. The pruned tree is further reduced through merging of model states corresponding to the same application states and on which user behavior is statistically similar. The test for similarity is a classical test of independence, and the resulting "tree", which we call the "upper tree", contains the most frequent behaviors, which are statistically significant. In our experiments, the resulting hybrid Markov usage model is drastically smaller than the original tree of sequences, but still contains all the significant behavioral and coverage information.

This improves on lower-order Markov usage models that contain usually all the application states but cannot capture the user behavior accurately since they have no concept of history. In our model, in the upper tree the entire history is preserved. Other history-preserving higher-order Markov models, such as N-grams, exist but come with their own set of limitations. For N-Grams, they do retain history of length N-1, but cannot capture sessions of length less than N [16].

Our model still has shortcomings. A main one is its inability to identify some of the common behavior, if the behavior occurs on branches that must be kept apart because they lead to statistically different behavior lower down in the tree. Indeed, our state merging process tends to merge states that are mostly toward the bottom of the tree. To overcome this, we are planning to use either extended Finite State Machine models or hierarchical models, in order to merge parts that are statistically identical but are included inside larger sequences that are not. One may also improve the model by introducing some history dependence in the "lower Markov model" by using, for instance, the N-gram approach. The other major shortcoming of our current model is that user inputs are not captured. The model must be enhanced to accommodate for a statistically accurate representation of the inputs and their relation with the followed path.

Acknowledgments: The authors wish to thank Marie-Aurélie Fund and Christophe Günst for their work on this project. This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.

Reference

1. S.A. Vilkomir, D.L. Parnas, V.B. Mendiratta and Eamonn Murphy, "Segregated Failures Model for Availability Evaluation of Fault-Tolerant Systems" 29th Australasian Computer Science Conference. Vol. 48 (2006)

2. W.Wang and M.Tang,f "User-Oriented Reliability Modeling for a Web System," in Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03), pp.1-12 (2003).
3. J.A. Whittaker and M.G. Thomason, "A Markov Chain Model for Statistical Software Testing," *IEEE Trans. Software Eng*., Vol. 20, No. 10, pp.812–824.(1994)
4. H. Le Guen, R Marie, and T Thelin. "Reliability Estimation for Statistical Usage Testing using Markov Chains". In ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering, pages 54-65, Washington, DC, USA. IEEE Computer Society.(2004)
5. W. Dulz, F. Zhen, "MaTeLo—statistical usage testing by annotated sequence diagrams, Markov chains, and TTCN-3", In Proceedings of Third International Conference On Quality Software (QSIC'03), IEEE, (2003).
6. Kirk Sayre. Improved Techniques for Software Testing Based on Markov Chain Usage Models. PhD thesis, University of Tennessee. (1999)
7. Borges, J. "A Data Mining Model to Capture User Web Navigation." PhD thesis, University College London, London Uiversity, (2000).
8. Borges, J and Levene, M: "A dynamic clustering-based Markov model for web usage mining". In CoRR:the computing research repository. cs.IR/0406032 (2004).
9. K.Goseva-Popstojanova and K.S.Trivedi, "Failure Correlation in Software Reliability Models", IEEE Trans. on Reliability, Vol.49, pp. 37-48.(2000)
10. K.Goseva-Popstojanova, M.Hamill, "Estimating the Probability of Failure When Software Runs Are Dependent: An Empirical Study," 20th International Symposium on Software Reliability Engineering, issre, pp.21-30, (2009)
11. A. Feliachi and H. Le Guen, "Generating transition probabilities for automatic model-based test generation", Third International Conference on Software Testing, Verification and Validation, pp. 99-102 (2010)
12. R. Cooley, B. Mobasher, and J. Srivastava, "Data Preparation for Mining World Wide Web Browsing Patterns," Knowledge and Information Systems, vol. 1, no. 1, Feb. 1999, pp. 5–32.(1999)
13. J. Pei et al., "Mining Access Patterns Efficiently from Web Logs," Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Springer-Verlag, New York, 2000, pp. 396–407.(2000)
14. K.W. Miller, et. al., "Estimating the Probability of Failure When Testing Reveals No Failures", IEEE Transactions on Software Engineering, Vol 18, pp 33-42.(1992)
15. Peter L.T. Pirolli and James E. Pitkow: "Distributions of surfers' paths through the world wide web: Empirical characterizations." World Wide Web pp. 29–45. (1999)
16. M. Deshpande, G. Karypis, "Selective Markov Models for Predicting Web-Page Accesses", in Proc. of the 1st SIAM International Conference on Data Mining, (2001)
17. Charniak, E. Statistical Language Learning. The MIT Press, Cambridge, Massachusetts. (1996)
18. Jespersen, S, Pedersen, T. B, and Thorhauge, J: "Evaluating the markov assumption for web usage mining", In Proceeding of the Fifth International Workshop on Web Information and Data Management (WIDM'03), pp. 82-89 (2003)
19. Ronald E.Walpole and Raymond H.Myers, "Probability and Statistics for Engineers and Scientists", Fifth Edition, published byMacmillan publishing company (1993)