Improved Usage Model for Web Application Reliability Testing

Bo Wan

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science

University of Ottawa
Ottawa, Ontario, Canada
May 2012

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 thesis, we propose a model for reliability testing that is created directly from the log file of a web application. Our proposed model is also based on Markov chains and has two components: one component, based on a modified tree, captures the most frequent behaviors, while the other component is another Markov chain that captures infrequent behaviors. The result is a statistically correct model that shows clearly what most users do on the site.

The thesis also presents an evaluation method for estimating the accuracy of various reliability-testing usage models. The method is based on comparison between observed users' traces and traces inferred from the usage model. Our method gauges the accuracy of the reliability-testing usage model by calculating the sum of goodness-of-fit values of each traces and scaling the result between 0 and 1.

Finally, we present an experimental study on the log of a real web site and discuss the way to use proposed usage model to generate test sequences, as well as strength and weakness of the model for reliability testing.

Acknowledgment

Very special appreciation to my supervisors, Dr. Guy-Vincent Jourdan and Dr. Gregor v. Bochmann, for the guidance, encouragement and patience they always gave to me. Without their knowledge and help, this thesis would not have been completed. I also want to extend my sincere appreciation to Emre Dinçtürk, Suryakant Choudhary and Seyed M. Mirtaheri. Although we did not work in the same group, their help and advice in my research and life are very important to make me complete this thesis.

I would like to thank National Science and Engineering Research Council (NSERC) of Canada. This project would not be possible without their support. I am also grateful to Marie-Aurélie Fund and Christophe Günst for their work on this project

In addition, I would also like to thank my family for their support. Very special thanks to my mother and to my father who made possible for me to achieve this. Finally, my thanks to my girlfriend for the encouragement and support she has given to me.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Motivation

Many formal testing techniques are oriented towards what is sometimes called "debug techniques": the goal is to fulfill some given criteria (branch coverage, all-uses coverage, all paths and many others), or uncover any fault[1] within 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. It is also desirable to have some control over the type of failures being uncovered. Note that in software reliability testing, failures are always ranked by their frequency. We cannot expect to detect the most frequent failures by using these formal methods if they are blind to software operation environment.

In this project, we are interested in testing the reliability of web applications. In order to test the reliability of an application, it is important to apply inputs that reflect the

[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.

behavior of the environment of the application in the normal operating conditions, so that the failures in most frequently used scenarios can be detected quickly. And these frequent failures impact user-experience more than other failures related to scenarios used infrequently. Therefore, from the software reliability point of view, they are more important. The Formal models most often used in the formal testing context (such as classical FSMs) are efficient at capturing all possible system's behaviors, but cannot capture the respective likelihood of these behaviors in an operational environment. Although various strategies have been explored (all edges, all-uses, all paths etc.) and test input data generation is usually done by sampling the input space based on selected coverage criterion (partition testing …), our intent is to gear these choices of test cases towards testing in priority scenarios that are more likely to happen. Since our target is testing software reliability, our aim is to generate test cases to simulate the running environment of the software, instead of discovering all faults. Google made a good example of this strategy with its web browser Chrome. Chrome was remarkably reliable from its first release, not necessarily because it was tested against more web pages than other browsers, but because it was tested against the web pages that Google knew people were most looking at[2]. Hence, testing software following users' usage is an important and interesting research topic.

An accurate usage model will result in a testing system that generates a set of test cases, in which scenarios refer to the real use. According to the probabilistic distribution presented by the usage model, frequent scenarios are tested more often than less frequent ones. As discussed in the paragraph above, such usage models are helpful for formal testing, as well as for reliability estimation. Therefore, the first goal of this thesis is to devel-

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

op an accurate usage model for statistical usage testing and reliability testing. In fact, there are several existing usage models (such as Markov Chain Usage Models) that can be used to generate sets of test cases. However, can these usage models capture the respective likelihood of behaviors in an operational environment? Are the generated test case sets similar to the usage of real user? All these questions should be answered before a usage model is applied for testing. In other words, in order to carry on the project, a fair estimation method is necessary. Hence, the second objective of this thesis is to propose a simple and effective approach which has the ability to measure the accuracy of a stochastic usage model, as well as the similarity between generated test scenarios and behaviors observed from real users.

1.2 Thesis Contributions

Our research activities focus on creating an accurate usage model while maintaining a user defined model size; and given a user defined model size, creating a usage model that is accurate. In order to balance accuracy and model size, chi-square tests and frequency pruning are applied to merge duplicated states to compress the model. Furthermore, we also propose an effective and simple model evaluation method to estimate the accuracy of a usage model based on covariance analysis between observed user's operations and user's behaviors implied by the model. This method also can be used to optimize the parameters in parameter-based usage models. Finally, we have generated abstract test sequences which present all usage patterns from user with a distribution that corresponds to the occurrence of these patterns. In a nutshell, our target is to solve a se-

ries of problems in software reliability testing in order to make reliability testing practical.

This thesis contains the following contributions:

1. We created a new usage model based on Markov Chains that overcomes infeasible test cases, a weakness of first-order Markov Chain Usage Models that have been widely used in statistical usage testing. Compared with some other popular usage models in the fields of data mining and pattern recognition, our proposed model also has some advantages (such as model size,accuracy and coverage) as discussed in Chapter 3.

2. We modeled a collection of user web visiting sessions as a frequency tree that stores the users' visiting histories when they browse the web application. Given enough observations, the frequency tree can alternatively be viewed as a probabilistic model that represents higher-order probabilities corresponding to the users' preferred visiting trails. We made use of chi-square tests as a criterion to determine the merging of duplicated states and branches in order to achieve the required trade-off between the model size and model accuracy. The approach was also presented at the conference ICTSS 2011 [1].

3. We provided a method to measure the accuracy of a usage model compared to the users' real usage histories. This is useful to overcome the practical questions encountered in reliability testing such as selecting usage models and op-

timizing parameterized usage models. Our method evaluates the accuracy of a model by taking into account the covariance differences between the observed transition visiting frequencies and the model-implied transition visiting frequencies, given all possible users' web visiting trails. In order to reduce the deviation of the assessment, we use the K-fold cross-validation testing strategy. The estimation method will be presented at the conference COMPSAC 2012 [2].

Our work can be of interests to software testing, quality assessment and product improvement. For software testing, testers can use our work to uncover the most frequent failures thus improving the end user experience. For quality assessment, engineers can use our work to generate more likely test cases to acquire more accurate reliability assessment. For product improvement, designer can use our work to find out the user's usage patterns which is helpful for designing more user-friendly software.

1.3 Thesis Outline

This thesis is organized as follows: Chapter 2 provides the background of works which are related to our research. Chapter 3 demonstrates the procedure of building a hybrid tree-like Markov usage model. In this chapter, we also explain how we make use of the independent test to merge similar states in the model to trade off model size and model accuracy. In Chapter 4, an estimation method for usage model accuracy is presented. We also discuss the statistical logic of the approach and the rationality of K-fold valida-

tion strategy to control bias of estimated accuracy. In Chapter 5, the implementation of the prototype for reliability testing (which is based on the approaches discussed in Chapter 3 and Chapter 4) is discussed. Through several experiments, Chapter 6 illustrates the performance of the hybrid tree-like Markov usage model, as well as the generated test sequences by walks on the model following model indicated distributions. Finally, the thesis ends with a conclusion and discussion of future work in Chapter 7.

Chapter 2 Background

2.1 Reliability Testing

For a physical system, reliability is usually defined by the expected time of operation after which the system will fail. In the case of software systems, a widely accepted concept of "Software Reliability" was first defined by Musa, Lannino and Okumoto [5] as "the probability of failure-free operation of a computer program in a specified environment for a specified time". Software quality is one of the most important aspects in Software Engineering. Nowadays, software is applied everywhere in finance, military and manufacturing. Even a small fault can lead to a big disaster. Therefore, when we discuss a system's reliability, we cannot ignore the influence of software. Martin Shooman delivered system reliability as $R_{sys} = R_h * R_s * R_o$[3], where R_h is hardware reliability; R_s is software reliability and R_o is human operator reliability. Due to different methodologies, the methods of software reliability estimation are developed in two different ways, reliability growth estimation models and statistical usage testing models.

2.1.1 Reliability Growth Estimation

The main question of software reliability is the probability of successful usage. If

we ignore the upgrading of the hardware environment, software, as logic or design, is always assumed to have perfect durability because it can't "wear out". If the software system does not have any logic mistake when it is released, then as long as the computer on which the software runs is reliable, the system will not yield any failure. That is, after software system release, the reliability of the software will not change as time passes. Normally, faults are introduced during the implementation phase. After the implementation phase, the reliability of the software system may be very poor. Therefore a subsequent testing campaign is carried out, and the more failures are uncovered and fixed, the higher the reliability will be. Hence, the software reliability can be presented as a function that is correlated positively with the testing time. Based on these considerations, Shooman, Musa, Okunoto etc. [4], [5], [6] used mathematic models to simulate the error-removal process in order to estimate software reliability.

Under a correct test strategy, normally more failures are detected and fixed at the beginning of a test campaign. If we split the whole testing phase into several time slots, as testing goes on, the number of failures found during a time slot decreases over time. Shooman defined the decay of the error-removal rate. To simulate the testing process, he assumed that the error-removal rate decreases exponentially [4]. Hence, as testing phase tends to infinity, the number of failures detected in a time slot tends to zero; and the number of cumulative detected faults tends to the real number of faults in the system under test. He defined the error-removal rate as follows, where τ is testing time, α is a constant number and E_T is the initial error-removal rate.

$$E_r(\tau) = E_T e^{-\alpha\tau} \qquad (1)$$

He assumed that the event of users encountering a remaining failure in the system is Poisson process. Then the reliability of software after the testing period is:

$$R(t) = e^{-\int k(E_T e^{-\alpha\tau})dt} = e^{-k(E_T e^{-\alpha\tau})t} \qquad (2)$$

In 1992, Musa and Okunoto proposed their reliability estimation model [7] based on the idea that software testing should be done according to the operational profile. In this case, the early fault corrections have greater effect on the failure rate. For example, let us assume that we have a large-scale software system with several modules. Module A has high probability of execution, say 80%, but module B has low probability of execution, say 1%. Let us assume that module A and module B both have defects. Since module A is far more frequently executed, module A will more likely trigger a failure and lead to the correction of a fault. Hence, module B has less impact on the software reliability. With this condition, they added three assumptions: (1) The software is operated under similar condition than the anticipated operational profile; (2) failures are independent from each other; (3) the failure rate decreases exponentially with test time. Based on these assumptions, they gave the software reliability:

$$R(t) = (1 + \lambda_0 \theta t)^{-\frac{1}{\theta}} \qquad (3)$$

where λ_0 is the initial failure rate; and the θ is failure rate decay rate. Both parameters are computed by maximum likelihood estimation.

Although reliability growth models provide a way to estimate the software system's reliability based on a mathematical simulation test of the process and usage profile,

the method is quite complicated and many conditions and assumptions make the performance of assessment and prediction quite questionable word in practice.

2.1.2 Statistical Usage Testing

Statistical usage testing is used to validate software based on the intended usage, and provides reliability measurements as well. In statistical testing of software, all possible uses of the software, at some level of abstraction, are represented by a statistical model wherein each possible use of the software has an associated probability of occurrence [8]. Since it is very difficult to specify all possible test cases, a model of the operation environment of system under test is required. Such a model is sometimes called a usage model. Usually, a usage model is constructed from a sample population of possible uses corresponding to a usage probability distribution. Then test cases are generated from the usage model and run against the software under test. 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 from 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.

One of the important benefits of statistical usage testing is the estimation of the failure rate and reliability of the software. As discussed before, software reliability relies heavily on the user's behavior of the environment of the application in normal operating conditions. For instance, assume a system under test has a number of functions that work correctly and one function that performs incorrectly. If the incorrect function is rarely used, few failures will be triggered, that is, the system is quite reliable even though it has some errors. On the other hand, if the incorrect function is frequently used, the system will exhibit a large number of failures which result in a low reliability. Therefore, a usage model which could capture the user's behaviors is essential for software reliability estimation based on statistical usage testing.

The typical process of performing statistical usage testing is as follows [9]:

Get Specifications: Some specification documents or design documents are required to develop the usage model. For web applications, it may be web topology document, UML diagram, web access log files (observation of user's usage patterns), etc.

Develop the Model Structure: The states and state transitions connecting the states are identified according to the specifications. i.e., the states can be derived from the web pages of the web topology structure or the services the web server provides. And transitions can be identified from web access log files.

Assign Probabilities: The state transition probabilities of the usage model are assigned manually or following a set of constraints and calculated to satisfy the test goal [10]. In this thesis, we assign probabilities of each transition based on higher-order probabilities as required.

Verify and Analyze the Model: After developing the model structure and setting probabilities on each transition, the usage model is ready for statistical usage testing. However, since the usage model plays an important role, verifying and analyzing the correctness of the usage model is unavoidable. Analytical results are calculated to aid in test planning and verifying that the model properly represents the expected use of the software [11]. This thesis also proposes a measurement method to evaluate different usage models based on covariance analysis.

Run Random Tests: test cases are generated by usage model and executed on the

software under test. Since the test cases generation follows the probabilities on each transition of usage model, the set of test cases should be similar to the operation environment of the system.

Analyze the Testing Result: The testing record contains information such as which test cases were run against software, where failures were observed, the frequency of failure occurrence and function coverage. These results can be used to verify and validate the software, to establish a strategy to improve the usability and quality of the software, and to estimate the software reliability. And reliability estimator is such a tool which can be use to estimate software reliability through analysis testing result.

2.1.3 Reliability Estimator

2.1.3.1 Laplace Rule of Succession

During software testing, whenever the software being tested fails, it is repaired and resubmitted for testing. We assume that the random test strategy is carried out and the test case population follows the software operational environment profile. Then we have the question of how to make a realistic estimation of the probability of failure when testing reveals no failures or few failures?

In probability theory, Pierre-Simon Laplace first introduced and discussed the problem of succession. If we repeat an experiment that yields two results, success or failure, after n independent experiments with s successes, what is the probability that the

next experiment will again be a success? Since the experiments are independent, the probability of success in the next repetition equals to the probability of success in the whole population. More abstractly: $x_1, x_2, \dots, x_n$ are conditionally independent random variables that each can assume the value *0* or *1*. Let us assume that through observation, there are *s* variables with value *1*. Then, if we know nothing more about them, the probability of the next independent random variable to leading to success is:

$$p(x_{n+1} = 1 | x_1, x_2, \dots, x_n = s) = p(s) = \frac{s+1}{n+2} \qquad (4)$$

Laplace introduced this rule of succession by illustrating a sunrise problem. Assuming that the sun has risen every day for the past 5000 years, he used the rule of succession to calculate the probability that the sun will rise the next day. He set θ as the probability of sunrise. Then the observed joint distribution of the sample $f(x_1, x_2, \dots, x_n; \theta)$ is that with probability sun rose everyday in the past 5000. Working along with Bayesian estimation, he calculated posterior distribution $f(\theta | x_1, x_2, \dots, x_n)$. Furthermore, he estimated the prior distribution $f(\theta)$. To review mathematical details, please refer to Appendix A.

2.1.3.2 Miller's Reliability Evaluator

Recently, Miller's reliability estimator attracted the interest of many researchers in software testing. Miller's reliability estimator is based on the rule of succession to estimate the software failure rate (prior distribution) $f(\theta)$ when random testing reveals no failure or few failures. Simply said, if we consider the whole software or a software component as a bin, each test is a "draw" from the bin; when the "draws" reveal no failure or

few failures, intuitively, we know that the software's failure rate is low. Miller made use of the rule of succession to quantify this intuitive definition. Here, θ is the software failure rate, $x_1, x_2, \ldots, x_n$ are n independent random tests. Each test can be a single test case or a test scenario composed of several sequential test inputs. As explained Appendix A, using the maximum likelihood function and Bayesian analysis, the failure rate distribution $f(\theta)$ follows a standard beta distribution [12]. Then the expected value of the failure rate of the system under test is given by Formula (5) where s is the number of successful tests runs, f is the number of failures, and a and b are parameters representing prior information about the software failure rate. In the case that no prior information about the software failure rate exists, we have $a = b = 1$.

$$E(\hat{\theta}) = \frac{f + a}{f + s + a + b} \qquad (5)$$

Thus, the expected value of the system reliability is

$$E(R) = 1 - E(\hat{\theta}) = 1 - \frac{f + a}{f + s + a + b} \qquad (6)$$

and the variance of the reliability of the system under test is

$$Var(R) = \frac{(f + a)(s + b)}{(f + s + a + b)^2(f + s + a + b + 1)} \qquad (7)$$

Miller's work provides a profound insight into software reliability estimation. Yet, the estimator's performance is of course restricted by the simulation model of the operational environment, in other words, the "draws". Markov models are commonly used to

simulate the operational environment of a system in order to establish reliability estimation.

2.2 Markov Chain Usage Models

A finite Markov chain, [13] [14], models a stochastic process which is characterized by a finite set of states. The stochastic process transfers from one state to another; and the probability of a transition to the next state depends only on the present state. The Markov chain model can be presented as a directed graph with transition probabilities labeled on arcs or a matrix of transition probabilities. A *kth*-ordered Markov Chain (or a Markov chain with memory *k*) is a process satisfying the conditional probability $Pr(X_n|X_{n-1}X_{n-2} \dots X_{n-k})$ for n>k. In other words, the future state depends on the past *k* states; and the order *k* is determined by the conditional probability. Markov chain 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. Markov chains have been used extensively over the past two decades in the domain of statistical usage testing for software.

2.2.1 First-order Markov Chain Usage Models

In reliability testing for web application, 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, requested 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 Chapter 6 section 1 for detailed explanations of how to obtain sessions from Web log files). Table 1 and Figure 1 illustrate the principle of building a Markov chain usage model (MCUM for short) 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. The artificial starting state S and terminating state T are added to theses sessions for simplicity. Some of the reconstructed sessions are 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 table, the column "Number of sessions" shows how many times each application state sequence was followed.

Table 1 A collection of application state sequences for first-order Markov chain usage model

Application State Sequences	Number of Sessions.
S-1-3-5-T	8
S-3-4-T	2
S-2-3-4-T	3
S-2-3-5-T	7
S-3-4-1-3-5-T	4

Figure 1 presents the first-order 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. For example, the probability

$p(4|3) = \frac{9}{28} \approx 32.14\%$ on the arc from state 3 to state 4 captures the fact that after visiting page 3, 32.14% of users visited page 4.

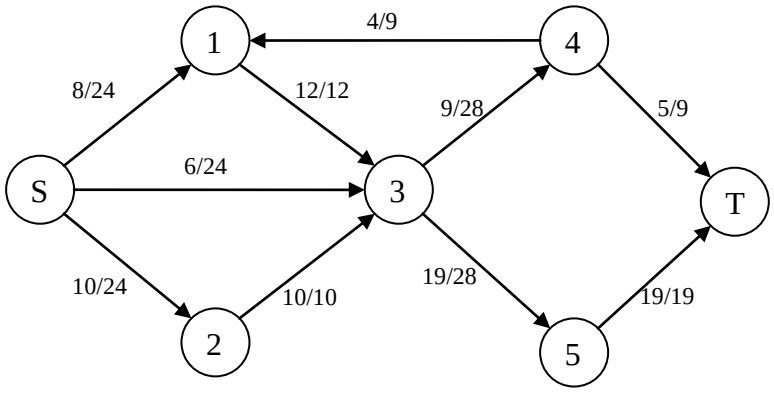

Figure 1 The first-order Markov chain usage model corresponding to Table 1

Markov chains and Markov chain based stochastic processes have been studied for over half a century. There are many variations. Here we define the above usage model as first-order Markov chain model since it characterizes the probability of the user's behavior based on the user's current state. In this example, the model has the set of states {S, T, 1, 2, 3, 4, 5}. Let w_i represent the number of times the page corresponding to state *i* was visited; let w_{ij} be the number of times the user traverses to state *j* from state *i*. Then the probability $p_{ij} = \frac{w_{ij}}{w_i}$ illustrates the users' behaviors based on user's current state. Therefore, we call p_{ij} the first-order transition probability. The MCUM built on first-order transition probability is a first-order Markov chain usage model. Furthermore, let w_{ijk} be the number of times users go to state k from state *i* by passing through state *j*. Then the second-order transition probability $p_{ijk} = \frac{w_{ijk}}{w_{ij}}$ represents the users' behaviors

depending on the current state and also the previous state. The Markov chain usage model constructed on such probabilities is called a second-order Markov chain. As the order grows, more user “memory” (previous *k* states the user visited) is introduced in higher-order Markov chain usage models. Higher, or *kth-order* chains tend to "group" particular states together, while 'breaking off' into other patterns and sequences occasionally. These higher-order chains tend to generate results with a sense of pattern structure, rather than the 'aimless wandering' produced by a first-order system [15].

2.2.2 Higher-order Markov Chain Usage Model

Usage models are widely used in other research domains, such as pattern recognition, data mining. Although the original goals of these usage models are different, it is very important to introduce them into statistical usage testing and software reliability testing. Here, we would like to introduce two higher-order Markov chain usage models which also can be used for statistical usage testing.

2.2.2.1 N-gram Model

In 2000, Borges and Levene developed a higher-order Markov model to extract user navigation patterns by using a Hypertext Probabilistic Grammar (HPG) [16] model and N-grams [17]. HPG is a probabilistic regular grammar which has a one-to-one mapping between the set of non-terminal symbols and the set of terminal symbols [18]. 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 [19]. 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 2.

Table 2 A collection of application state sequences for 3-gram model example

Application State Sequences	Number of Sessions
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 is given by the frequency of the bigram (sequences including two application states) divided by the overall frequency of all bigrams with the same current position. In the example of Table 2, if we are interested in the probabilities of choices from application state *A2*, we have to consider bigrams 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(A2-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 usage model. The second-order model is obtained by computing the relative frequencies of all trigrams. The HPG of trigram is represented as $p_{ijk} = \frac{w_{ijk}}{w_{ij}}$. And higher orders can be computed in a similar way. Figure 2 shows the 3-gram model corresponding the sessions in Table 2.

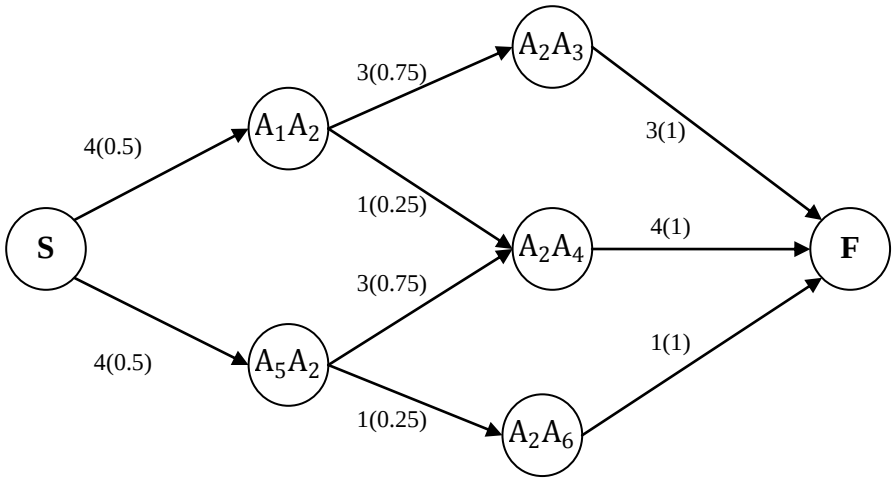

Figure 2 3-gram model corresponding to the sessions given in Table 2

2.2.2.2 Dynamic Clustering Model

Later on, the same authors presented how to use higher-order Markov models in order to infer web usage from log files [45]. In this work, 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 2 again as example. Consider state *A2* and its one-order probability *p(A3|A2)=3/8*, and its second-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.

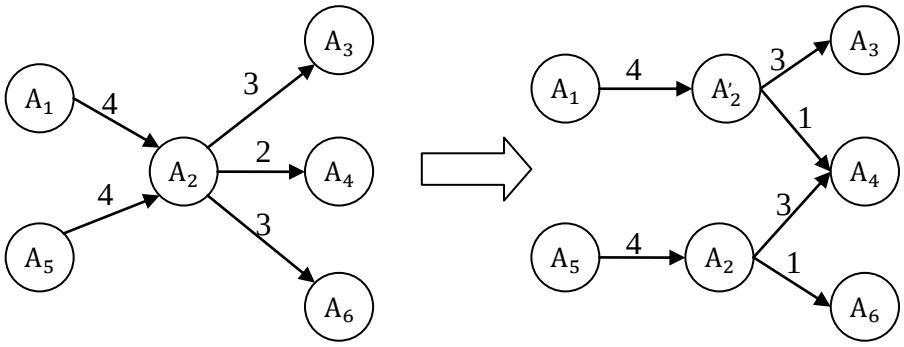

Figure 3 An example of the cloning operation in dynamic clustering modeling

2.3 Reliability Testing based on MCUM

2.3.1 MCUM for Reliability testing

Since a Markov chain model has the ability to simulate a stochastic process, it is widely used to evaluate reliability, be it hardware [20], [21] or software [9], [11], [23]. Goseva-Popstojanova and Trivedi formulated a Markov model for failure correlation, and studied its effects on software reliability measures. They assumed a two-state discrete time Markov chain model to simulate software operating states; one is an operating state, the other is a failure state. The system reliability/availability is computed by transforming discrete-time Markov chain model into a continuous-time Markov chain model [22]. Later on, they extended their work to higher order Markov chain to model a sequence of dependent software runs [23].

To our knowledge, Whittaker was the first to use a Markov chain model as a us-

age model of a system under test for statistical testing and reliability estimation [11]. He added two artificial states into the Markov chain model "Invoke" and "Terminate" that denote the start and the end of a usage scenario. The other states depend on the structure of the application. The finite state machine is constructed following the software specification. And the transition probability of each edge in the finite state machine is assigned by investigation, such as inspect the log files of previous version software or analyze software requirement. Independently of how the Markov chain usage model is obtained, the model can be used to generate "use cases" for statistical testing. A use case here is a sequence of states starting with "Invoke" and ending with "Terminate". For example, a use of word processing software could be to invoke the software, load a document, and print the document and exit. He defined the software reliability as the probability of the software executing a randomly selected usage without a failure. In his model, the feedback of testing randomly selected usages is utilized to build another Markov chain model called "testing Markov chain". Every time a unique error is encountered while testing, a failure state and arcs to the failure state are added to the testing Markov chain. The reliability is computed as the probability of going from "Invoke" to "Terminate" in the testing Markov chain without entering the failure state.

In Sayre's dissertation, a reliability estimator is introduced. It is a Markov chain usage model testing methodology which has been discussed above. A Markov chain usage model is used to generate "usages" for the software under test. However, the reliability is computed by an improved reliability estimator based on the rule of succession theory [9], [23].

In industry, the European IST project MaTeLo [3] (Markov Test Logic) uses MCUM to do statistical usage test and estimate software reliability [24], [25]. Test cases randomly generated from MCUM test against software in order to uncover failures and evaluate reliability. Sequence diagrams are mainly used to describe interactions among system components, thus MaTeLo makes use of sequence diagrams to construct a MCUM. The objects in sequence diagram are mapped to states; and events within sequence diagram are mapped to transitions of the MCUM. The transition probabilities of MCUM are assigned by requirement definitions, monitored field data of predecessor system or simulation data. Figure 4 from [24] shows the framework of MaTeLo.

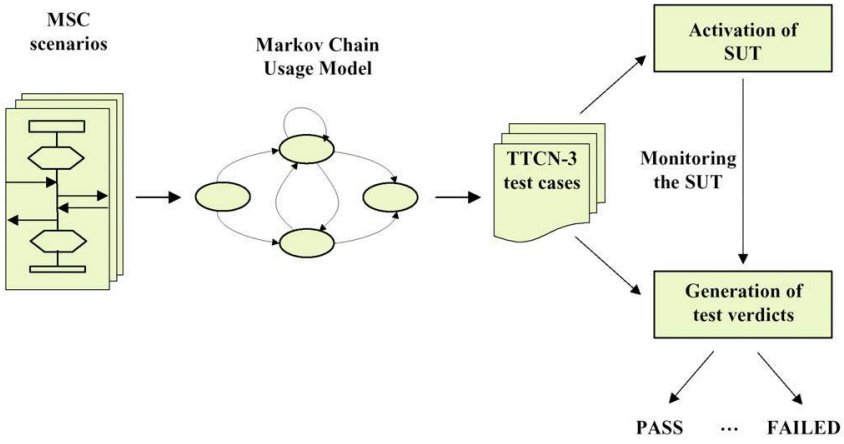

Figure 4 MaTeLo Framework[4]

[3] http://www.all4tec.net/index.php/All4tec/matelo-product.html

[4] The figure is from paper "*MaTeLo—statistical usage testing by annotated sequence diagrams, Markov chains, and TTCN-3"* [24]

2.4 Usage Model Evaluation

Although several Markov chain usage models have been proposed, the accuracy of these usage models is rarely discussed. To our knowledge, Deshpande and Karypis [26] were the first to have reported the accuracies of different Markov models by the use of statistical tests. In their work, they defined accuracy of a model as the ratio of the number of Web sessions for which the model is able to correctly predict the hidden page to the total number of Web sessions in the test set. The test set is a set of web sessions that are not used to build the model. Then the last page of each web session in the set is pruned. The model is given a trimmed session from a test set for prediction. The prediction made by the model is then compared with the trimmed page of the session to compute the accuracy of the model. For example, there is the Web session *A-B-C-D-E*. The last page is *E*. They cut the last page and use a usage model to predict it following the trail *A-B-C-D*. If the model successfully predicts the last page E, then we add 1 score, otherwise add 0. After testing all trimmed visiting sessions in test set, the accuracy of the model is computed by the amount of scores divided by the amount of predictions. Some web sessions in the test set are ignored because either the length of the test session is less than the order of the model (such as N-gram model), or the target page for prediction is unreachable.

In 2007, Borges and Leven proposed other methods for testing Markov model's predictive power [27]. The first one uses a χ^2 test to measure the distance between the probability distribution of a model built from the full collection of web sessions and a model induced from a subset of such a collection. If the χ^2 value is above the signifi-

cance level of the test (usually 5%), then the null hypothesis is accepted. It means that there is no difference between the two models. That is, assume there is a data set *S*. They split S into k subsets of roughly equal size. For instance for $k = 5$, then the subsets are $\{S_1, S_2, S_3, S_4, S_5\}$. They use the whole data set S to build a Markov usage model, ed MC_0. Subsets $\{S_2, S_3, S_4, S_5\}$ are used to build another Markov usage model, ed MC_1. The subset S_1 is used as test set. Similar to Deshpande's approach, the last pages of sessions in S_1 are trimmed. Follow the trimmed visiting session (for example *A-B-C-D*) in S_1 on MC_0 and MC_1 to reach the last page *D*. After that, they compute the χ^2 value of transition distributions at page D in MC_0 and MC_1. Do the process again until all visiting session in subset S_1 are tested. Finally, they sum all χ^2 values of each prediction and assert that the degree of freedom is the size of $S_1 - 1$. Since MC_1 usage model is built from partial information while $MC_0$is constructed from the full information, the author claimed that the acceptance of statistical hypothesis implies that the usage model has good predictive power. Although we used some of the ideas in our own work, we should point out that because MC_0 and MC_1 is induced by the same inducement algorithm, if the inducement algorithm is wrong, the χ^2 test may also accept the hypothesis. In other words, we end up using the wrong model to test the wrong model.

The second method evaluates the model's ability to predict the last page viewed of a web session based on the preceding views. This method is also similar to Deshpande and Karypis's approach. The main difference is that in Deshpande's approach if the model successfully predicts the last page, then count 1 score, otherwise count 0. In their method, they set the score by the frequency of the predicted page. If the last page is fre-

quent, they give the prediction a high score; otherwise they set a low score. After testing all visiting sessions in the test set, the higher the score is the better the prediction is.

In 2011, Sprenkle also presented an empirical study supporting specific design decisions in constructing N-gram usage models for automatic generation of test sequences [28]. They proposed a group of empirical conclusions for automatic testing based to the evidence from their experiment results. Although this paper did not discuss the accuracy of Markov models directly, in our view, some empirical implications given in the paper are highly related with model accuracy, such as the suggestion about the accuracy of transition probabilities in the model and the likelihood of generated test sequences. However, all these suggestions are based on observation of experimental results. For example, they suggested that "If a tester wants to generate tests that are more closely aligned in navigation to the user sessions, then he should choose a large n. If the test is interested in more new navigations being tested, then she should choose a smaller n." However, the accuracy of N-gram models increases with n. And no doubt that with a more accurate Markov model, the generated test sequences are more similar to the user's behavior. On contrary, if the Markov model is less accurate, the generated test sequences are different from the user behavior. Therefore, we believe that part of this empirical study can be seen as an instance of model accuracy.

Chapter 3 Hybrid Tree-like Markov Usage Model

3.1 What is a good usage model

In order to carry out a reliability test, two factors must be treated carefully. First, the generated test sequences must accurately reflect usage scenarios. Second, the generated test sequences set must precisely follow the users' usage probability distribution. First-order Markov Chain Usage Models are simple and compact, but they have limitations when they are used to model usage profiles. In Web applications, a first-order Markov model captures the page-to-page transition probabilities: $p(x_2|x_1)$ where x_1 denotes the current page and x_2 denotes one of the pages reachable from x_1. Such a low order Markov model cannot capture behaviors 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, "sign in" is required when a user attempts to "proceed to checkout" or "review my order". The probability of doing one or the other is certainly not identical after "sign in" compared with a state where some items have been put into cart. Figure 5 illustrates this example: In the snippet of a traditional Markov usage model, we see that there are two ways to reach "sign in" (from "add item" and from "review order"), and from "sign in", there is 30% chance to go to "order information", and 70% chances to go the "check out process". However, looking at the provided visiting sessions, we can see

that users reaching “sign in” from “add item” never go to “order information” afterwards. The first-order Markov chain is misleading, and may produce unfeasible test sequences, even sequences never made by users. Since it is reasonable to assume that most Web applications involve such history-dependent behaviors, accurate models of user behavior cannot be obtained with first order Markov chains [29]. The same problem is also discussed by Deshpande and Karypis [26]. Thus, a good usage model requires higher-order Markov chains.

Table 3 An example of first-order Markov chain usage model’s limitation

Visiting Sessions	Number of Sessions
add item – Sign in - check out process	210
review orders – Sign in – order information	90

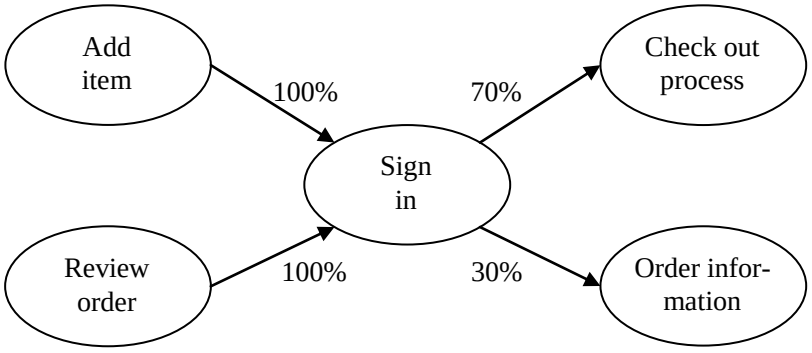

Figure 5 Example of first-order MCUM’s limitation

Some higher order Markov chain usage models, such as the N-gram model, can capture user behavior more accurately. But they do not overcome the problem illustrated above, because they focus on the last N-1 pages and will thus ignore effects involving earlier pages visited. On the other hand, it is also difficult for fixed-order Markov models

to trade off coverage and accuracy [26]. Coverage is defined as the ratio of behaviors captured by model to all behaviors from users. With N-gram, the usage patterns of lengths shorter than N cannot be captured by the model. That is, for instance, Figure 2 shows a 3-gram usage model corresponding to the visiting sessions given by Table 2. If there is a visiting session "A_1A_2" in Table 2, this visiting session cannot be presented by 3-gram usage model. Therefore, in 3-gram usage model, *the model captured behaviors* approximately equals *all usage behaviors* minus *behaviors whose lengths smaller than 2*. Hence, the *coverage* is *model captured behaviors / all user behaviors*. Thus, the set of behaviors presented by N-gram model is a subset of all user behaviors. Although the model accuracy is increased with N, the model's coverage is decreased. The same problem is also discussed by Jespersen, Pedersen and Thorhauge [30]. The results of experiments from [31] also show that as the order of N-gram model increases, the coverage of the model decreases, accompanied by an increase of accuracy.

Considering the weakness of existing usage models, we are going to create a new usage model which can address the mentioned problems. We proposed a variable order Markov usage model. The length of "memory" does not depend on the model itself, but depends on the effects of pervious behaviors towards current user behavior. In other words, if the "history" impacts the user's current behavior significantly, we should keep these "histories"; otherwise these "histories" should be compressed in order to reduce the model size. The usage model should be based on parameters. Therefore, the model users can obtain a smaller model size with user-defined model accuracy; or a more accurate usage model with fixed model size by defining appropriate parameters.

3.2 Buliding the tree of sequences

In the remaining part of this chapter, 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 that records infrequent behaviors.

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 through the same application states during their sessions. We group sessions in “application state sequences”. Table 4 shows an example, along with the number of times each state sequence occurs in the log files. The details of how to extract visiting sessions from web application access log files is discussed in Chapter 6, Section 1.

Table 4 A collections of application state sequences

Application State Sequence	Number of Sessions
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 tree of sequences (TS for short) is constructed from the given application 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 given application state corresponds in general to several 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 6 shows the TS model built from the state sequences listed in Table 4. 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, let's consider the state 4 in TS model. 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*. On the other hand, from the same state 4 following the state sequence 3-2, users choose between state 3 and state 5 with equal probability.

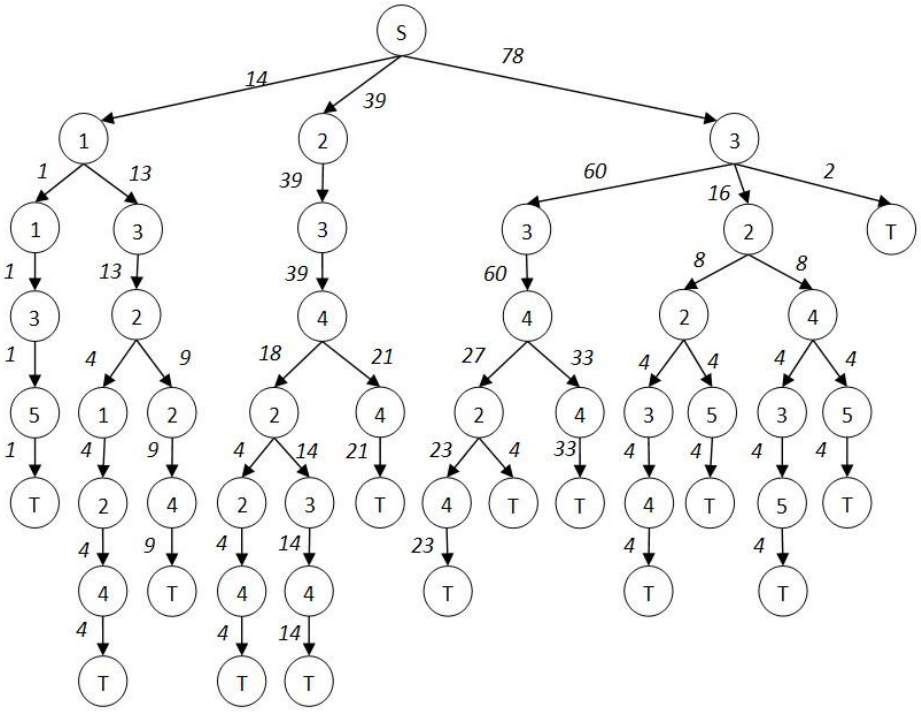

Figure 6 The tree of sequences captured from Table 6

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). And models built from very large dataset tend to be very large themselves. In addition, we mention that it is not adequate for creat-

ing new test cases; and it does not pinpoint common user behavior across different leading state sequences.

3.3 Frequency Pruning

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 very low frequency in the application state sequences do not carry reliable information from a statistical point of view. For such states, the estimation of the conditional probabilities will not be reliable. 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 first-order Markov usage model.

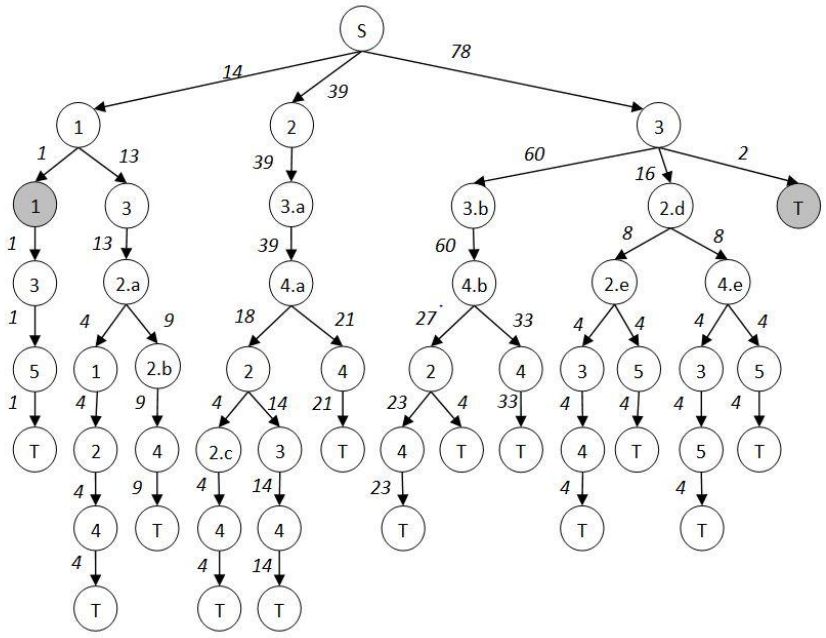

Figure 7 Frequency-pruned TS Model

The amount of pruning in the TS model is controlled by two parameters, the "percentage threshold" θ which indicates that if the probability of outgoing edge from a state is smaller than θ the edge with sub-tree should be cut, and the "frequency threshold" *c*, which is used to cut the edges whose frequency is lower than *c*. When the calculated conditional frequency of a branch is lower than frequency threshold c and its probability is lower than the percentage threshold, the branch is cut. The values of the parameters θ and c allow a trade-off between model size and model accuracy. Figure 7 shows the result of pruning of the TS model of Figure 6, with θ set to 10% and *c* set to 3. Here is an example; we consider the branches *1-1-3-5-T*. Its probability of going to state 1 (marked in grey in Figure 7) $p(1|1)=1/14 < \theta$ and the frequency $c(1|1) < c$, therefore the branch *1-1-3-5-T* is cut at the transit from first state 1 to second state 1 of the tree and the remaining part, *1-3-5-T*, is used when building the "lower Markov model". The grey nodes in Figure 7 repre-

sent access points from the TS model to the lower Markov model.

3.4 Independent Testing and State Merging

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 with different leading histories, 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 project, 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.

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 2.a, 2.b,

2.c, 2.d, 2.e, all correspond to the application state 2. 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 can be merged. An example is shown Figure 8 (a) where the statistical user behavior is nearly identical in the two states 1.a and 1.b, and these two states can therefore be merged (see Figure 8 (b)).

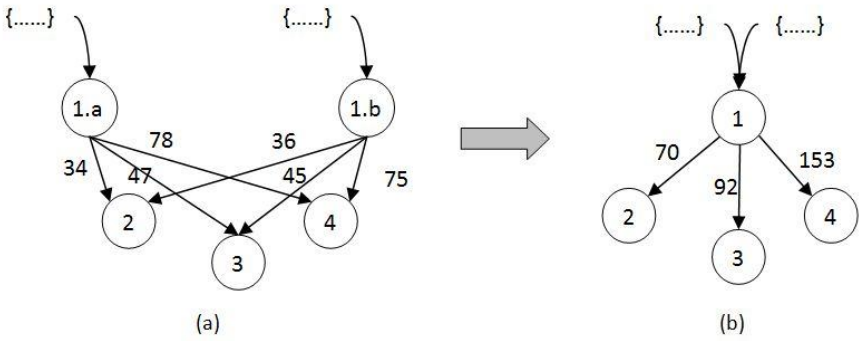

Figure 8 An example of merging two model states

We note that the model states that are the successors of the states to be merged must be identical[5] for the two states to be merged, as shown by the example of Figure 8. And merging operations must be applied from the bottom-up through the original TS

[5] 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.

model. We also 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 [32]). 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.

The purpose of Cochran criterion is to make sure the independent test has adequate data. However, due to user's infrequent behaviors, many nodes in TS model fail the Cochran criterion. Let's assume in the TS model there is a state 1 as shown in Figure 9 (a). There are two visiting sessions from state 1 to state 2 and three visiting sessions from state 1 to state 6. Comparing with other branches, these are very low frequency. According to the Cochran criterion the node 1 is not valid for independent test, since there are more than 20% of the possible alternatives that have fewer than six instances. However, it

is quite unreasonable to abandon this node, because 320 user visiting sessions have gone through this node. Some branches have limited information, while others have ample data for independence tests. To avoid that the upper tree part is strongly pruned by the Cochran criterion, we precede the application of the Cochran criterion by frequency pruning. Figure 9 (b) shows the node after frequency pruning. And then it can now be used for the independence test.

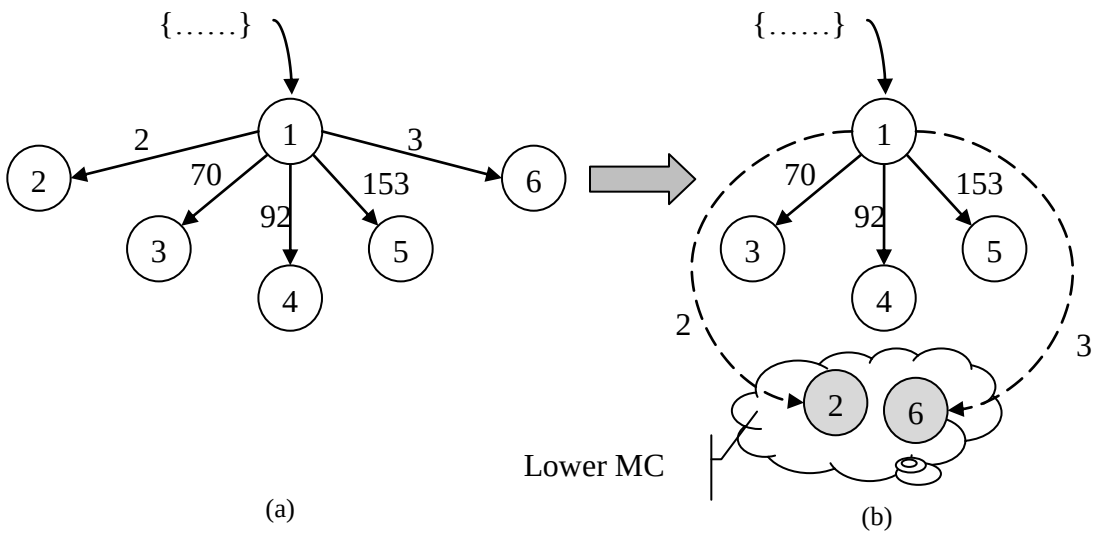

Figure 9 An example of Cochran criterion

We 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”. For our running example of Figure 7, we obtain after the application of the frequency pruning and the Cochran criterion the model of Figure 10. The grey nodes form the “lower Markov model”. For example, the model

states 2.a, 2.b, 2.c of Figure 7 were merged into the state 2 (in grey) of Figure 10. For state 2.d in Figure 7, 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 10 as state 2.d to state 2 (in grey) or state 4 (in grey) with frequencies 8 and 8, respectively.

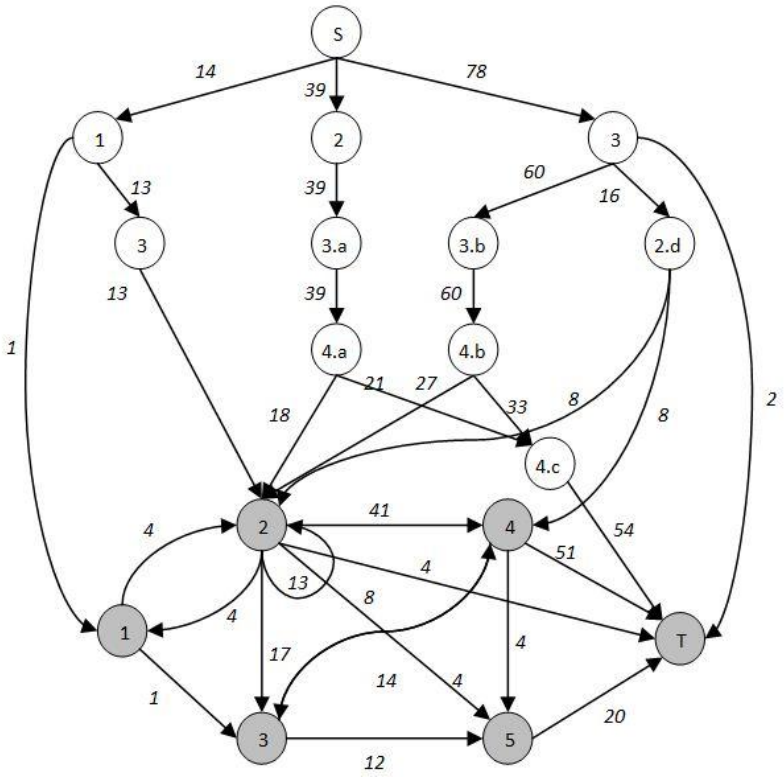

Figure 10 The model after pruning

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 4.3. 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.

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 (8)

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \qquad (8)$$

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 [32].

Let us consider the example shown in Table 5 below. It shows the observed choices to application states 2 and 4.c from the model states 4.a and 4.b (see Figure 10). If we assume that these two model states can be merged, that is, the branching probabilities to states 2 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 5 An 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.c	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 (8) 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 (in Figure 11).

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. 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 10 and Figure 11). 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, can also be merged (in this case it is merged since they have a single successor corresponding to the same application state 3). Once all candidates for merging have either been merged or are determined not to satisfy the merging condition, we obtain our final performance model, as shown for our example in Figure 11.

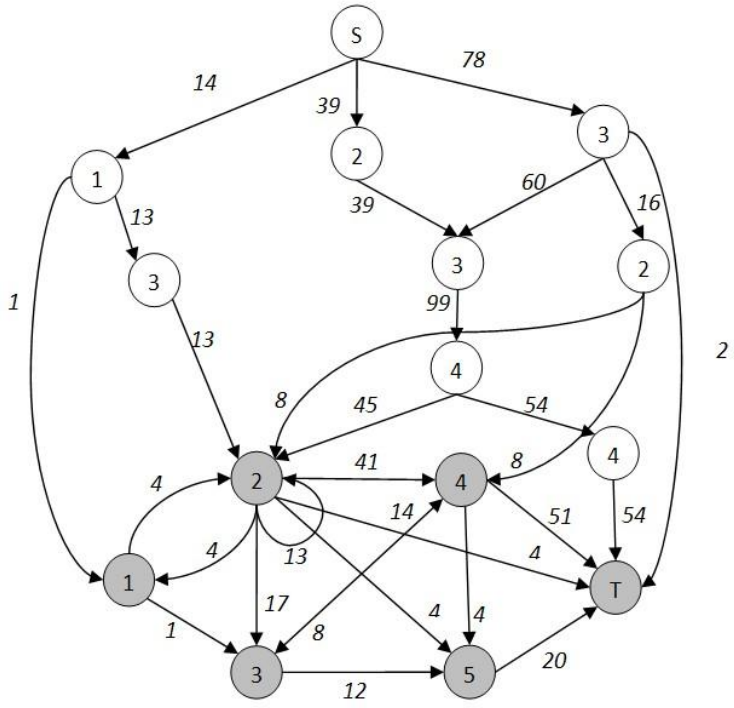

Figure 11 Hybrid Tree-like MUM constructed by sessions in table 6

The accuracy of our usage model is mainly determined by two parameters of the frequency pruning phrase, namely count threshold and probability threshold (see Section

3.3). We note that the test sequences generated by an accurate usage model are similar to the observed visiting sessions. And almost all of them are feasible. On the contrary, inaccurate usage models could generate more new test sequences, but some of them will be infeasible. We explained the reason in Section 3.1. An empirical study [28] and our experimental results also support this fact. Therefore, for software reliability estimation, the generated test sequences aim to simulate operational environment of the software. In other words, they are used to mimic the way the users use the software. An accurate model will improve the accuracy of reliability estimation. On the other hand, for software faults elimination, test sequences aim to discover more failures. the tester may be interested in a lower-accurate usage model in order to generate more new test sequences. Therefore, by choosing appropriate parameters, we allow different users to obtain different usage models according to their needs. But in this thesis, we focus on building an accurate and compressive usage model for reliability testing.

Chapter 4 Model Evaluation

As reviewed in Chapter 2, lower-order or higher-order Markov models can be used to capture software usage patterns. In other words, they all can be used to simulate the operational environment of a system for reliability testing. However, the following question arises: Which model can best represent the real operational behavior of the users? Or in other words: To what extent are the test cases generated by a usage model similar to the observed user behavior as documented in the execution sequences recorded in the application log files? These questions will have to be answered by the software engineer who wants to select a usage model that is to be used for estimating the reliability of the system through reliability testing. Therefore, determining a statistically significant usage model has a practical and profound meaning.

4.1 What is a good model evaluation method

In chapter 2, we reviewed several methods for measuring the accuracy of Markov models. Ranking or scoring is one way to assess the accuracy. Borges measures accuracy by ranking the probabilities of transitions from each state [27]. Zhang estimate accuracy by weighting the edit operations (insert, delete, and modify) to transform one dataset to another in order to calculate the distance of two sets [33]. However, since different peo-

ple may use different way to score, it is very difficult to draw a conclusion that one model is more accurate than the other.

The typical chi-square test is another way to assess usage accuracy. Borges made use of goodness-of-fit to measure the accuracy and predictive power of dynamic clustering usage models [27]. However, the chi-square test is very sensitive to the test sample size. When test sample is small, the chi-square test tends to accept the model even if it has some evident bias. When the test sample is large, the chi-square test tends to reject the model even if it is only minimally wrong. This phenomenon is also discussed by [34], [35] and [36]. Thus, chi-square test is too strict to estimate usage model when sample size is large, while it is too lenient when sample size is small.

A good model accuracy estimation method should be reliable. We desire an approach which can minimize the above problems associated with significance testing under varying sample size condition. For model selection, classical chi-square test answers a usage model fit the data or not. As George said "All models are wrong, but some are useful" [37]. When we have two different models, the classical chi-square test may be not very helpful for us to select the better one if both of them failed the test. We could not use it to pick a better model, much less to optimize the usage models by selecting appropriate parameters. Therefore, a good estimate method should have ability to measure all possible Markov usage models and rescale them into a continuous interval. We also are aware that “no accuracy estimation can be correct all the time” [38]. Due to the difference of testing sets, a usage model may fit one test set very well but fail for another test set.

Hence, we would like an estimation strategy with low bias and low variance.

4.2 Observed and model-implied data

A good Markov usage model contains the users' usage information and accurately captures the probabilities of these usages. In other words, if we observe the users' behaviors under the same conditions, the behavior suggested by the usage model should not be significantly different from the behavior observed with real users. For example, Table 6 is a collection of observed application state sequences from users' usage, also called testing sample, and the usage model to be evaluated is a first-order Markov usage model (Figure 12) obtained from a training sample (Table 7).

Table 6 A test sample observed from users' usage

Application State Sequence	Number of Sessions
S-1-3-5-T	8
S-3-4-T	2
S-2-3-4-T	3
S-2-3-5-T	9

Table 7 A training sample obtained from users' usage

Application State Sequence	Number of Sessions
S-1-3-5-T	4
S-3-4-T	2
S-2-3-4-T	1
S-2-3-5-T	3

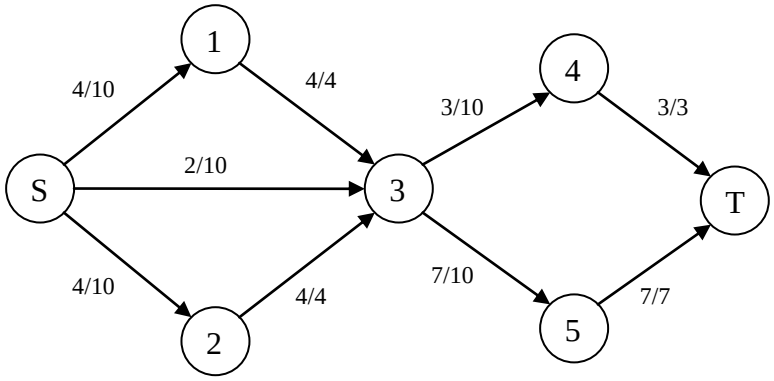

Figure 12 A first-order Markov Chain usage model obtained from table 7

Our observation sample of Table 6 suggests that users reaching state *3* after the state sequence *S-1-3* always choose state *5* afterwards, which can be expressed by the conditional probability $p(5|S\text{-}1\text{-}3) = 1$. The observed frequency of choosing *5* after visiting trail *S-1-3* is $c(5|S\text{-}1\text{-}3) = 8$. In the following, we call this the "observed values". On the other hand, the usage model implies that the probability of choosing state *5* with the previous visiting trail *S-1-3* is $p(5|S\text{-}1\text{-}3)' = 70\%$. Counting from the observation sample (Table 6), the $c(S\text{-}1\text{-}3) = 8$. Thus, the model indicates that the frequency of choosing state *5* after the visiting trail *S-1-3* is $c(5|S\text{-}1\text{-}3)' = p(5|S\text{-}1\text{-}3)' \times c(S\text{-}1\text{-}3) = 0.7 \times 8 = 5.6$. In the following, we call this the "model-implied values". Due to the significant difference between the observed and model-implied values, we can draw a conclusion that the usage model does not capture the user's behavior after visiting *S-1-3* according to the test sample (Table 6).

This simple example shows that if we want to evaluate the whole model's fitness, we need to analyze the differences between observed and model-implied values under all conditions. Assume $T = s_1, s_{2,} \dots, s_{i-1}, s_i$ is a trail observed in the test sample, the model implied frequency value $c(s_i|s_1, s_{2,} \dots, s_{i-1})$ is calculated by formula 9, where $p_m(s_i|s_1, \dots, s_{i-1})$ is the conditional probability from the model and $c(s_1, \dots, s_{i-1})$ is the frequency of prefix subsequence $s_1, \dots, s_{i-1}$ in the test sample.

$$c(s_i|s_1, \dots, s_{i-1}) = p_m(s_i|s_1, \dots, s_{i-1}) * c(s_1, \dots, s_{i-1}) \qquad (9)$$

Applying formula 9, all observed values and model-implied values are calculated and stored in two matrixes: the sample matrix (*S*) and the expected matrix (*E*). For the example of Table 6 and Table 7, the matrices *S* and *E* are shown in the Table 8. In matrix *S, S[i][j]* represents the observed frequency of going to state *i* when the prefix *j* was followed. Similarly, in matrix *E*, *E[i][j]* represents the model-implied frequency of going to state *i* when the prefix *j* is followed. Note that *S* contains all the information we can collect from the observation sample.

Table 8 An example of observed values and model implied values

	s	s-1	s-2	…	s-2-3	…
1	8	0	0	…	0	…
2	12	0	0	…	0	…
3	2	8	12	…	0	…
4	0	0	0	…	3	…
5	0	0	0	…	9	…

(a) S - Observed Value

	s	s-1	s-2	…	s-2-3	…
1	8.8	0	0	…	0	…
2	8.8	0	0	…	0	…
3	4.4	8	12	…	0	…
4	0	0	0	…	3.6	…
5	0	0	0	…	8.4	

(b) E – Model-implied Values

4.3 Logic of Chi-square Test

The classic statistical method to evaluate whether a model fits the observation is through covariance analysis and the chi-square test. Borges and Levene also used the chi-square test to estimate the predictive power of higher-order Markov models [27]. The χ^2 value is given by

$$\chi^2 = \sum\sum \frac{(S_{ij} - E_{ij})^2}{E_{ij}} \qquad (10)$$

A significant χ^2 value relative to the degrees of freedom indicates that the ob-

served and model-implied matrices differ. A non-significant χ^2 value indicates that the two matrices are similar; indicating that the usage model accurately represents the usage pattern. Whether a χ^2 value is significant or non-significant is determined by the χ^2 distribution and its null hypothesis. If the χ^2 value is larger than χ^2_α given by the corresponding χ^2distribution and a given significant level α, we reject the hypothesis, that is, the model under test does not represent the user's behavior within the confidence interval α. Otherwise, we say the usage model fits the user behavior represented by the observation sample.

However, in practice, the chi-square test has limitations:

1. "Some basic assumptions underlying the chi-square test may be false and the distribution of the statistics may not hold when these assumptions are violated." [39]

2. "A chi-square test offers only a dichotomous decision strategy implied by a statistical decision rule and cannot be used to quantify the degree of model fitness along a continuum." [35]

3. "The chi-square test of model fitness can lead to erroneous conclusions. Since the chi-square test is a direct function of the sample size, the probability of rejecting the model increases as the sample size increases, even when the model is minimally false." [36]

4.4 Logic of the Goodness-of-Fit Index Test

As we discussed above, the chi-square test may reject any model when the sample size is large enough. But some of the rejected models are still quite useful. Our goal should not be to decide whether the model is "correct" or "wrong", but to describe the extent to which the model captures the data. In particular, we want to be able to know whether one model fits the data better than another one.

Markov usage models can be used to create a series of nested models capturing more and more “history”. It starts with an independent usage model, which does not capture any correlations between states at all. In this case, the user behavior as described by a model does not depend on any factors, not even the current state. Then, the first-order Markov usage model introduces the first-order conditional probabilities: it captures a behavior where the choice of the next operation depends on the current state Then, higher-order Markov usage models can capture correlations between states that are further and further apart in the “history” leading to a more and more detailed usage pattern. Finally, the series of usage models ends with an ideal Markov usage model which is able to describe exactly the user behavior and all correlations in the observed usage behavior. In other words, the ideal Markov usage model fits all observations. Thus, the difference between the values of the observation matrix S and the matrix E of the ideal Markov usage model is 0 in all situations. Consequently, we can position any possible Markov usage model on a scale ranging from 0 to 1 representing the so-called Goodness-of-Fit Index (GFI), where the independent model is the reference of the worst usage model (GFI = 0)

and the ideal Markov usage model is the best one (GFI = 1). The equation to find the position of any "proposed" usage model on this scale is given by (11) [36]

$$GFI = \frac{\chi^2_{independent} - \chi^2_{proposed}}{\chi^2_{independent} - \chi^2_{ideal}} \qquad (11)$$

where $\chi^2_{\text{independent}}$ is the chi-square value of the independent model, χ^2_{proposed} denotes the chi-square value of the proposed model and χ^2_{ideal} expresses the chi-square value of ideal model (thus χ^2_{ideal} is always 0 by definition).

We note that the GFI value has no statistical meaning; therefore it is not easy to provide a meaningful interpretation. For instance, what does it mean if the GFI value of a model is 80%? – Experience is required to associate some meaning the various possible GFI values. Bentler and Bonett claim that a GFI value of more than 90% indicates the model fits the observed data well [36]. This shortcoming of the GFI value comes with an advantage: Since there is no statistical meaning, we do not have to worry about the basic assumptions on the statistical distributions (see limitation (1) in Section 4.3).

In practice, statistical accuracy is not the only criterion that is relevant to evaluate a usage model. The model size is also of importance. In fact, provided that we have access to a training sample that is large enough, the tree model will have a GFI of 1. However, this model cannot really be used because of its very large size. Therefore, in our model evaluation, in order to choose a model for reliability testing, we would tend to select a model that has fewer states among the ones that have a good enough GFI (say above 90%).

4.5 k-fold cross validation

The above discussion considers the question, given a finite set of test samples, of how to estimate the accuracy of a Markov usage model derived from a subset of the test samples. Single accuracy estimation is usually meaningless due to differences of test samples. Estimating the accuracy many times by different test sample sets can reduce the estimation bias and variance. However, in practices, data is always expensive and limited. Thus we have to consider how to reduce the bias and variance in order to improve the confidence of the accuracy estimation with limited data.

In this thesis, we use a k-fold cross validation strategy to solve this problem. In k-fold cross-validation, the dataset S is randomly split into k subsets (the folds) $S_1, S_2, S_3, \dots S_k$ of approximately equal size. The Markov usage model is built and tested k times; each time $t \in \{1, 2, 3 \dots k\}$, the model is built by $S - S_t$ and tested by S_t. The final estimate value of accuracy is the average of the k estimates [40]. Formally, the cross-validation estimated accuracy is:

$$acc = \frac{1}{k}\sum_{1}^{k} acc_k \qquad (12)$$

K-fold cross validation can efficiently reduce the bias of estimate. Ron Kohavi in his paper [41] discussed this issue. His proposition presents that no matter whether the usage model inducement algorithm is stable[6] or not, the variance of the k-fold cross-

[6] The inducement algorithm is **stable** for a given dataset and a set of perturbations, if it induces usage model that make the same decision when it is given the perturbed datasets.

validation estimation has hardly any change when the number of folds is varied. Indeed, it is impacted by the size of data set. The general proof idea is that for each test instance from test fold S_i is a Bernoulli trail, because the test result is the model fit the data or not. All folds are used to test the model; that is if the sample *S* has n data then there are n trails. Therefore, estimated accuracy has a binomial distribution with n trails. According to the feature of binomial distribution, the variance of estimation is associated with sample size n not with the number of folds k. About bias, he presented that k-fold cross validation will be unbiased with moderate values of k. The straight idea is simple. Like testing a fair coin, the more tests we do, the less bias between estimated value and real value is. If the test set size does not change, the larger the k is, the more accurate the estimate result is. However, the size of test set decreases when the number of folds k increases. That is the information in each test set decreases which leads to the bias of estimation. Therefore, we need trade of the test set size and the number of folds. We will illustrate the way we used this strategy to reduce the estimation bias later in Section 6.2.

Chapter 5 Prototype for Modeling and Evaluation.

We have developed a prototype which (a) is capable of building a hybrid tree-like Markov usage model to capture the user's usage behaviors from web application log files, (b) has the ability to estimate the model, and (c) is able to generate abstract test cases. The abstract test cases here are the sequences of application states. They can be converted into executable test cases by adding a set of input parameters. In this chapter, we briefly present the algorithm we implemented for each module. We also discuss how we generate abstract test cases based on walks on the proposed usage model. As illustrated by Figure 13, the tool consists of several modules which are described in the following subsections.

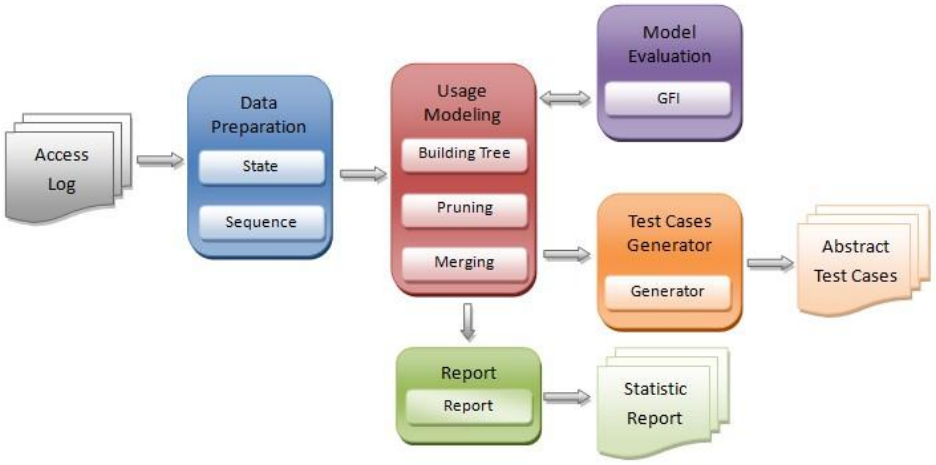

Figure 13 Modules and main functions of prototype

5.1 Data Preparation Module

The Data Preparation module was implemented by two international exchange students, Christophe Günst and Marie-Aurélie Fund [42]. It is responsible for the logic of extraction of application sequences from the web server log files. This process contains the following steps.

Noise Elimination: The web server log files contain every request to the site. These include genuine users' activities as well as crawling robots. Crawling robots traverse the web site based on some strategies to collect web site information. Therefore, it is unreasonable to include the search engine's log entries in our usage data. Figure 14 shows a fraction of a log file from the web site "Bigenet". It records the user's IP address, time, service URL and protocol, status, size and Agent. If the client is a search engine, some information will be recorded in log entry. For example, Google's search engine leaves marks as "GoogleRobot" and "crawling" etc. Eliminating the logs entries that contain such key-words can eliminate the noise in the log files.

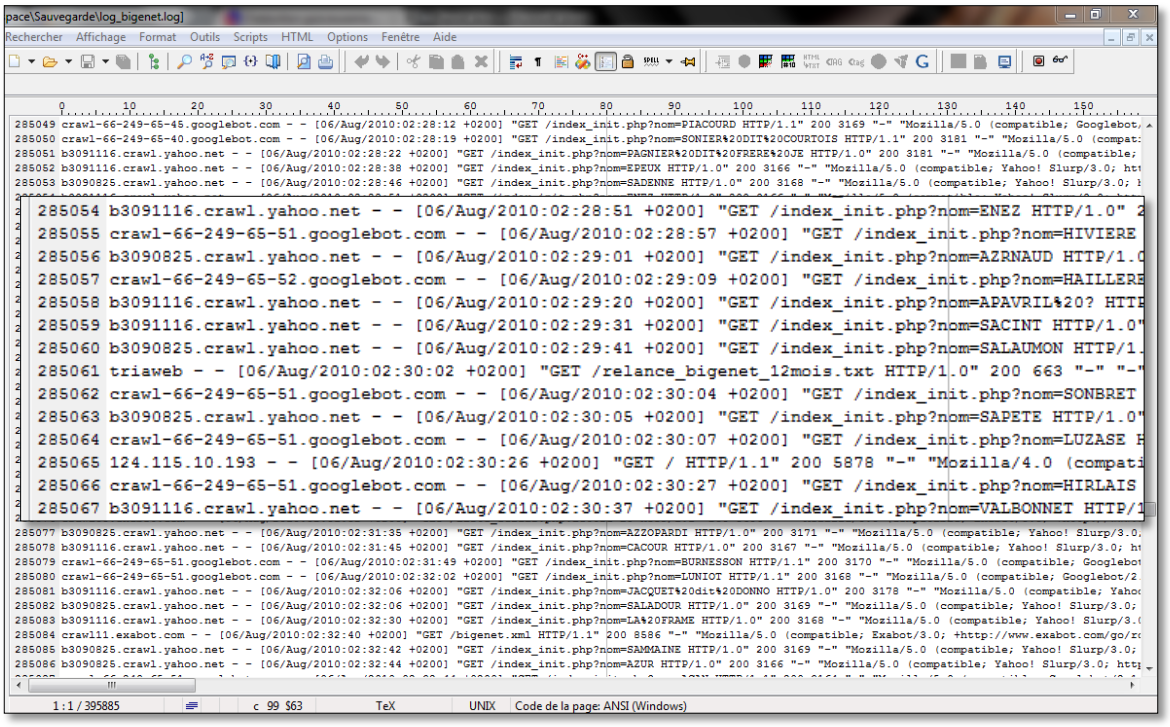

Figure 14 Log file from Bigenet

User Identification: Although the IP address identifies the sender's computer, it is not enough to be used to identify the user. As presented in [43], if the IP address is the same but the log shows a change in browser software or operating system, a reasonable assumption to make is that each different agent type for one IP address represents a different user. Therefore, we identify users not only by IP address but also by browser types and operating system.

State Identification: the state in the usage model normally represents the result of the function offered by the web application. Therefore, it can be defined by the design documents of the web application. Figure 15 shows the web site map of Bigenet. Each php page is a service offered by the web application. Table 9 shows the application states

corresponding to the web pages.

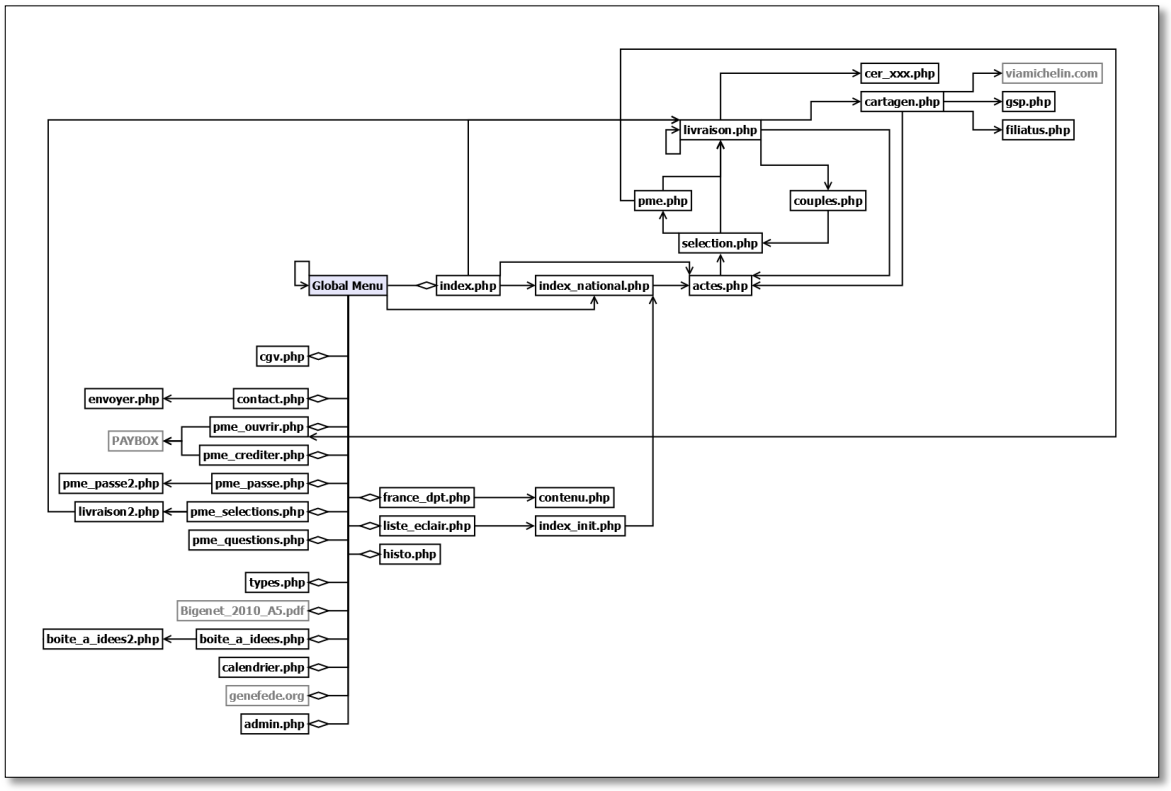

Figure 15 Bigenet web site map[7]

[7] The figure is from project report "*ELABORATION D'UN MODELE STOCHASTIQUE D'USAGE POUR UNE APPLICATION WEB A PARTIR DE L'ACTIVITE UTILISATEUR*" written by Marie-Aurélie Fund and Christophe Günst [Günst2010]

Table 9 Application States Table

Application State	URL
1	http://www.bigenet.org/index.php
2	http://www.bigenet.org/index_national.php
3	http://www.bigenet.org/actes.php
4	http://www.bigenet.org/selection.php
5	http://www.bigenet.org/livraison.php
6	http://www.bigenet.org/couples.php
7	http://www.bigenet.org/wiki.php
8	http://www.bigenet.org/contact.php
9	http://www.bigenet.org/cer_*.php
10	http://www.bigenet.org/cgv.php
11	http://www.bigenet.org/filiatus.php
12	http://www.bigenet.org/pme.php
13	http://www.bigenet.org/envoyer.php
etc…	etc…

Session Identification: If log entries associated to a given user span a very long periods of time, it is very likely that the user visited the web site more than once. The goal of sequence extraction is to divide the page accesses of each user into individual state sequences and each state sequence represents a use of the web application. Some empirical studies show that the time period that people browse a web site follows a Poisson distribution. And 95% of the visiting sessions are less than 25.5 minutes long [44]. Thus we applied 25.5 minutes as session's timeout to identify unique user's visiting sessions.

Finally, this module exports all extracted visiting sessions into an XML file. Figure 16 shows a segment of application sequences extracted from a log file where each item presents a visiting session.

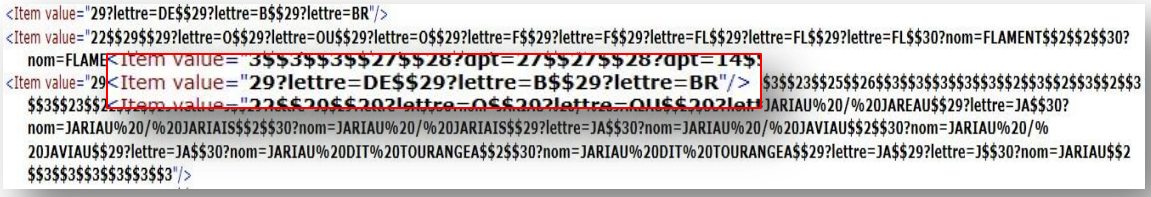

Figure 16 A segment of application sequences extracted from log file

5.2 Usage Modeling Module

The usage modeling module is responsible for building and training the usage model following modeling algorithm and application sequences generated from the data preparation module. In the following, we introduce the main steps for creating the hybrid tree-like Markov chain usage model. The logical process of deriving the usage model has been discussed in Chapter 3. Here we mainly discuss the implementation of bottom-up state merging.

After pruning, the leaves on the upper tree connect to the nodes in the lower first-order Markov chain (see Figure 17). Each node in the lower Markov chain part records the leaves which connect to it in upper tree part. As we discussed in Chapter 3, here 1.a, 1.b, 1.c etc. represent model states. They are defined by different visiting histories. For instance, 1.a, 1.b, 1.c are three leaves in the upper tree part. They all correspond to application state 1 which represents a web page. Due to different visiting histories, they are three different model states in the model.

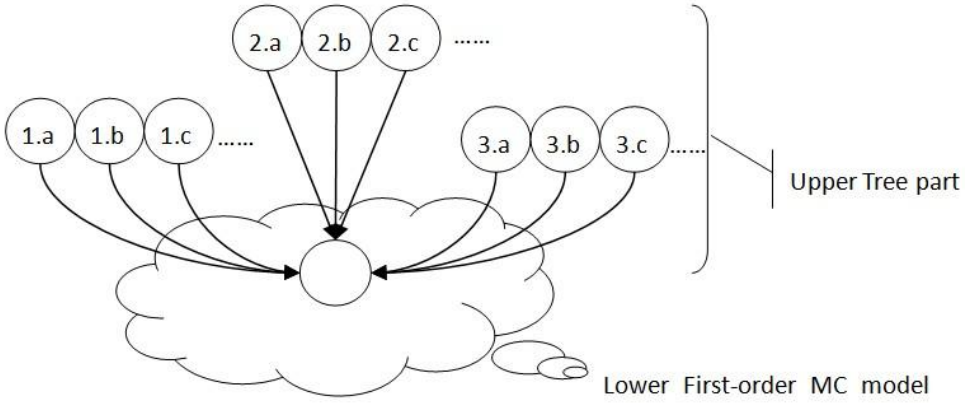

Figure 17 A node in the lower first-order Markov chain model

Merging process starts from the leaves of the tree and goes to the top. Since nodes in the lower Markov chain part stores the references (pointers) of incoming leaves in the tree, it is very easy to reach them from the lower Markov model. In order to reduce the deviation produced by merging, we compute chi-square values of all possible pairs of leaves and always merge the most similar node first (the pair with the smallest Chi-square value). When two nodes merge into one, for instance node 1.b merges into node 1.a, we mark 1.b as "merged" and delete all references (or pointers) in 1.b. However, in the model some other nodes may be connected to 1.b. So after merging process, we traverse the model and clean up all "merged" nodes. The pseudocode of the merging algorithm is shown in Figure 18.

```
Merge(){
    For each node in the Lower MC part{
        For each application state {
            // assume there are N model states corresponding to current application state.
            Create an N × N matrix
            Compute each pair's chi-square value, fill the value into corresponding cell in
the matrix
            λ = the smallest chi-square value in matrix
            While (λ<significant value){
                Merge the pair of nodes whose chi-square value correspond to λ;
                Recursively merge their parents while it can be done.
                Update matrix and re-calculate chi-square values;
                λ = the smallest chi-square value in matrix.
            }
        }
}
```

Figure 18 Pseudocode of merging algorithm

The merge operation is costly. The most expensive part is calculating the Chi-square values for all possible pairs. Take Figure 19 as an example. Assume there are 5 candidate leaves in the tree part denoted 1.a, 1.b, 1.c, 1.d, 1.e corresponding to application state 1. Then we calculate the chi-square values of all possible pairs involving them. Due to the commutative law, only cells in a triangular matrix are necessary to be computed. Assume that the chi-square value of node 1.a and node 1.c ($\lambda_{a,c}$) is the smallest in the matrix. Then we merge node 1.a and node 1.c which forms a new node 1.a'. After merging operation, we delete the rows and the columns corresponding to 1.a and 1.c; add a new row and columns 1.a'. Then we update the matrix by computing all cells in row 1.a' and column 1.a'. The complexity of merging process is $O(mn^2)$, where m denotes the number of application states and n denotes the number of nodes in the upper tree.

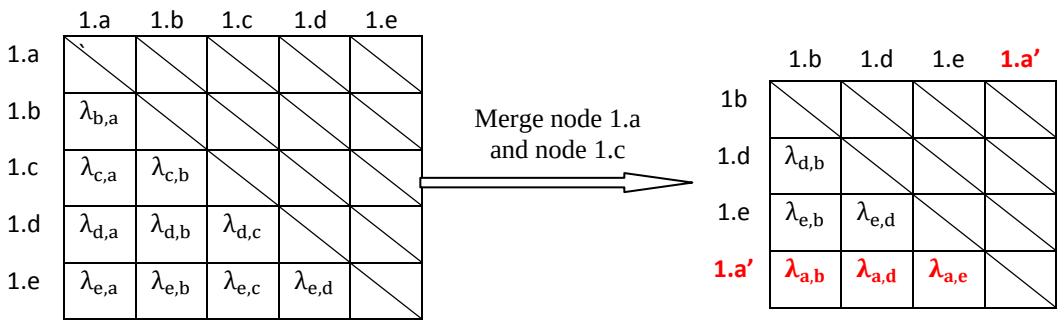

Figure 19 An example of chi-square value matrix updating

5.3 Model Evaluation Module

We implemented the usage accuracy estimation algorithm which we discussed in Chapter 4 in this module. After the hybrid tree-like Markov chain usage model is built, it is passed to this module to be tested. If the module accuracy does not meet the tester's requirement, the tester can adjust the parameters to rebuild the usage model based on the feedback from the evaluation module. Our estimation assumption is that all Markov chain (MC for short) usage models lie between the worst MC usage model and the best MC usage model. Here, the worst MC usage model is the independent model (discussed in Section 4.3), because it does not present any correlation between states in the model. And best model is an imaginary MC usage model which can precisely capture all users' behaviors and fits all test samples. Then, taking these two models as references (100% as best model and 0% as worst model), the accuracy of any MC usage model can be rescaled by percentage in the range of 0% to 100%. Therefore, in this module, we implemented

these two reference models. We note that the perfect model does not exist. The TS model (see Section 3.2) is the most accurate model in practice, because it contains all the information from this sample. Therefore, in this module, we also implemented a TS model. In accuracy testing, the TS model can be considered as a very important reference. Hence, the module provides not only the proposed usage model's Goodness-of-Fit Index (GFI for short), but also the GFI of the TS model as accuracy reference.

5.4 Test Case Generation Module

The abstract test cases are generated in this module. The generation algorithm is described by the pseudocode of Figure 20. In the usage model, two nodes are connected by an edge. The edge contains the transition probability and the probability of the generated sequences passing through this edge during test cases generation (here we called "passage probability"). The passage probability is given by (the visiting times of the edge / the visiting times of the node). When the pointer reaches a node, it always chooses the edge which leads to the minimum deviation between the transition probabilities and passage probabilities of the node.

```
Test Cases Generation (node S, int Num){
        Input S: start node S of usage model;
        Input Num: the number of test sequences to be generated.
        For(int i = 0; i<Num; i++){
                Create an empty test sequence.
                Set the pointer points to state node S
                //here the pointer walks on the usage model until it reach terminal state.
                while(pointer != "Terminal"){
                        add node into test sequence.
                        For each outgoing edges{
                                Calculate deviation between transition probability and pas-
                                sage probability
                         }
                        Choose the edge with biggest deviation
                        Pointer goes to next node through this edge.
                }
                Add generated test sequence into test sequence set
        }
        Return test sequences set
}
```

Figure 20 Pseudocode of Test Cases Generation

5.5 Report Module

The Report module is a flexible utility module. It computes records and reports statistics during model construction, model evaluation and test cases generation. The information is finally exported to an Excel file. When the program runs, all interesting information is recorded in a dictionary. After the execution, the dictionary is passed to the report module. The report module then parses the dictionary, extracts data and generates a report. During modeling, this module keeps track of data such as the training sample size, average length, the induced model size, executing time etc. During the estimation proce-

dure, it records running time and the GFI etc. And during test case generation, it calculates the basic statistical features of generated test cases, such as test cases size, mean and variance of test sequence length etc.

Chapter 6 Experiments and Evaluation of Results

This chapter is divided into four parts. In Section 6.1, experiments are conducted to evaluate the results and performance of the usage model derivation algorithm. We are interested in the size and the structure of the proposed usage model after pruning and merging. It is also important to see how quickly the usage model could be built from the visiting sessions. In Section 6.2, we evaluate the accuracy of usage models based on the Goodness-of-Fit Index evaluation method. Here we compare our model with the TS usage model and the first-order Markov Chain usage models. In Section 6.3, we present and analyze the generated test sequences from different Markov chain usage models. Through comparison, the result also supports the correctness of the proposed accuracy evaluation method. Finally, in Section 6.4, through an empirical study of models built by different parameters, we discuss some implications of selecting the best parameters for modeling.

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. We had at our disposal the access log files for the period from September 2009 to September 2010 which contains more than ten thousands visiting sessions. Table 10 presents a summary of the characteristics of the visiting sessions from a raw log file.

Table 10 Summary statistics for the data set from the raw web log

Characteristics	Bigenet
Num. of Application States	30
Num. of Request	900,689
Num. of Sessions	108,346
Num. of Application State Sequences	27,778
Ave. Session length	8.3132
Max. Session length	301
Standard Deviation of Session length	18.4123

6.1 Model Building

In the experiment, we eliminated search engines' traces from the raw log files. Table 11 shows a summary of the characteristics of the visiting sessions after the elimination.

Table 11 Summary statistics for the data set after data clearing

Characteristics	Bigenet
Num. of Application States	30
Num. of Request	794,958
Num. of Sessions	88,666
Num. of Application State Sequences[8]	27,017
Ave. Session length	8.9657
Max. Session length	100
Standard Deviation of Session length	16.2741

We have developed a prototype that implements the model construction approach

[8] the sessions going through the same sequence of application states, such sequence we call "application state sequences"

described in Chapter 3. It creates the TS model from the list of application state sequences inferred from the reconstructed sessions, and then performs the frequency 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 327,450 nodes in the tree. Figure 21 shows the TS model based on a subset of 1000 visiting sessions.

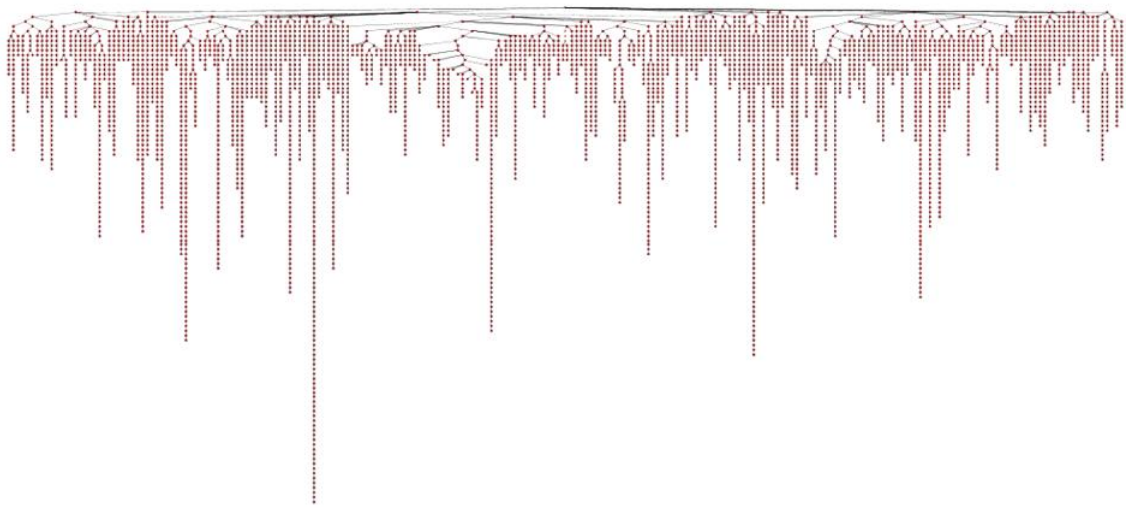

Figure 21 The TS model created from 1000 visiting sessions.

Obviously, the TS model is too large to be used. Using the proposed modeling approach, we built the usage model from these 88666 visiting sessions. Table 12 shows the results of our analysis. These results are based on executions on a laptop computer running Windows 7 with a 2.53GHz Intel Core 2 P8700 and 2.93 GB RAM. The following parameters were used during our analysis: (a) the frequency threshold was 50 and the probability threshold was 5%; (b) the confidence level for the independence test was 95%. We built three usage models: from 88666 visiting sessions (All), from 44333 visiting sessions (Half) and from 22166 visiting sessions (Quarter).

Table 12 Summary of usage model building results

Characteristics	All	Half	Quarter
Num. of states in TS	327,450	163,110	81,870
Num. of states after frequency pruning	194,829	95,423	48,290
Num. of states after Cochran Pruning	3,055	1,552	836
Num. of states merged	420	179	80
Num. of states in “lower MC model”	29	29	29
Num. of states in “upper tree”	2,606	1,344	727
Num. of states in usage model:	2,635	1,373	756
Execution time for TS modeling	1,953ms	694ms	312ms
Execution time for pruning &merging	106,628ms	14,429ms	1,784ms

We note that the frequency-pruning and the Cochran criteria leads to a usage model that has a very much reduced “upper tree” and a “lower Markov model” that corresponds to the states of the application, which contains 29 states. The merging of non-independent states in the “upper tree” leads in our case to a further reduction of 10% of the model states. Most of the applied tests for merging succeeded. The “upper tree” is shown in Figure 22

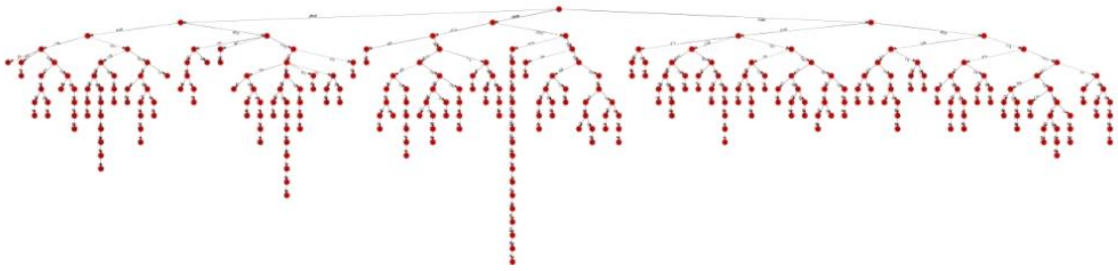

Figure 22 The upper tree part of the usage model

Table 13 shows how the execution time grows when the number of visiting sessions grows. The visiting sessions in each set are randomly selected. And Figure 23 gives the trend of this growth.

Table 13 The summary of model derivation execution times

Characteristics	Set 1	Set 2	Set 3	Set 4	Set 5
Num. of visiting sessions	10,000	20,000	30,000	40,000	50,000
Num. of states in TS model:	38,683	74,395	111,707	146,579	181,223
Num. of states after Pruning	404	775	1,119	1,447	1,721
Num. of states merged	47	93	119	187	211
Num. of states in usage model	357	682	1,000	1,260	1,510
Execution time for TS modeling	117ms	269ms	592ms	661ms	759ms
Execution time for pruning & merging	251ms	1,637ms	4,555ms	12,889ms	22,353ms

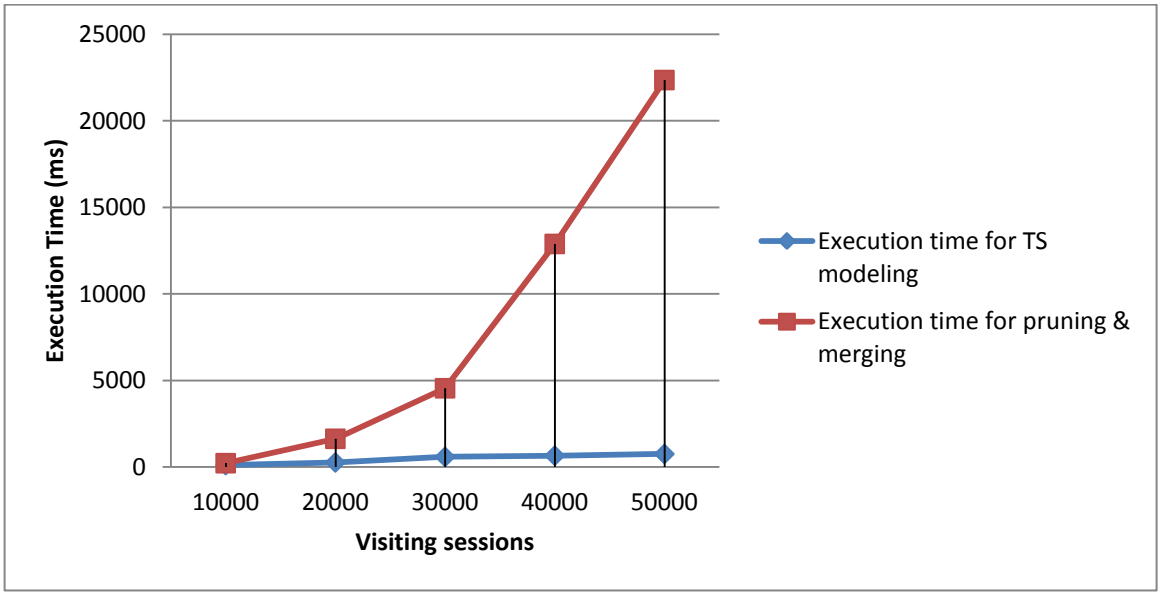

Figure 23 Time taken to generate usage model for 10000 to 50000 visiting sessions

6.2 Model Accuracy Estimation

As we discussed in Chapter 3 and Chapter 4, we believe that the TS model is the

most accurate usage model in practice, because it records all information about the user's behaviors. In Chapter 2 we also reviewed the first-order Markov chain usage model (shown as Figure 24) which is widely used in reliability testing and statistical usage testing. Therefore, in this experimental section, we estimated and analyzed these two models and two hybrid tree-like Markov usage models with different parameters.

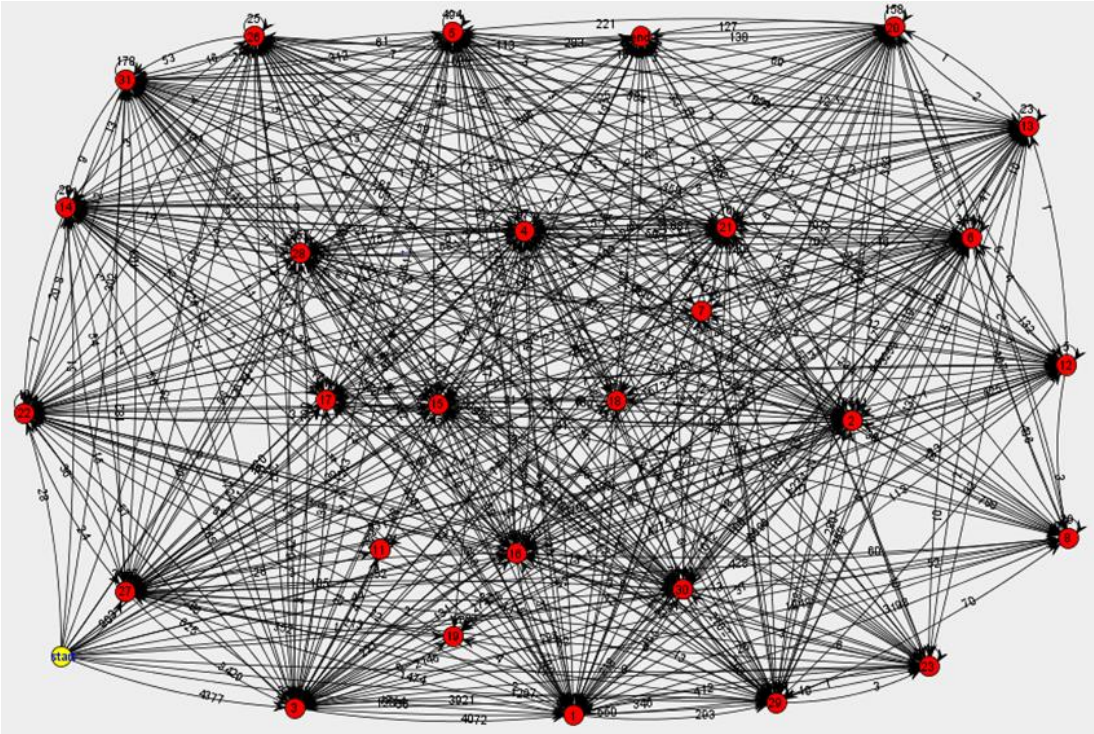

Figure 24 First-order Markov chain usage model from 88666 visiting sessions

In order to avoid the estimation bias, we used k-fold cross validation: we split the whole sample *S* into *k* sub-samples of approximately equal size. The usage model is trained and tested *k* times. Each time, we use all the *k-1* sub-samples to train model, then the remaining sub-sample is used to test the model. We ran two experiments. considering the test sample size, we split our sample set S of 88666 visiting sessions into three ran-

domly selected sub-samples (folds) for one experiment. And in the same way, we split S into 5 sub-samples for the other experiment. The adequate training sample and testing sample insure that the usage model is trained and tested with large enough data.

In the first experiment, we estimate the accuracy of four different models. The four models are (1) The first-order Markov chain usage model, (2) the simplified hybrid tree-like Markov usage model presented in [1] with a frequency pruning threshold of 5%, (3) a hybrid tree-like Markov usage model where a branch is pruned when there are less than 25 observations for it (instead of having less than 5% of the alternatives), and (4) the tree model containing all branches of the training sample. To reduce the estimation bias, we use K-fold cross validation presented in Section 4.5. Although K-fold cross validation can efficiently reduce estimation bias, we still need to trade off the number of folds, K, and the testing sample size. When K is large, the model can be estimated more times which imply that the estimated accuracy is closer to the real value. On the other hand, when the testing sample size is large, the model can be tested more. Thus in a single test, the estimated value is more accurate. However, the data size is fixed. If we pick a large K, the testing sample size in each test is small, and vice versa. To balance the number of folds and testing sample size, we selected a 3-fold cross validation for the first set of experiments. Table 14 shows the result of the first set of experiments. We also want to estimate usage model using less testing sample but more rounds of tests. So we select 5-fold cross validation for the second set of experiments. The results are shown in Table 15.We note that there is a little difference between the validation results obtained by 3-fold cross validation and 5-fold cross validation. We believe that 3-fold cross validation

provides more accurate results, because the model is trained by less visiting sessions but tested by more.

Table 14 Details of the Goodness-of-Fit Index test by 3-fold cross validation.

	1st round	2ed round	3rd round	Average
First-order MCUM	71.866%	69.519%	75.472%	**72.286%**
Simplified Hybrid tree-like MUM	87.635%	87.969%	86.761%	**87.455%**
Hybrid tree-like MUM	90.898%	92.184%	92.809%	**91.964%**
TS model	99.303%	98.685%	98.789%	**98.985%**

Table 15 Details of the Goodness-of-Fit Index test by 5-fold cross validation

	1st round	2ed round	3rd round	4th round	5th round	Average
Model 1	65.367%	63.113%	78.187%	65.464%	74.094%	**68.525%**
Model 2	85.912%	83.100%	82.8972%	83.588%	86.755%	**84.450%**
Model 3	90.405 %	89.266 %	88.895%	90.106%	91.482%	**90.031%**
Model 4	98.509%	98.288%	98.309%	98.462%	98.588%	**98.431%**

Model 1: 1st-order MCUM
Model 2: Simplified Hybrid tree-like MUM
Model 3: Hybrid tree-like MUM
Model 4: TS model.

The summary results of the GFI tests are reported in Table 16. The model size is the average of model sizes in three rounds of tests. And Figure 25 give you more direct information.

Table 16 Summary result of the Goodness-of-Fit Index tests for four different usage models

	Goodness-of-fit Index	Model Size
First-order MCUM	72.278%	32
Simplified Hybrid tree-like MUM	87.455%	950
Hybrid tree-like MUM	91.964%	898
TS model	98.985%	225,066

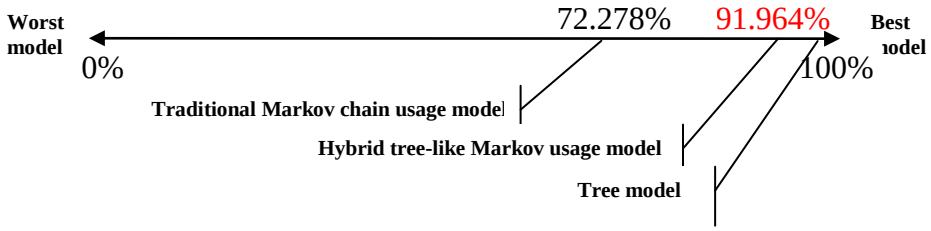

Figure 25 The accuracy of hybrid tree-like Markov usage model

We note that the first-order Markov chain usage model has a relatively low GFI. Thus, it is not very suitable for simulating the software's operational environment for reliability testing. Although the simplified hybrid tree-like Markov usage model has a much better GFI, it is still below 90%, which shows that it could be improved. Tracing the issue, we found that using a percentage to prune the low-frequency branches in the tree may lead to over-pruning. Since the hybrid tree-like Markov usage model is a parameter-based model, we were able to optimize the pruning parameter (in this case by setting it to a count of 25) to achieve high accuracy while maintaining a small model size.

As shown in Table 16, we note that the accuracy of the hybrid tree-like Markov

usage model has an improved GFI value (91.964% instead of 87.499%) while the model size is reduced by more than 50 states. The GFI value of the TS model is nearly 100%. The difference of 1.015% results from the difference between the training sample and testing samples. The difference decreases as the sizes of the training and testing samples increase. In this case, the tree model, which approaches the ideal model, has an GFI value that is 7.061% better than the optimized hybrid tree-like Markov chain usage model, however, the size of this model is much larger (by a factor 250).

6.3 Test Sequence Generation

The goal of the project is of course to be able to generate test sequences for reliability testing. In this section, we mainly analyze the result and performance of test sequence generation. We generated test sequences by walk on the hybrid tree-like Markov chain usage model which was trained by 88666 visiting session. To build the usage model, the following parameters were used: (a) the frequency threshold was 50 and probability threshold 5%; (b) the confidence level for the independence test was 95%. The main characteristics of the usage model are shown in Table 17. In order to compare the statistical features of the generated test sequences and the original visiting sessions, we applied this usage model to generate the same number of test sequences (88666). Table 18 shows the statistical features of the original visiting sessions and of the generated test sequences.

Table 17 Characteristics of usage model for test sequence generation

Characteristics	Usage Model
Num. of used visiting sessions	88,666
Num. of states in TS	327,450
Num. of states after frequency pruning	194,829
Num. of states after Cochran Pruning	3,055
Num. of states merged	420
Num. of states in “lower MC model”	29
Num. of states in “upper tree”	2,606
Num. of states in usage model:	2,635

Table 18 Summary result of the generated test sequences

Characteristics	Original	Generated
Num. of Sessions	88,666	88,666
Num. of Request	794,958	771,253
Num. of Application State Sequences	27,017	28,326
Ave. Session length	8.9657	8.6984
Max. Session length	100	83
Standard Deviation of Session length	16.2741	15.5463
Execution time:	NaN	1,738ms

Compared with each other, the generated test sequences are highly similar with the original visiting sessions. But the generated test sequences are a little bit shorter than the original ones.

The generated test sequences also give us another way to verify our proposed estimation approach. The reliability-testing usage model can generate very similar test sequences, if and only if the usage model captures the user’s behaviors accurately. In order to see visually how well the test sequence generated from the model fit, we continue the experiment which we discussed in Section 6.2. There, we analyzed and discussed the GFI

of three usage models, first-order MCUM, simplified hybrid tree-like MUM model and hybrid tree-like MUM model. Now, we would like to compare the test sequence sets that generated by these three usage models.

In this experiment, we randomly split 88,666 visiting sessions into 3 folds of approximately equal size. We selected two folds to train the usage models and used the other fold as reference set. We generated test sequences based on different usage models. The number of test sequences generated in each case is equal to the number of the visiting sessions in the reference set.We sorted the sequences using radix-sort to have a consistent and regular ordering of the sequences in all four sets. We then mapped the sorted sequence sets to the BMP image by assigning colors to different application states. Table 19 shows the statistical features of the original visiting sessions and the three generated test sequences. Figure 26 shows the difference between the four sets, where white means empty.

Table 19 Statistical features of original visiting sessions and three generated test sequences

Characteristics	Original	Model 1	Model 2	Model 3
Goodness-of-Fit Index:	NaN	72.278%	87.455%	91.964%
Model Size:	NaN	32	950	898
Num. of Sessions	29,642	29,642	29,642	29,642
Num. of Request	263,249	265,079	212,145	262,247
Num. of AS Sequences	10,039	13,668	9,519	10,536
Ave. Session length	8.8809	8.9426	7.1784	8.728
Max. Session length	100	32	85	79
StandDev of Session length	16.100	13.340	14.519	15.558
Execution time:	NaN	662ms	442ms	556ms

Model 1: 1st-order MCUM
Model 2: Simplified hybrid tree-like MUM
Model 3: Hybrid tree-like MUM

Num. of AS Sequences: Number of Application State Sequences
StandDev of Session length: Standard Deviation of Session Length.

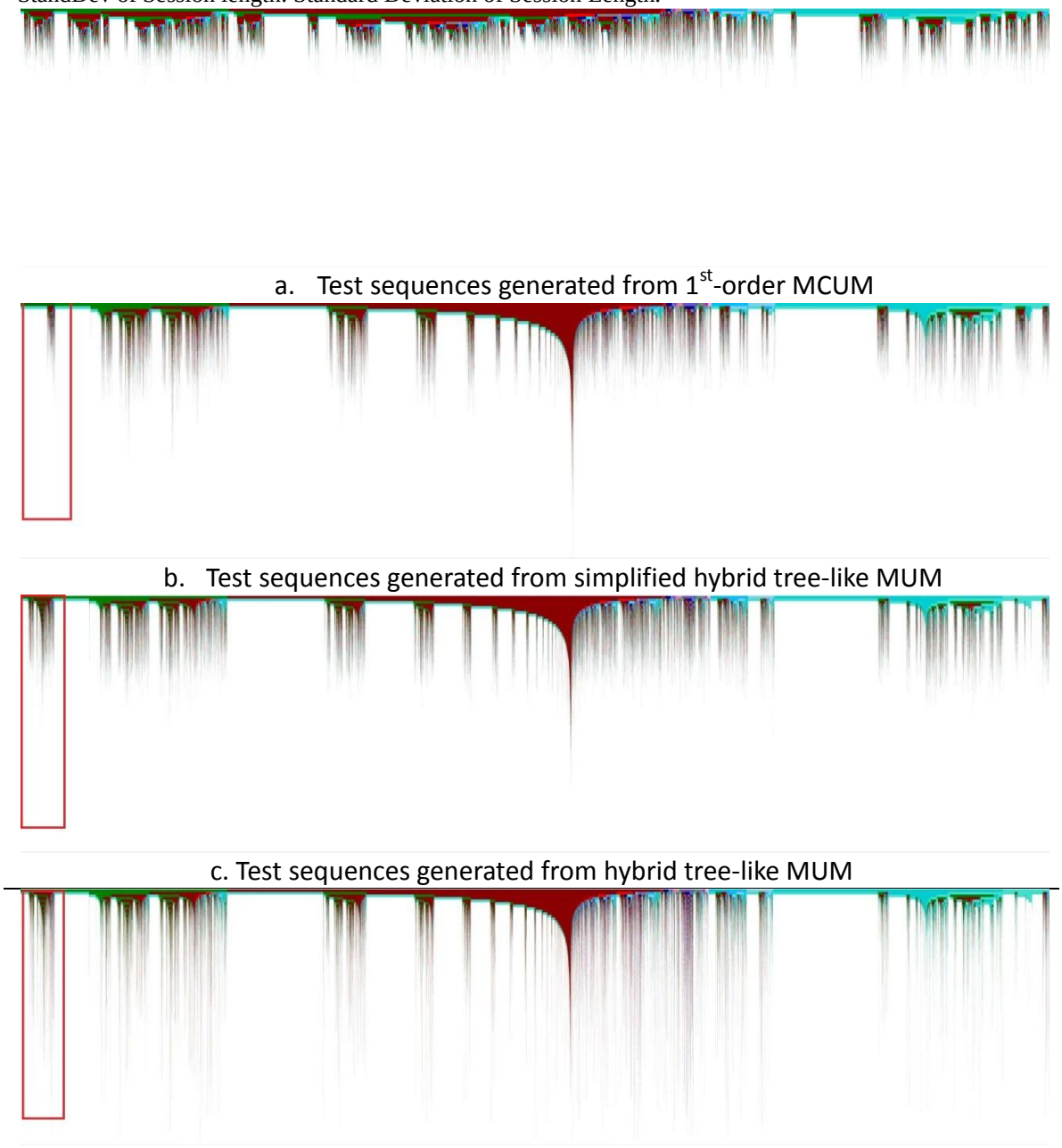

Figure 26 Generated Test Sequences Sets from different usage Models

As explained in Section 6.2, the GFI of the first-order MCUM is 72.278%, the GIF of the simplified hybrid tree-like MUM is 87.455%, while the GFI of the hybrid tree-

like MUM is 91.964%. The summary statistical features of the first-order MCUM are very similar to the reference set. However, from Figure 26.a we could see that the generated sequences are very different with original visiting sessions. And from this figure we could easily find out that many test sequences generated by the first-order MCUM do not exist in the reference set. In other words, these generated test sequences may be not feasible. This also confirms the weakness of the first-order MCUM presented in Section 3.1.

Figure 26.b shows the test sequences generated from the simplified hybrid tree-like MUM which we presented in [1]. The model yields 87.455% for the GFI in the experiment (see Section 6.2). Compared with original visiting sessions, the overview and summary statistical features are quite similar to the reference set. However, if we focus on some details, the differences are obvious. Please compare the part circled by the red rectangle in Figure 26.b and Figure 26.d. The user's behaviors in this part are ignored by the simplified hybrid tree-like MUM model.

Figure 26.c shows the test sequences generated from the hybrid tree-like MUM. Compared with the simplified hybrid tree-like MUM model, it has fewer states but capture user's behaviors better. As expected, both the graphical and statistical features obtained from the sequences of the hybrid tree-like MUM closely match the reference set. The user's behaviors ignored in Figure 26.b are well captured by optimized model (see red rectangle part in Figure b, c and d)

We also wonder whether the proposed model can accurately capture user's behav-

iors from limited visiting sessions and reproduce more test sequences for reliability test. In practice, if we have large amount of training data, we can use the data to test the software directly, test case generation seems not necessary. In fact, the training data is always expensive and limited. Therefore, the capability of capturing the user's behaviors from limited data is very important. Thus we did another very interesting experiment. We randomly selected 10,000 visiting sessions to train the model (probability threshold 5%, frequency threshold 25 and confidence level 95%). We randomly selected another 20,000 visiting sessions as reference set. We generated 20,000 test sequences from the hybrid tree-like MUM and analyzed the difference between generated test sequences and original visiting sessions. The results are reported in Table 20 and Figure 27.

Table 20 Statistical features of test sequences generated by hybrid tree-like MUM trained by 10000 visiting sessions

Characteristics	Reference	Generated
Num. of Sessions	20,000	20,000
Num. of Request	172,178	166,417
Num. of Application State Sequences	6,664	6,966
Ave. Session length	8.688	8.398
Max. Session length	100	62
Standard Deviation of Session length	15.777	14.992

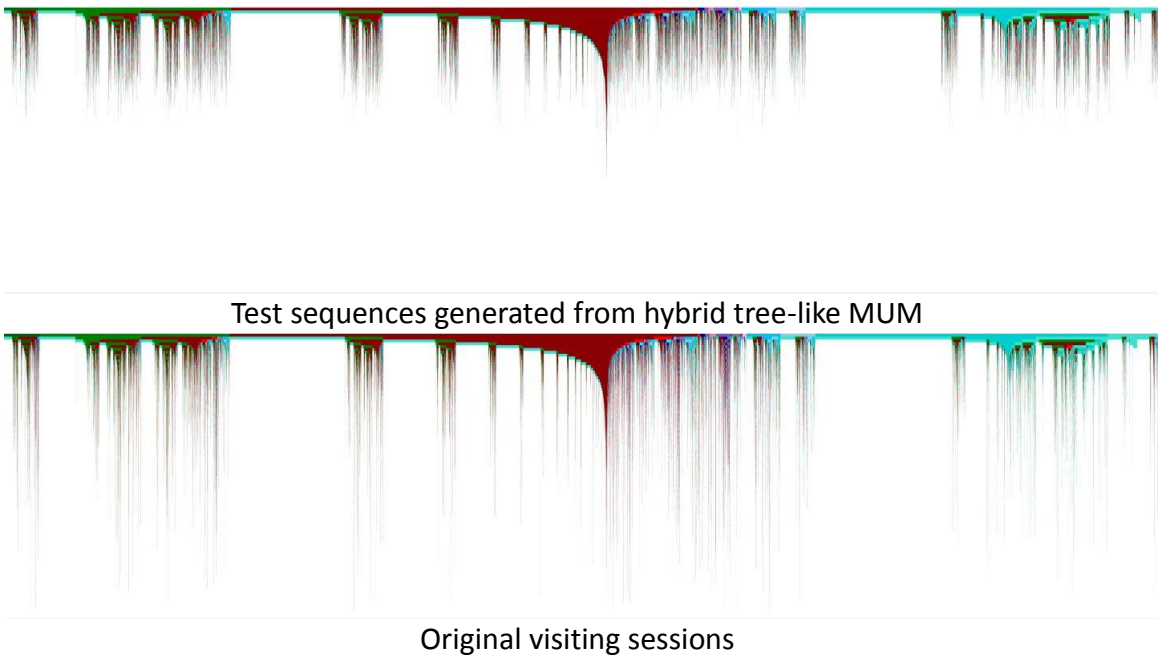

Figure 27 Generated Test Sequences from the hybrid tree-like MUM trained by 10000 visiting sessions

The experimental result shows that, even if the model is trained by a small training sample, it still captured the user's behaviors very well. However, we also are aware that when the training sample is too small our model, of course, cannot capture the usage pattern precisely.

6.4 Selecting the Best Model Parameters

In parameter-based usage models, using different parameters with the same training rules leads to different models. In this section we use our hybrid tree-like Markov us-

age model to illustrate how the Goodness-of-Fit Index can be used to optimize parameter-based usage models. As explained in Chapter 3, the hybrid tree-like Markov usage model has parameters (frequency-pruning and Cochran criterion) that can lead to a usage model that has a very much reduced "upper tree" and injects more information into the "lower Markov model". Therefore, the frequency-pruning thresholds play a crucial role for model accuracy and size. In our previous sections, we introduced these parameters but did not discuss how to set them.

The elimination of low-frequency branches in the upper tree can be done via two parameters: the probability threshold θ and the frequency threshold c. When the conditional probability of a branch is lower than θ and the frequency of a branch is lower than c, the branch is cut.

Table 21 The trends of model size and model accuracy with different parameters

(a) GFI matrix

θ\c	max	500	200	100	50	25	0
1.00	72.84[9]	87.96	89.87	90.84	91.47	**91.96**	92.73
0.20	83.14	90.96	91.42	91.94	92.23	92.41	92.73
0.15	86.72	91.13	91.58	92.04	92.31	92.46	92.73
0.10	86.98	91.48	91.9	92.17	92.36	92.5	92.73
0.05	**87.45**	91.82	92.21	92.36	92.48	92.57	92.73
0.00	92.73	92.73	92.73	92.73	92.73	92.73	92.73

(b) Model size matrix

θ\c	max	500	200	100	50	25	0
1.00	32[10]	87	160	278	485	**898**	2,051
0.20	191	756	980	1,201	1,406	1,694	2,051
0.15	520	917	1,123	1,360	1,537	1,769	2,051
0.10	589	1,121	1,331	1,532	1,677	1,835	2,051
0.05	**950**	1,416	1,633	1,738	1,824	1,927	2,051
0.00	2,051	2,051	2,051	2,051	2,051	2,051	2,051

Table 21 shows how model accuracy (a) and the model size (b) change with different parameter values. The rows show how these values evolve for c with a fixed θ and the lines show the other way around. Each branch is pruned if its conditional probability is lower than θ and the branch frequency is lower than c. The matrices show us that with θ and c decreasing, the accuracy of the model increases but so does its size. One can then choose parameters that yield a good enough accuracy (e.g., usually above 90%) with a fairly small model size.

The experiment result implies that compared with probability threshold the fre-

[9] When θ =1.00 and c = max, all branches are cut. Here we fill the GFI of first-order Markov chain usage model as a reference.

[10] When θ =1.00 and c = max, all branches are cut. Here we fill the size of first-order Markov chain usage model as a reference.

quency threshold results in smaller model size and better accuracy. That is, the probability threshold is likely to prune some branches which carry significant information, especially towards the upper part of the tree. The comparison of hybrid tree-like MUM with $\theta=0.05$ and tree-like MUM with $c=25$ discussed in Section 6.3 illustrates this issue clearly. In Figure 26.b, the red rectangle circles the minor branches cut by probability threshold ($\theta=0.05$). In Table 22 these two usage models are highlighted using red bold font.

Chapter 7 Conclusion and Future Work

7.1 Summary of the Research

In this thesis, 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 (0.4% of TS model) than the original tree of se-

quences, 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.

In this thesis, we also presented a method to assess the accuracy of various Markov usage models. The goal is to help test designers to select a practical usage model for web application reliability testing. The method uses the chi-square value to measure the distance between the probability distributions of predicted by the usage model and the probabilities observed in a testing sample. We use the Goodness-of-Fit Index (GFI) to position the proposed model on a scale ranging from a worst usage model to an ideal model that fits all observations. We can position any Markov usage models on this scale to reflect its relative accuracy. Therefore, testers can now compare different Markov usage models and selected the one that yields an acceptable accuracy and has an acceptable size. It is also possible to use this approach to optimize the values of model parameters in order to find the best usage model instance.

In contrast to some empirical studies on the same question (such as [28]), we provide a statistical view of model fitness and accuracy that can be applied to all Markov

usage models. Compared with previous studies based on classical chi-square tests to estimate the model's accuracy (such as [27]), our method overcomes the following two weaknesses. First, the result of a chi-square test depends on the test sample size. When the test sample size is small, the chi-square test tends to accept the hypothesis that the model accurately captures the user's behavior. On the other hand, when the test sample is large, the chi-square test tends to reject the hypothesis. Our method reduces this impact of the sample size. Second, the chi-square test is a dichotomous decision based on the chi-square distribution relative to the degrees of freedom; it determines the confidence that the model accurately represents the behavior of the testing sample. The Goodness-of-Fit Index positions the accuracies of a model within a continuous closed interval which makes it easy to compare the accuracy of different usage models. Thus, it is easy to determine the best model or most suitable model for the simulation of the operational environment in reliability testing

7.2 Future work

The ultimate goal of this research is to test web application reliability automatically. In this work, we have solved the first a few problems, but we are still in an early stage of development. There are several areas to research.

Usage Model

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 hierarchical models such as Hidden Markov Model, in order to merge parts that are statistically identical but are included inside larger sequences that are not. Hidden Markov Models are particularly useful in modeling sequential data where we believe the observed sequence is actually dependent on separated phrases, especially in natural language parsing. For example, the meaning of sentence "Siri, I would like to text a short message to Bob" depends on the words Siri, text, message. In Hidden Markov model, these words are states with transition probabilities between them. And other words in the sentence are possible observations with emission probabilities "hidden" in the model. In our research the application state sequence may depends on several separated actions, while other actions can be considered possible observations. An efficient method of model merging for Hidden Markov model was presented by Andreas Stolcke and Stephen Omohundro [46]. 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.

Accuracy Estimation Method

Though the proposed approach addresses some other limitations (discussed in

Section 4.3) associated with chi-square testing under varying sample size condition, the sample size still impacts testing result. Table 14 and Table 15 show the GFI test results under different sample size. And the values of GFI test for model built from large samples are commonly larger than the values of GFI test for model built from samples. As we discussed in Section 4.4, we are also aware that the bias of estimated accuracy is impacted by the number of folds in k-folds cross validation. We select the fold number based on empirical study. In future work, we would like to discover the relationship between the value *k* of k-fold cross validation and the estimation bias, in order to improve the accuracy of the estimation method.

Test Sequence Generation

Although we developed a prototype tool which can produce abstract test sequences, they still cannot be used for reliability testing or statistical usage testing without input parameters. There is much work to be done to make the prototype tool to produce executable test cases, particularly in adding parameters into test sequences. We have extracted those input parameters from log files. The next step is to add the parameters into the usage model in order to generate executable test sequences.

References

[1] G.Bochmann, G-V.Jourdan and B.Wan. “Improved Usage Model for Web Application Reliability Testing”, The 23rd IFIP Int. conference on Testing Software and Systems (ICTSS'11), 2011.

[2] B.Wan, G.Bochmann and G-V.Jourdan “Evaluating Reliability-Testing Usage Models”, the 36th Annual International Computer Software and Applications Conference (COMPSAC 2012), Izmir, Turkey.

[3] M.L.Shooman , Quality and Reliability Engineering International, Mcgraw-hill, 1983, PP 351-353

[4] M.L.Shooman, “Probabilistic Reliability: An Engineering Approach”, New York: McGraw-Hill Book Co., 1968 (Second edition, Melbourne, FL: Krieger, 1990).

[5] JD.Musa, A.Iannino, K.Okumoto, Engineering and Managing Software with Reliability Measures, McGraw-Hill, 1987

[6] A.Goel, K.Okumoto, "A logarithmic Poisson execution time model for software reliability measurement" Reliability, IEEE Transactions on, Volume: R-28, Issue: 3,On Page(s): 206 - 211,Aug. 1979

[7] JD. Musa and K.Okumoto, “A Logarithmic Passion Execution Time Model for Software Reliability Measurement”, Bell Laboratories, Whippany.1992

[8] J.H. Poore and C.J. Trammell, “Application of Statistical Science to Testing and Evaluating Software Intensive Systems”, Statistics, Testing, and Defense Acquisition, National Academy Press, Washington D.C., 1998.

[9] K.Sayre. Improved Techniques for Software Testing Based on Markov Chain Usage Models. PhD thesis, University of Tennessee. (1999)

[10] G.H. Walton, J. H. Poore, C J. Trammell, “Statistical testing of software based on a usage model” Software: Practice and Experience, Volume 25, Issue 1, pages 97–108, January 1995

[11] J.A. Whittaker, M.G. Thomason, “A Markov Chain Model for Statistical Software Testing,” IEEE Trans. Software Eng., Vol. 20, No. 10, pp.812–824.(1994)

[12] K. L.Chung, F.AitSahlia, “Elementary probability theory: with stochastic processes and an introduction to mathematical finance”. Springer. pp.129-130. ISBN 978-0387955780, 2003

[13] J. G.Kemeny and Snell, J. L. (1960). Finite Markov Chains. D. Van Nostrand, Princeton,New Jersey.

[14] W. Feller, (1968). An Introduction to Probability Theory and Its Applications, volume I. John Wiley & Sons, New York, third edition.

[15] C. Roads. “The Computer Music Tutorial”. MIT Press. ISBN 0-262-18158-4. 1996

[16] J. Hopcroft J.Ullman. “Introduction to Automata Theory, Languages, and Computation.” Addison-Wesley, 1979.

[17] J.Borges and M.Levene “Data Mining of User Navigation Patterns” WEBKDD’99, LNAI 1836, pp. 92–112, 2000

[18] C. S. Wetherell. Probabilistic languages: A review and some open questions. Computing Surveys, 12(4):361–379, December 1980.

[19] E.Charniak, “Statistical Language Learning”. The MIT Press, Cambridge, Massachusetts. (1996)

[20] K.S. Trivedi “Probability and Statistics with Reliability, Queuing, and Computer Science Applications”, John Wiley and Sons, New York, 2001. ISBN number 0-471-33341-7

[21] R. M. Smith, K. S. Trivedi, and A. Ramesh “Performability Analysis: Measures, An Algorithm and a Case Study", IEEE Transactions on Computers, Vol. C-37, No. 4, pp. 406-417, Apr. 1988.

[22] K.Goseva-Popstojanova and K.S.Trivedi, “Failure Correlation in Software Reliability Models”, IEEE Trans. on Reliability, Vol.49, pp. 37-48.(2000)

[23] 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)

[23] K.Sayre and J.Poore, “A Reliability Estimator for Model Based Software Testing”, 13th Int’l Symp. Software Reliability Engineering, pp. 53-63, 2002.

[24] W.Dulz and 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.

[25] 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)

[26] M. Deshpande, G. Karypis, “Selective Markov Models for Predicting Web-Page Accesses”, ACM Transactions on Internet Technology Volume 4 Issue 2, May 2004

[27] J.Borges, M.Levene, "Testing the Predictive Power of Variable History Web Usage", Soft Computing - A Fusion of Foundations, Methodologies and Applications - Web intelligence and change discovery Volume 11 Issue 8, March 2007

[28] S.Sprenkle, L.Pollock and L.Simko, "A study of Usage-Based Navigation Models and Generated Abstract Test Cases for Web Application”, Fourth International Conference on Software Testing, Verification and Validation, 2011

[29] L.T.Peter and James E. Pitkow: “Distributions of surfers’ paths through the world wide web: Empirical characterizations.” World Wide Web pp. 29–45. 1999

[30] S.Jespersen, T.B.Pedersen, and J. Thorhauge, “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)

[31] 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)

[32] R. E.Walpole and Raymond H.Myers, “Probability and Statistics for Engineers and Scientists”, Fifth Edition, published byMacmillan publishing company (1993)

[33] K. Zhang, D. ShaSha, “Simple Fast Algorithm for the Editing Distance Between Trees and Related Problems” SIAM J. COMPUT. Vol. 18, No. 6, pp. 1245-1262, December 1989

[34] B.McGaw, & K. G. Joreskog, "Factorial invariance of ability measures in groups differing in intelligence and socioeconomic status”. British Journal Mathematical and Statistical Psychology, 1971, 24, 154-168.

[35] L.Hu, & P.M.Bentler, “Evaluating mode fit”. In R. H. Hoyle (Ed.), Structural equation modeling: Concepts,issues and applications (pp. 76–99). Thousand Oaks, CA: Sage 1995.

[36] P.M.Bentler and D.G.Bonett, “Significance Tests and Goodness of Fit in the Analysis of Covariance Structures” Psychological Bulletin 1980 Vol. 88, No. 3 588-606.

[37] E. P. George,“Empirical Model-Building and Response Surfaces”, co-authored with Norman R. Draper, p. 424, ISBN 0471810339, 1987.

[38] D.H. Wolpert, "The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework" Technical report The Santa Fe Institute, Santa Fe, NM, 1994.

[39] P.M.Bentler, “Comparative fit indexes in structural models.” Psychol Bull. 1990 Mar;107(2):238-46.

[40] Geisser, Seymour. “Predictive Inference”. New York, NY: Chapman and Hall. ISBN 0412034719.1993

[41] R.Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, International Joint Conference on Artifical Intelligence”, (IJCAI), 1994.

[42] C.Günst, M.A.Fund “Elaboration d’un modele stochastique d’usage pour une application Web apartir de l’activite utilisateur”, project report, 2011.

[43] P. Pirolli, J. Pitkow, and R. Rao. Silk from a sow’s ear: Extracting usable structures from the Web. In Proc. of 1996 Conference on Human Factors in Computing Systems (CHI-96), Vancouver, British Columbia, Canada, 1996.

[44] L. Catledge and J. Pitkow. Characterizing browsing behaviors on the World Wide Web. Computer Networks and ISDN Systems, 27(6), 1995.

[45] J.Borges and M.Levene, M: “A dynamic clustering-based Markov model for web usage mining”. In CoRR:the computing research repository. cs.IR/0406032 (2004).

[46] A. Stolcke, O.Stephen, “Best-first Model Merging for Hidden Markov Model Induction”, Tech. Rep.TR-94-003, ICSI, University of California, Berkeley, 1994

Appendix A: Mathematical Deduction of Rule of Succession

If $x_1, x_2, \ldots, x_n$ are conditionally independent random variables that each can assume the value 0 or 1, through observation, there is s variables are 1. Then, if we know nothing more about them, calculate the probability of next independent random variable equals to 1.

According to Maximum Likelihood Function:

$$f(x_1, x_2, \ldots, x_n; \theta) = p\{X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\} = \prod_{i=1}^{n} p(x_i; \theta) \qquad (1)$$

Since the random variable's value is 0 or 1, it is Bernoulli trial:

$$\prod_{i=1}^{n} p(x_i; \theta) = \prod_{i=1}^{n} \binom{n}{i} \theta^i (1-\theta)^{n-i} \qquad (2)$$

If all random variables equal 1: $f(x_1, x_2, \ldots, x_n; \theta) = (1-\theta)^n$. If *s* random variables equal 1: $L(x_1, x_2, \ldots, x_n; \theta) = \theta^s(1-\theta)^n$. Therefore, the probability density function of n random variables with s random variables equal 1 are:

$$f(X; \theta) = \theta^s(1-\theta)^{n-s} \qquad (3)$$

The marginal distribution of $f(\mathrm{X}; \theta)$ is

$$g(x_1, x_2, \ldots, x_n) = \int_{-\infty}^{+\infty} f(x_1, x_2, \ldots, x_n; \theta)\, d\theta \qquad (4)$$

Replace $\int_{-\infty}^{+\infty} f(x_1, x_2, \dots, x_n; \theta)$ by formula (3)

$$g(x_1, x_2, \dots, x_n) = \int_0^1 \theta^s (1-\theta)^{n-s}\, d\theta \qquad (5)$$

In probability theory, Bata Function is defined as:

$$B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1} dt \qquad (6)$$

Let x = s+1, y = n-s+1. Replace marginal distribution by Bata Function:

$$\int_0^1 \theta^s (1-\theta)^{n-s}\, d\theta = B(x, y) \begin{cases} x = s+1 \\ y = n-s+1 \end{cases} \qquad (7)$$

According to Bata Function's property:

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} = \frac{\Gamma(s+1)\Gamma(n-s+1)}{\Gamma(n+2)} \qquad (8)$$

where $\Gamma(z) = \int_0^{+\infty} t^{z-1} e^{-t} dt$; When z is positive integer, $\Gamma(z) = (z-1)!$. Since x and y are positive integers, then marginal distribution of $f(\mathrm{X}; \theta)$:

$$g(x_1, x_2, \dots, x_n) = \frac{\Gamma(s+1)\Gamma(n-s+1)}{\Gamma(n+2)} = \frac{s!\,(n-s)!}{(n+1)!} \qquad (9)$$

With theorem of Bayesian Estimation:

$$f(\theta|x_1, x_2, \dots, x_n) = \frac{f(x_1, x_2, \dots, x_n; \theta)}{g(x_1, x_2, \dots, x_n)} \qquad (10)$$

Then:

$$f(\theta|x_1, x_2, \dots, x_n) = \frac{\theta^s(1-\theta)^{n-s}}{\frac{s!\,(n-s)!}{(n+1)!}} = \frac{\theta^s(1-\theta)^{n-s}(n+1)!}{s!\,(n-s)!} \qquad (11)$$

Replaced by Bata Function again, the equation 11 can mutate as:

$$f(\theta|x_1, x_2, \dots, x_n) = \theta^s(1-\theta)^{n-s}\frac{1}{B(s+1, n-s+1)} \qquad (12)$$

According to the definition of Bata distribution function, equation (12) is Bata distribution function, showing $f(\theta, s+1, n-s+1)$.

Since law of total probability: $p_r(A) = E[p_r(A|f_x)]$, then:

$$f(\theta) = E[f(\theta|x_1, x_2, \dots, x_n)] = E[f(\theta, s+1, n-s+1)] \qquad (13)$$

The expectation of Bata distribution function is $f(\theta, x, y) = \frac{x}{x+y}$. Hence, $E[f(\theta, s+1, n-s+1)] = \frac{s+1}{s+1+n-s+1} = \frac{s+1}{n+2}$.

Consequently, the prior probability estimated:

$$f(\theta) = \frac{s+1}{n+2} \qquad (14)$$