Software Reliability Test Based on Markov Usage Model

Kuanjiu Zhou
[1]School of Software, Dalian University of Technology, Dalian, China
Email: zhoukj@dlut.edu.cn

Xiaolong Wang[1], Gang Hou[1*], Jie Wang[1] and Shanbin Ai[1]
Email: {wxl_dut, hg.dut, wangjie1003, aishanbin}@163.com

***Abstract*—The reliability test for embedded software system is very difficult due to its complex structure and large scale. Markov usage model which describes the software usage with Markov process is widely used for statistical test. Software statistical test based on Markov usage model is an effective approach to the generation of test cases with high efficiency and the evaluation of software reliability in a quantitative way. This paper mainly focuses on the generation of Markov usage model of software system and the method of software reliability test based on it. Firstly, a method to build Markov usage model based on improved state transition matrix (STM), which is a table-based modeling language, is proposed. Then a software reliability test method including test case generation and test adequacy determination based on Markov usage model is presented. An improved Kullback discriminant is selected as the judgment criteria of convergence from the test chain to the usage chain in order to measure whether the testing process is sufficient. Finally, a Markov test cases generation tool MTCG is developed which implements the methods put forward in this paper. Experimental verification of test adequacy and efficiency is made through a comparison between the Markov usage model-based method and a completely random test method, the results of which show that software reliability test based on Markov usage model is approving, high-efficient and promising.**

***Index Terms*—Markov Usage Model, Software Reliability Test, Test Case Generation, Test Adequacy**

I. INTRODUCTION

The reliability of embedded software system attracts more and more attention since it is widely applied in aerospace, military, medical, communications, nuclear reactions, and industrial control areas. Due to the large scale and complex structure of embedded software system, once the software system fails, inestimable consequences would be caused, such as loss of life and property, severe damage to the natural environment. For example, in June 1996, the European Space Agency's Ariane 5 launch vehicle exploded catastrophically after 37 seconds took-off because of software failures. In the mid-1980s, the United States Therac-25 radiation therapy went out of order in 2nd treatment mode, resulting in a severe medical malpractice of two deaths and several injuries, because its security system had not been tested sufficiently after its interlock device was replaced. Most of those accidents are owing to inadequate software testing, which cannot guarantee the software reliability effectively.

Different from traditional software testing, software reliability test requires the software usage model to generate test cases. Software reliability can be evaluated according to the statistical results of software testing. There are three basic problems of software reliability test, the reliability test model, test adequacy and test case generation [1-3].

Software testing method based on usage model can effectively reduce the number of test cases, save testing costs and improve software reliability, since it is much closer to the actual usage of software system than the completely random testing method. Markov usage model is a commonly used model in software reliability test, which is widely used in the fields such as information theory, automatic control, communication techniques, computer sciences and genetics [4].

Much research has been done on Markov usage model. In 1994, Whittaker and Thomason applied Markov chain to software reliability test for the first time [5]. Wang Ji and Chen Huowang proposed a method of Markov chain usage model based on UML use case diagram and state diagrams which provides formal description base for testing UML-based software [6]. Gao Haichang used the Kullback discriminant as the convergence judgment criteria of the test-chain to usage-chain, and testified the inevitability of the convergence theoretically [7]. Murali Krishna proposed a new semi-Markov process-based model to compute the network parameters such as saturation throughput for the IEEE 802.11 Distributed Coordination Function (DCF) employing the Binary

This work is supported by the National Natural Science Foundation of China (Grant No. 91018003) and the Central University Basic Research Special Foundation (Grant No. 1600-852007).

* Corresponding author: Email: hg.dut@163.com

doi:10.4304/jsw.7.9.2061-2068

Exponential Backoff (BEB) [8]. He Yiming and Pu Yingxia applied Markov chains and Spatial Markov chain to investigate the spatial and temporal characteristics of industrial structure level [9]. Liu YS proposed the CSP (checkpoint-based spare processor) model which is an improvement of the traditional SP (spare processor) model to solve the fault-tolerant problem of SP [10]. Li Mingshu proposed an adaptive multilateral negotiation model for software process modeling, namely AMNM-PA, which is based on non-stationary finite-horizon Markov decision processes and uses the model-independent Q learning algorithm to choose negotiation strategies [11]. Mo YC and Yang XZ demonstrated how to analyze mission reliability of the generalized PMS with random phase duration and Markov regenerative intra phase processes to attack the weak points of the state-of-the-art [12]. Liu Y and Ma ZY and Shao WZ proposed an approach to transforming UML diagrams of software architecture to Markov chain for the quantitative evaluation of reliability [13]. Kong DG presented a method based on Hidden Markov model for the analysis of time sequences in multithread programs [14]. Zhang DP and Xia CH proposed an effective method for computing optimal state transition probabilities for software reliability estimation based on a Markov usage model [15]. Xu Libo and Wu Guoxin studied the performances of various congestion control strategies through building accurate Markov chain network models for TD and RED arithmetic [16]. Brian and Yariv presented an expectation-maximization procedure for estimating the generator of a bivariate Markov chain and compared the performance of the estimation algorithm to an earlier approximate estimation procedure based on time sampling [17]. Zhang Youzhi described a hidden Markov model in the field of data mining applications [18]. In addition, Tian Xinguang and Wang Zikun did lots of contributions to the application of the Markov model [19-22].

Our major work in this paper focuses on the construction and verification of Markov usage model, and how to generate test cases as few as possible based on the usage model in the premise of ensuring test adequacy in order to improve the efficiency of embedded software reliability test.

This paper is structured as follows. Section II presents an introduction of Markov usage model and proposes a modeling method for Markov usage model based on STM. In Section III, we put forward a software testing method based on Markov usage model, including test case generation and test adequacy determination. In Section IV, we carry out an experiment with an independently developed test case generation tool *MTCG* and make verification on the efficiency and adequacy of software reliability test based on Markov usage model. The conclusions are drawn in Section V.

II. MARKOV USAGE MODEL

A. Introduction of Markov Usage Model

Markov usage model is a classic software usage model, which describes the software usage with Markov process.

In probability and statistics theory, if the state of a process or a system at the moment t_0 is known and its state at the moment t has no relationship with any other states before t_0 ($t > t_0$), we can conclude that the process is a Markov process and the process or the system has Markov property, which is also called memoryless property.

The Markov process can be described with a distribution function. Assume that a stochastic process $X(t)$ ($t \in T$) is a Markov process and has a state space I. For $t_1 < t_2 < \ldots < t_n$ ($n \geq 3$, $t_i \in T$), the conditional probability distribution of $X(t_n)$ under the conditions that $X(t_i) = x_i$ ($x_i \in I$, $i = 1, 2, \ldots, n-1$) equals its conditional probability distribution under the condition that $X(t_{n-1}) = x_{n-1}$. That is to say, for $x_n \in R$,

$$P\{X(t_n) \leq x_n \mid X(t_1) = x_1, X(t_2) = x_2 \ldots, X(t_{n-1}) = x_{n-1}\} = P\{X(t_n) \leq x_n \mid X(t_{n-1}) = x_{n-1}\} \quad (1)$$

The Markov chain is the Markov process with discrete time and states, which can be denoted by $\{X_n = X(n), n=0, 1, 2, \ldots\}$. Assume that a Markov chain has a state space $I=\{a_1, a_2, \ldots\}$ $(a_i \in R)$, the Markov property can also be described with the conditional probability distribution as follows:

$$P\{X_{m+n} = a_j \mid X_{t1} = a_{i1}, X_{t1} = a_{i1}, \ldots, X_{tr} = a_{ir}, X_m = a_i\} = P\{X_{m+n} = a_j \mid X_m = a_i\} \quad (2)$$

Here, n, r $\in N^+$, $0 \leq t_1 < t_2 < \ldots < t_r < m$.

The conditional probability P_{ij} denotes the transition probability of the Markov chain from state a_i at the moment m to state a_j at the moment $m+n$. The definition of P_{ij} can be described as follows:

$$P_{ij}(m, m+n) = P\{X_{m+n} = a_j \mid X_m = a_i\} \quad (3)$$

As the Markov chain transfers to one of the states in $\{a_1, a_2, \ldots\}$ at the moment $m+n$ from the state a_i at the moment m inevitably, the following equality can be discovered:

$$\sum_{j=1}^{\infty} P_{ij}(m, m+n) = 1, (i = 1, 2, \ldots) \quad (4)$$

A matrix constituted of transition probability is called the transition probability matrix of a Markov chain. The transition probability matrix is of great use in the generation of test cases and in the determination of test adequacy. Based on (4), we can obviously conclude that the summation of the elements value in each row is 1 in a transition probability matrix.

Integrated the concepts presented above, the Markov usage model is a sequence of states and transitions with the key characteristic that the appearance probability of state s_i depends only on the previous state and is independent of any other history states.

Fig. 1 gives an example of Markov usage model. {*START, A, B, C, EXIT*} is the state space, in which *START* denotes the beginning state of the software usage and *EXIT* denotes the ending state. {(*a*, 1.0), (*b*, 0.5), (*c*, 0.5), (*d*, 0.75), (*e*, 0.25), (*f*, 0.5), (*g*, 0.25), (*h*, 0.25)} is the transition set of the usage model and the decimals

denote the transition probability. Especially, for an embedded software system, the transitions or events in the transition set can be used to denote interrupts or messages.

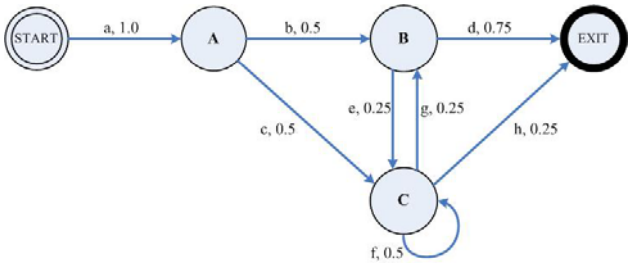


Figure 1. An example of Markov usage model.

B. Modeling Method Based on STM

One of the key problems in software testing based on Markov usage model is how to construct the model in the requirements modeling. Nowadays, some useful research in the Markov modeling has been done in Reference [1-2, 5-7]. In this paper, we present a method of modeling Markov usage model based on State Transition Matrix (STM) [23-24]. STM is a table-based modeling language for describing behavior of distributed systems [25-26]. Kong Weiqiang has given a detailed description of STM [27], and uses STM to model checking for embedded software. In this paper, we improve the form of STM by adding the probability of events, in order to suit for the generation of Markov usage model.

Programming language and table notations can be mixed for specifying a system design in STM [28-29]. In this paper, we consider a subset of STM with the motivation of giving a precise description that can be formally verified efficiently. In order to formally describe UML state machines, we formalize the structure of STMs and the dynamic behavior of their design, adopting a similar approach used in Reference [27]. Fig.2 shows an example of STM.

S / E	S_1	S_2	…	S_m
E_1	P_{11}	P_{21}	P_{i1}	P_{m1}
	/	Case1 / S_i / $Action_i$; Case2 / S_j / $Action_j$; Case3 / S_k / $Action_k$	×	/
E_2	P_{12}	P_{22}	P_{i2}	P_{m2}
	×	/	Case1 / S_i / $Action_i$; Case2 / S_j / $Action_j$; Case3 / S_k / $Action_k$	/
…	P_{1j}	P_{2j}	P_{ij}	P_{mj}
	/	/	×	/
E_n	P_{1n}	P_{2n}	P_{in}	P_{mn}
	/	/	/	×

Figure 2. An example of state transition matrix(STM).

We firstly describe an action language L that is needed for defining the structure of STMs. L is chosen to be a simple subset of ANSI C language and thus the syntax and semantics follow the conventions of C programming language. Type system of L consists of Boolean, Integer, and Real. Supported expressions of L are (A) Boolean literals true and false, Integer literals and Real literals, (B) Variable identifiers, and (C) infix expressions *leftexpr op rightexpr*, where *op* can be one of +, -, *, /, %, &&, ||, >, <, >=, <=, = =, or !=, with the semantics of ANSI C. Supported *statements* of L are (A) assignments of the form *lhs* = *rhs*, and (B) if statements of the form if *condition {statement1}* else *{statement2}*.

Assuming this action language L, the structure of a STM H is a tuple $<S,E,P,C>$, where

- S is a finite set of state. Each state $s \in S$ is associated with a (unique) index number denoted by $index(s) \in Nat$. During execution of H, only one state denoted by $active(H)$ is active. Initially, the active state is s where $index(s) = 0$.
- E is a finite set of events consisting of external events $E_{external}$ and internal events $E_{internal}$, where $E_{external} \cap E_{internal} = \phi$. An event $e_E \in E_{external}$ is represented by a Boolean variable of L (whose name is prefixed with a lower-case "x" by convention), and an event $e_I \in E_{internal}$ is represented by a Boolean expression (possibly a Boolean variable) of L. Each event is associated with a (unique) index number denoted by $index(e) \in Nat$.
- P is a finite set of transition probability. $P_{ij} \in P$ means the probability of event E_j occurs, while in the state S_i. Here, the value of P_{i1} is limited by two types of constraint, one is structure constraint, and the other is usage constraint. Structure constraint reflects the basic properties of Markov usage chain, mainly including (A) $0.001<=P_{i1}<=0.999$ and (B) the summation of P_{i1}, P_{i2}, ..., P_{in} is 1. Structure constraint is fitted for all software system. Usage constraint reflects the actual usage of software system, including (A) linear constraints such as $P_{2,4} = 2P_{2,3}$, $P_{3,5} = P_{1,3} + 2P_{1,5}$, and (B) interval constraints such as $0.25<=P_{3,1}<0.35$, $0.5<P_{3,1}<=0.85$. Usage constraint can be obtained from requirements analysis and usage statistics of software system. Based on these constraints above, all the transition probability in P can be figured out by solving the equation constituted of inequality constraints.
- C is a finite set of cells consisting of Normal cells C_{normal}, Ignore cells C_{ignore}, and Invalid cells $C_{invalid}$. Each normal cell $c_N \in C_{normal}$ is a tuple $< s, e, u, a, s' > \in S \times E \times (L \cup \{null\}) \times (L \cup \{/\}) \times S$. We define $source(c_N)=s$, $event(c_N)=e$, $guards(c_N)=u$, $actions(c_N)=a$, and $target(c_N)= s'$. Each Ignore cell is a tuple $< s, e, />$, and each Invalid cell is a tuple $< s, e, \times>$. Functions *source* and *event* are also defined for Ignore and Invalid cells $c_I \in C_{ignore} \cup C_{invalid}$ as for c_N, but *guards*, *actions* and *target* are not.

Events of a STM and guards of a Normal cell are expressed as L expressions of type Boolean, and actions of a Normal cell are a list of L statements. A cell c of a STM H, which is pinpointed by its index numbers ($index(source(c))$, $index(event(c))$) together with its guards if $guards(c) \neq$ null[1], specifies the behavior of H (under condition of $guards(c)$ if available) when event

event(*c*) is dispatched while *H* is in state *source*(*c*) (i.e., *active*(*H*) = *source*(*c*)). If $c \in C_{normal}$, *actions*(*c*) will be executed[2] atomically and after that, *H* moves to state *target*(*c*). If $c \in C_{ignore}$, denoted in a STM table by the symbol "/", nothing changes. If $c \in C_{invalid}$, denoted in a STM table by the symbol "×", an error occurs. Informally, an Ignore cell means that the dispatch of an event in a state is ignored, and an Invalid cell means that the dispatch of an event in a state is never possible.

III. Software Testing Methods Based on Markov Usage Model

The Markov usage model constructed with the methods given above provides an essential model for the reliability testing of embedded software. The method of test case generation based on Markov usage model is the key problem in reliability test of embedded software and the most important job we focus on in this paper.

A. Algorithm of the Generation of Test Cases

In the generation of test cases for embedded software based on Markov usage model, a test case is a sequence of states or transitions passing though the Markov test chain from the beginning state to the ending state. The Markov test chain is formed from the usage model as follows:

Firstly, the test chain is built based on the structure of the usage chain and each transition probability of the edges is replaced to a counter with an initial value 0. Then, as the testing process goes forward, the counter of the edge increases by 1 if a test case goes through it. Finally, the counter of each edge is converted to transition probability based on the statistical results.

In this paper, we present an algorithm of test case generation based on Markov usage model in software reliability test as follows:

Step 1: Let *st_curr* denote the current state and set the value of *st_curr* = *START*.

Step 2: Generate a uniform random decimal *rnd* between 0 and 1 at the state *st_curr* and select a transition with the strategy in Step 3.

Step 3: Based on all the transitions $\{(t_k, p_k)\}$ (k=1, 2, …, m and k, $m \in N^+$) from the state *st_curr* to the state *st_next* in the usage model, divide the interval [0, 1) into m intervals $\{I_1, I_2, \ldots, I_m\}$. Here $I_1=[0, p_1)$, $I_2=[p_1, p_1+p_2)$,…, $I_m=[p_1+p_2+\ldots+p_m, 1)$. If rnd$\in I_k$, record the transition t_k and set *st_curr* = *st_next*.

Step 4: Repeat Step 2 and Step 3 until the value of *st_curr* equals *EXIT* and all the transitions recorded in Step 3 form a test case.

Step 5: Repeat all the steps above to generate enough test cases until the testing process is sufficient.

In the algorithm given above, test cases are generated from uniform random decimals referenced to transition probability of Markov usage model. This algorithm not only satisfies the randomness of the test case generation process, but also takes the transition statistical probability of software usage into account. That is why we can generate test cases more corresponded to the actual usage of software system with a smaller quantity. Table I shows an example of a test case generated based on the Markov usage model in Fig. 1.

TABLE I.
AN EXAMPLE OF A TEST CASE GENERATED BASED ON MARKOV USAGE MODEL

Order	Transition/Event	Next State
1	a	A
2	c	C
3	g	B
4	e	C
5	f	C
6	h	EXIT

B. Feasibility of Test Case Generation Algorithm

In software testing based on Markov usage model, the key point to ensure the feasibility of test case generation algorithm is to generate random paths in which P_{tj} is equal to P_i. P_{tj} is the output edge transition probability of state *S* in the test chain, and a_{tj} is the corresponding edge to a_j in the usage chain.

Assume that the state *S* has *k* output edges, such as a_1, a_2, ..., a_k, P_1, P_2, ..., P_k are their corresponding probability. We have the following equality

$$\sum_{i=1}^{k} \mathrm{P_i} = 1 \tag{5}$$

Assume that the number of visiting state *S* in the test chain is n and the number of output edge a_j ($1 \le j \le k$)is m, corresponding to the Bernoullitheorem we have the conclusion that

$$\lim_{n\to\infty} \frac{m}{n} = P_j \tag{6}$$

According to the above formula, it can be ensured than the transition probability a_{tj} in the test chain is equal to the access probability P_{tj} in the usage chain when n is large enough as long as the random decimal generation obey the 0~1 uniform random distribution.

$$\mathrm{P_{tj}} = \int 1dx \Big|_{\sum_{k=1}^{j-1} P_k}^{\sum_{k=1}^{j} P_k} = \sum_{k=1}^{j} P_k - \sum_{k=1}^{j-1} P_k = P_j \tag{7}$$

It is obviously that P_{tj} is always equal to P_j when n is large enough as long as the random decimals generated for the path selection obey the 0~1 uniform random distribution.

C. Test Adequacy Determination Criteria

Test adequacy determination criteria become an essential problem that must be addressed in software reliability test. The test adequacy determination criteria based on Markov usage model is a model comparison criteria which comparing the usage environment (usage chain) and test environment (test chain). The test can be considered sufficient until the test chain impend over the usage chain.

The test chain *U* and the usage chain *T* can be seen as two different descriptions of Markov process in software

reliability test theory based on Markov usage model, so the Discriminant value can be applied to determine the adequacy of software testing. Discriminant value, also known as Kullback discriminant, is the mathematical expectation of convergence value of two random processes. For simplicity, discriminant value is denoted by *D(U,T)*, where *U* represents the usage chain, *T* represents the test chain. The value of *D(U, T)* is closer to 0, the test chain *T* is more similar to the usage chain *U*. The testing process is considered sufficient and can be stopped when the discriminant value is less than the expected threshold value. Discriminant value can be calculated accurately as follows:

$$D(U,T) = \lim_{n\to\infty} \frac{1}{n}[log(\frac{Pr(X_0,X_1,...,X_n|U)}{Pr(X_0,X_1,...,X_n|T)})] \quad (8)$$

$Pr(X_0, X_1, ..., X_n \mid \lambda)$ represents the circumstances of executing *n* set of test cases under the condition of probability distribution with parameter λ. The *D(U,T)* cannot be obtained directly by calculating the usage chain *U* and the test chain *T* and the test. It's equivalent formula as follows:

$$D(U,T) = \sum_{i=1}^{u} \pi_i \sum_{j=1}^{u} u_{i,j} log \frac{u_{i,j}}{t_{i,j}} \quad (9)$$

The $u_{ij,}$ and t_{ij} represent the transition probability from state *i* to state *j* in usage chain and test chain respectively. π_i represent the long-run occupation ratio of state *i*. Clearly, the state's occupation ratio in the long-run is equal to the sum of products of adjacent state's occupation ratio and the transition probability from the adjacent state to the state. And the sum of all state's transition probability in the long-run is equal to one. The calculate method as follows:

$$\pi_j = \sum_{i}^{n} \pi_i P_{i,j} \quad (10)$$

$$\sum_{i=1}^{n} \pi_i = 1 \quad (11)$$

The discriminant value of *D(U,T)* is not always calculable according to (9). If the test chain does not cover all edge of the usage chain-chain, the value of t_{ij} is zero and is not calculable. Therefore, only when the test chain covers all the edge, the *D(U, T)* is feasible and meaningful.

We adopt an improved Kullback discriminant, denoted *K(U,T)*, to be criteria for judging the adequacy of software reliability test. The *K(U,T),* as follows, is a variant calculation method to *D(U,T)* which is always calculable after modified.

$$K(U,T) = \sum_{i=1}^{u} \pi_i \sum_{j=1}^{u} u_{i,j} log \frac{u_{i,j}}{\varepsilon\text{-}\varepsilon(sgn(t_{i,j})) + t_{i,j}} \quad (12)$$

Here, ε is a sufficiently small positive decimal. *sgn (x)* represent a function: When $x = 0$, *sgn (x)* = 0; when $x < 0$, *sgn (x)* = 1; when $x > 0$, *sgn (x)* = 1. Obviously, when the test chain *T* does not cover all the edges, *K(U, T)* is calculable as usual. When the test chain *T* covers all edges, ε-ε *(sgn* $(t_{i,j})$*)* = 0. Thus *K(U, T)* is equal to *D(U,T).*

The closer is the test chain *T* to the usage chain *U*, the nearer to zero is the *K(U,T)*. The convergence threshold can be set properly and acceptably according to the parameters of different software systems. When the value of *K(U,T)* is less than the threshold, the test adequacy can be considered reached and the testing process can be stopped.

IV. Experiments and Verification

In this paper, a generation tool of test cases based on Markov usage model, named *MTCG* (Markov Test Cases Generator), was successfully developed and implemented in C# language with Microsoft Visual Studio 2008 integrated development environment and .NET Framework 3.5 framework. This test case generation tool *MTCG* enable software testers to customize and verify a Markov usage model of software system based on STM and then automatically draw a graph for the usage model. However, the main function of *MTCG* is to generate test cases based on the usage model customized above automatically and get the statistical information in testing process in order to evaluate and improve the reliability of the software. Using *MTCG*, we carried out an experiment of software testing based on the usage model shown in Fig. 1 and analyzed the adequacy of the testing process and the efficiency of the testing method proposed in this paper.

A. Adequacy Verification of Software Reliability Test Process

Based on the improved Kullback discriminant proposed in section III, an experiment was carried out with ε=0.00001 and *K(U, T)* threshold =0.001 to test and verify the adequacy of the testing process. Table II shows the statistical results in the experiment, including the node coverage, edge coverage and the value of *K(U, T)* with different number of test cases.

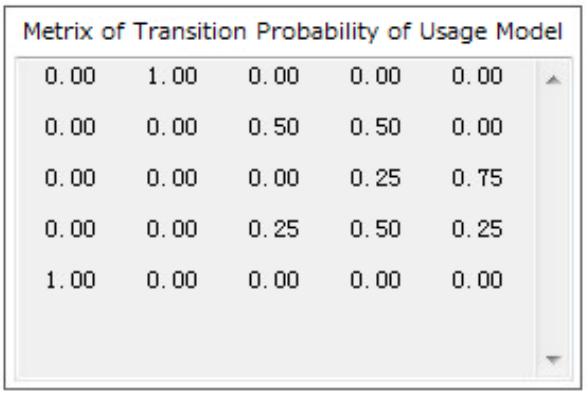
TABLE II.
STATISTICAL RESULTS OF TEST CASES GENERATED BASED ON MARKOV USAGE MODEL

No.	Number of Cases	Node Coverage	Edge Coverage	K(U,T)
1	1	0.8000	0.5000	0.8055
2	2	1.0000	0.8750	0.6643
3	3	1.0000	1.0000	0.5928
4	4	1.0000	1.0000	0.1807
5	5	1.0000	1.0000	0.0706
6	10	1.0000	1.0000	0.0123
7	20	1.0000	1.0000	0.0188
8	50	1.0000	1.0000	0.0099
9	100	1.0000	1.0000	0.0064
10	200	1.0000	1.0000	0.0049
11	300	1.0000	1.0000	0.0042
12	391	1.0000	1.0000	0.0006

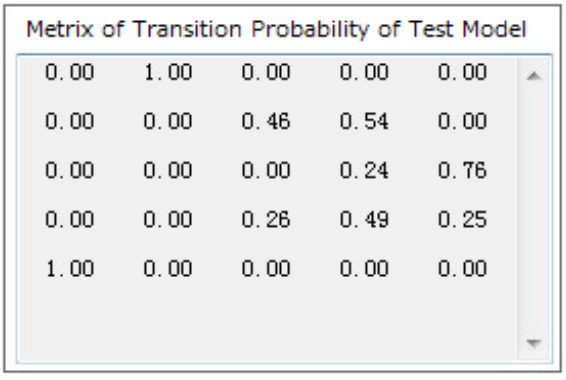
The first two rows of Table II show that the value of *K (U, T)* also can be calculated with the use of improved Kullback discriminant, even under the condition that the test cases failed to cover all the migration edges in usage model. This result coincides well with the theory proposed in section III and also indicates that the calculation method of *K(U,T)* with the improved Kullback discriminant is approving.

Statistical data in Table II shows that the convergence value *K(U, T)* from the test chain to the usage chain decreases overall as the number of test cases increases, which proves the fact that the test adequacy enhances gradually with the increase of test cases in software reliability test.

Fig. 3 shows the two matrixes of transition probability of the test chain and the usage chain when the test adequacy is reached. We can obviously discover that the difference between the two matrixes is extremely small, that is to say, the test chain has been very close to the usage chain when the testing process ends.



(a)



(b)

Figure 3. Matrixes of transition probability of the usage chain (a) and the test chain (b) when testing is sufficient.

B. Efficiency Verification of the Test Case Generation Method

In order to test and verify the effectiveness of the test case generation method presented in this paper, an external button is added to our test case generator *MTCG* to generate test cases in a completely random way. Comparison between the statistical results of test cases generated by Markov usage model and those in a completely random way are made, the results of which indicates that the generation method is efficient.

Table III shows the statistical results of test cases generated in a completely random way for a comparison, including the number of test cases, node coverage, edge coverage and the discriminant value *K(U,T)*. Here ε is set to 0.00001 and the threshold of *K(U,T)* is set to 0.001.

TABLE III.
STATISTICAL RESULTS OF TEST CASES GENERATED IN A COMPLETELY RANDOM WAY

No.	Number of Cases	Node Coverage	Edge Coverage	K(U,T)
1	1	0.8000	0.5000	0.8055
2	2	1.0000	0.8750	0.6833
3	3	1.0000	0.8750	0.6258
4	4	1.0000	0.8750	0.6307
5	5	1.0000	1.0000	0.5690
6	10	1.0000	1.0000	0.3739
7	20	1.0000	1.0000	0.3048
8	50	1.0000	1.0000	0.2832
9	100	1.0000	1.0000	0.0859
10	200	1.0000	1.0000	0.0968
11	400	1.0000	1.0000	0.0754
12	800	1.0000	1.0000	0.0413
13	1000	1.0000	1.0000	0.0295
14	2000	1.0000	1.0000	0.0012
15	2169	1.0000	1.0000	0.0009

Based on the statistical data in Table III, for the same software usage model in Fig. 1, 2169 test cases are generated until the testing process is sufficient in a completely random way. That is to say, in a completely random testing, at least 2169 test cases have to be executed in order to correspond to the actual usage of software system. However, based on the statistical data in Table II, only 391 test cases are generated by Markov usage model when the test adequacy is reached. That is to say, in a Markov usage model-based testing, much fewer test cases (18% in our experiments) need to be executed to correspond to the actual usage of software system. Integrated the comparison and analysis above, we can conclude that the test case generation method based on Markov usage model proposed in this paper is of high efficiency (See Fig. 4).

In addition, we can discover that the testing methods based on Markov usage model can cover all the transitions faster than a completely random testing method if the edge coverage is compared in Table II and Table III. That is to say, software testing based on Markov usage model has stronger ability of path coverage than completely random testing.

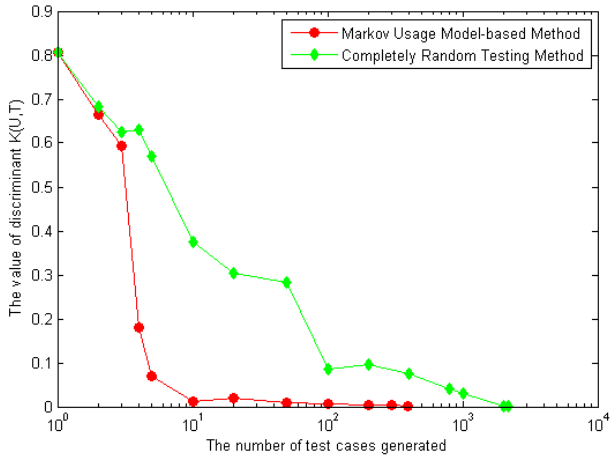


Figure 4. Different convergence status of discriminant between the Markov usage model-based and completely random testing methods.

C. Characteristic Analysis of Testing Method

Furthermore, we can find that the convergence value *K(U, T)* from the test chain to the usage chain increases abnormally while the test cases increasing from ten to twenty according to the sixth and seventh rows in Table II. This is a normal phenomenon which is caused by the instability of the test chain. In fact, this instability characteristic is a localized phenomenon, and will not affect the test adequacy determination.

This phenomenon also can be found in Table III. The convergence value *K(U,T)* increases abnormally twice according to the third, fourth, ninth and tenth rows in Table III. This proves that random testing is much more instable than the testing method proposed in this paper which is based on Markov usage model.

V. Conclusions

Software reliability test based on Markov usage model is an efficient approach to test cases generation and software reliability evaluation in a quantitative way. In this paper, we propose a modeling method for Markov usage model based on improved state transition matrix (STM), which also contributes to usage model checking. A method for test case generation based on Markov usage model is put forward in detail, and the test adequacy determination method is furtherly presented. We select an improved Kullback discriminant as the convergence judgment criteria from the test chain to the usage chain in order to measure whether the test adequacy has been reached. An experiment with an independently developed test case generation tool *MTCG* is carried out and the statistical results show that software reliability test based on Markov usage model can generate much fewer test cases (18% in our experiment) than the completely random testing method in the premise of ensuring test adequacy. That is to say, Markov usage model-based software reliability test presented in this paper is approving, high-efficient and promising.

References

[1] Prowell SJ. JUMBL: A tool for model-based statistical testing // Proceedings of the 36th Annual Hawaii International Conference on System Sciences [C].IEEE Computer Society Press, 2003.

[2] Sayre K, Poore JH. Stopping Criteria for Statistical Testing [J]. Information and Software Technology, 2000, 12(42): 851-857.

[3] Xi Hongyu, Xu Hong, Gao Zhongyi. Ada Software Test Case Generation Tool [J]. Journal of Software, 1997,8(04): 297-302.

[4] Zhang Yufen, Zhang Qunfeng, Yu Ruihua. Markov Property of Markov Chains and Its Test [C]. Proceedings of the Ninth International Conference on Machine Learning and Cybernetics, July 2010, 11-14.

[5] Whittaker JA, Thomason MG. A Markov Chain Model for Statistical Software Testing[J]. IEEE Transactions on Software Engineering, 1994, 20(10):812-824.

[6] Yan Jiong, Wang Ji, Chen Huowang. Deriving Software Markov Chain Usage Model from UML Models[J]. Journal of Software, 2005, 16(8):1386−1394(in Chinese).

[7] Gao Haichang, Feng Boqin, Zeng Ming, HeXiaohong. Statistical Software Test Based on Markov Chain Path Usage Model [J]. Computer Engineering, 2006, 32(19): 20-22(in Chinese).

[8] Murali Krishna Kadiyala, Dipti Shikha, Ravi Pendse et al. Semi-Markov Process based Model for Performance Analysis of Wireless LANs [C]. Seventh IEEE PerCom Workshop on Pervasive Wireless Networking, March 2011,613-618.

[9] Yiming He, Yingxia Pu, Jiechen Wang et al. Spatial-temporal dynamics of Sichuan industrial structure with Markov chains approach [C]. 18th International Conference on Geoinformatics, June 2010, 1-6.

[10] Liu YS, Zhang T, Zhang CF, Zha YB. A fault-tolerant scheduling algorithm for heterogeneous distributed real-time simulation systems [J]. Journal of Software, 2006, 17(10):2040−2047(in Chinese).

[11] Li N, Li MS, Wang Q, Zhao C, Du SZ. Adaptive Agent Negotiation for Software Process Modeling [J]. Journal of Software, 2009, 20(3): 557−566(in Chinese).

[12] Mo YC, Yang XZ, Cui G, Liu HW. Mission Reliability Analysis of Generalized Phased Mission Systems [J]. Journal of Software, 2007, 18(4):1068−1076(in Chinese).

[13] Liu Y, Ma ZY, He X, Shao WZ. Approach to transforming UML model to reliability analysis model [J]. Journal of Software, 2010, 21(2): 287−304(in Chinese).

[14] Kong DG, Tan XB, Xi HS, Shuai JM, Gong T. Hidden Markov model for multi-thread programs time sequence analysis [J]. Journal of Software, 2010, 21(3): 461−472 (in Chinese).

[15] Zhang DP, Nie CH, Xu BW. Importance Sampling Method of Software Reliability Estimation [J]. Journal of Software, 2009, 20(10): 2859−2866(in Chinese).

[16] Xu Libo, Wu Guoxin. Analysis on Congestion Control Strategy Based on Time Series Deduction [J]. Chinese Journal of Computers, 2007, 30(9): 1638-1644(in Chinese).

[17] Brian L. Mark and Yariv Ephraim. On Modeling Network Congestion Using Continuous-time Bivariate Markov Chains [C]. 45th Annual Conference on Information Sciences and Systems March, March 2011, 1-6.

[18] Zhang Youzhi. Research and application of hidden Markov model in data mining [C]. Second IITA International Conference on Geoscience and Remote Sensing (IITA-GRS), Aug. 2010, 459-462.

[19] Tian Xinguang, Gao Lizhi, Sun Chunlai3, Zhang Eryan. Anomaly Detection of Program Behaviors Based on System Calls and Homogeneous Markov Chain Model [J].

Journal of Computer Research and Developmen, 2007, 44(9): 1538-1544.
[20] Zhang DP, Nie CH, Xu BW. Cross-Entropy Method based on Markov Decision Process for Optimal Software Testing [J]. Journal of Software, 2008, 19(10): 2770-2779 (in Chinese).
[21] Wang Zi-kun, Yang Xin-gu. Birth-and-death process and Markov chains. Beijing, Science press, 2005.
[22] Wang HJ, Li ZS, Cheng Y, Zhou P, Zhou W. A latent variable model for cluster ensemble [J]. Journal of Software, 2009, 20(4): 825−833(in Chinese).
[23] M. Watanabe, Extended Hierarchy State Transition Matrix Design Method -Version 2.0 [J], CATS Technical Report, 1998.
[24] M. Matsumoto, K. Anada, D. Ueshima, M. Watanabe, and A. Fukuda, Model Checking of State Transition Matrix[C], In ITSSV 2005, AIST Technical Report AIST-PS-2005-017, pp. 2-11, 2005.
[25] A. Armando, J. Mantovani and L. Platania, Bounded Model Checking of Software using SMT Solvers instead of SAT Solvers, In SPIN 2006, LNCS 3925, pp. 146-162, 2006.
[26] L. Moura and N. Bjørner, Z3: An Efficient SMT Solver [C], In TACAS 2008, LNCS 4963, pp. 337-340, Springer, 2008.
[27] Weiqiang Kong, Tomohiro Shiraishi, Yuki Mizushima, et al. An SMT Approach to Bounded Model Checking of Design in State Transition Matrix [C]. Proceeding ICCSA 2010, IEEE Computer Society Washington, DC, USA, 2010, pp. 231-238.
[28] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs [C], in TACAS, 2004, pp. 168–176.
[29] L. Xu, SMT-based bounded model checking for realtime systems [C], in QSIC, 2008, pp. 120–125.

Kuanjiu Zhou received B.S. degree in 1989 and M.S. degree in 1993 from Harbin Institute of Technology in computer science, Harbin, China, and the Ph.D degree in 1996 from Harbin Institute of Technology in management science and engineering. During his M.S. work, his research interests include computer science, software testing. During his Ph.D work, he specialized in the complexity theory and system engineering. Since 1996, he has been a software engineer, working for Huawei Co. in Shenzhen, and Hisense Group in Qingdao. At Huawei, he leaded a team that developed INSM in an intelligent network. At Hisense, he researched the CDMA (Code Division Multiple Access) mobile. His current research interests include software testing and reliability theory.

Xiaolong Wang received B.S. degree in software engineering in 2011 from Dalian University of Technology, Dalian, China, is a M.S. candidate in computer science at Dalian University of Technology, Dalian, China, since 2011. His current research interests include software engineering, software testing and reliability theory.

Gang Hou received B.S. degree in 2005 from Dalian University of Technology in computer science and engineering, Dalian, China, and M.S. degree in 2008 from Dalian University of Technology in software engineering. His current research interests include software engineering, software testing and reliability theory.

Jie Wang received B.S. degree in 2002 and M.S. degree in 2004 from Harbin Institute of Technology in computer science, Harbin, China, and the Ph.D degree in 2009 from Harbin Institute of Technology in computer architecture. His current research interests include FPGA, software testing and reliability theory.

Shanbin Ai received B.S. degree in software engineering in 2011 from Dalian University of Technology, is a M.S. candidate in computer science at Dalian University of Technology, Dalian, China, since 2011. His current research interests include software engineering, software testing and reliability theory.