Markov Chains for Testing Redundant Software

Allan L. White; NASA Langley; Hampton
Jon A. Sjogren; USAAV RA; Hampton

Key Words: Software reliability, Multiple-version programming, Markov chains, Statistical confidence level.

Abstract

A preliminary design for a validation experiment has been developed that addresses several problems that are unique to assuring the extremely high quality of multiple version programs in process control software. If successful, the experimental procedure tests the programs in a realistic manner; it makes a minimum number of assumptions; and it is more efficient than natural life testing. The procedure uses Markov chains to model the error states of the multiple version programs. The programs are observed during simulated process control testing, and estimates are obtained for the transition probabilities between the states of the Markov chain. The experimental Markov chain model is then expanded into a reliability model that takes into account the inertia of the system being controlled. The reliability of the multiple version software is computed from this reliability model at a given confidence level by using confidence intervals obtained for the transition probabilities during the experiment. The major disadvantage of this approach is that the estimates for the transition probabilities must be inside a small region in order for this approach to be more efficient than natural life testing. The paper gives a general description of the procedure and then applies it to a simple example.

Introduction

A preliminary design for a validation experiment is developed that addresses seven problems that are unique to assuring the extremely high quality of multiple version programs in process control software (Migneault[6], Peterson[9]). First, the extremely high reliability requirement makes natural life testing impractical (Dunham[1]). Second, the similarity of program development casts doubt upon the assumption of independent failures for multiple versions (Knight&Leveson[4], Littlewood&Miller[5]). Third, there is need of a software failure model that uses directly observable parameters (Goel[3], Nagel&Skrivan[8]). Fourth, the common approach of testing by random input does not match the operating environment where the next input is a function of external forces and previous control reactions. Fifth, testing by random input does not take into account the effect of a region in the input space that causes trouble. Sixth, programs are commonly reset before each trial when testing by random input, but such testing does not capture the lingering effect of a previous fault. Seventh, software verification schemes neglect confidence bounds which indicate how much faith to have in the results (Miller[7]). In addition, the approach described below takes into account the inertia of the controlled system since it can take more than one failure of the control program to cause the controlled system to fail.

This work is not concerned with the reliability growth of a program during a debugging stage (Goel[2]). It is assumed that the programs are in their final form and will not be modified. The method below establishes their reliability as they will be used.

We use some terminology from control theory. The controlled system is called the plant; the controlling system with the programs is called the controller.

The material below assumes some basic

U.S. Government work not protected by U.S. Copyright.

1988 PROCEEDINGS Annual RELIABILITY AND MAINTAINABILITY Symposium
knowledge of finite Markov chains, statistical estimation, and confidence intervals (Trivedi[11], Rohatgi[10], Wilks[12]).

Sections two, three, and four cover the general material on experimental models and procedures, reliability models, and estimation and confidence intervals respectively. Section five presents a simple example. Section six contains the conclusions.

Attempting to address the seven items mentioned above is an ambitious undertaking, and it is not surprising that some problems were encountered. The most serious one is that the validation method accomplishes its goals only for programs that fall within a narrow range of failure behavior. This problem appears and is discussed in the example in section five.

Description of the Experimental Model

The experimental model is a finite state Markov chain where each state depicts the success-and-failure status of the multiple software for the current control cycle and where a transition probability between two states is the likelihood of jumping from one to the other state at the next cycle. The parameters to be estimated are these jump probabilities.

The experimental procedure to obtain the parameters for the experimental model is a modification of the plant operation simulations that test the effectiveness of control algorithms. The simulations to check the control algorithm use a realistic sequence of inputs that reflect the plant's operating environment. The response of the plant or controller (or both) is tracked to provide information about the properties of the controller. The multiple-version software experiment uses the same realistic sequence of inputs over the mission, but in this case the response being tracked is the correctness or incorrectness of the programs for each control cycle.

This tracking of the correctness-or-incorrectness of each version of the control program for every cycle is the observation of detail that might make it possible for this approach to be more efficient than natural life testing. However, a gain in efficiency is not guaranteed. Paradoxically, there is no gain for extremely good programs with no observed failures since in this case the Markov chain approach has the same amount of information as natural life testing—no observed failures.

The Markov chain model and the experimental procedure address the seven problems mentioned in the introduction. Under certain conditions, the detailed observations of program failures during each cycle permit this approach to be more efficient than natural life testing. The Markov chain model does not make any assumption about the independence of program failures. The model uses directly observable parameters, the jump probabilities. The next section discusses how the confidence level for the reliability of the multiple version package follows from the confidence bounds on the estimates for the jump probabilities. In this approach the inputs to the programs are functions of the operating environment and previous control decisions. The approach captures the effect of troublesome regions in the input space since the simulation forces the controlled system to follow a natural path through the input space. The lingering effect of a previous fault is also captured because the programs perform during the simulation as they would during a real mission.

The Reliability Model

The reliability model is developed from the experimental model by expanding the experimental model to include the specification of failure for the controlled plant. This expansion requires a knowledge of plant inertia which is assumed to come from control theory and is not covered in this paper. This knowledge of plant inertia tells us what failure patterns the controller can have [Band still keep the plant within the required performance envelope. The simplest type of tolerated failure pattern, and the one used in the example below, is that the controller retains acceptable control of the plant if the controller has no more than a certain number of consecutive failures.

Usually when a reliability model is obtained, it is merely a matter of using the experimental values to compute the probability of failure for the system. More work is required in the present case since the intent is to establish reliability at a given confidence level instead of just accepting a single number. The general procedure is as follows. First, a confidence level is chosen. Second, the experiment determines a
multivariable confidence interval for the parameters, the jump probabilities from state to state. Third, the reliability model is used to compute the probability of plant failure while varying the parameters over their confidence intervals. Fourth, the largest computed probability of failure is chosen as the reliability of the system at the given confidence level.

Pre-design efforts, working with the reliability model before any experiments are performed, can occasionally increase the efficiency of both the experiment and the subsequent calculations. One of the best things to discover is that the reliability is monotonic with respect to some of the parameters. If a parameter has this property then only a one-sided confidence bound is needed from the experiment, and only this single value is used when computing the reliability of the system at the given confidence level. The example in section 5 makes extensive use of the monotonicity of some of the parameters.

Number of Trials and Confidence Intervals

The purpose of the experiments is to establish that there is a low probability of failure for the multiple-version-software during a single mission. The software may be used for only one mission as on a space probe, or it may be used for many missions as on an airplane. Regardless of how the software is used after testing, the testing procedure is designed to establish its reliability during a single mission. To establish this reliability at a high confidence level, it is necessary to perform many test missions.

This section first considers the number of test missions necessary to establish the desired reliability by natural life testing assuming that the system performs extremely well. That is, there are no observed failures. The number of missions necessary for verification by natural life testing is used to derive the number of simulated missions allowed in the current approach under the provision that this approach has the goal of being more efficient than natural life testing. Next, the number of simulated missions allowed is used to determine how many trials are available to observe the parameters that are to be estimated, the jump probabilities. The number of trials are important for establishing confidence intervals. There is a derivation of obtaining the confidence interval for a single parameter by using Chebychev's formula. Finally, a theorem is presented on obtaining the multivariable confidence intervals from the single variable confidence intervals.

Suppose that natural life tests are performed to establish that the probability of failure is less than or equal to \( P(F) \) during one mission at a confidence level of \( 100(1-a)\% \). Assuming that the system is highly reliable and no system failure occurs during testing, then the number, \( N \), of testing missions required is given by the formula

\[
(1 - P(F))^N = a. \tag{1}
\]

That is, if the probability of system failure during a mission is bigger than or equal to \( P(F) \), then the probability of successfully completing \( N \) missions is less than or equal to \( a \). Hence there is a \( 100(1-a)\% \) chance that the experiment has not misled us, which is what is meant by a confidence level.

The number of simulated missions to be run is obtained from the number of natural-life testing-missions by dividing by two factors. One factor represents how much slower a simulated mission runs compared to a real life mission. The other factor represents the desired gain in efficiency (decreased running time) that is desired. The total number of process control cycles available for observing the jumps between states is the multiple of the number of simulated missions to be run times the number of process control cycles per mission.

A difficult item in the pre-experimental analysis arises from the fact that the total number of process control cycles for observing the jumps between states must be allocated among the different states in the experimental Markov chain model. This allocation is important because the number of trials for deriving the confidence bounds for the jumps out of a particular state depend on the number of visits to that state during the experiment, and these numbers are unknown before the experiment is performed. However, a reasonable value for the number of visits to each state can be derived firstly by assuming that the programs under consideration are fairly reliable and secondly by noticing that the estimates for the jump probabilities are ratios of
the number of visits to another state and the number of visits to the starting state for the jump. Suppose $K$ is the total number of cycles in the experiment, $p_{i,j}$ is the probability of jumping to state $j$ given the system is in state $i$, $x_{i,j}$ is the estimate for $p_{i,j}$, $m_i$ is the number of visits to state $i$, and state 1 is the state where all versions of the program give the correct answer. The assumption that the programs are fairly reliable implies that the number of visits to state 1, $m_1$, is nearly equal to $K$ the total number of cycles. This assumption also implies that most visits to state $j$ will be from state 1. Hence,

$$x_{i,j} = m_j/m_1 = m_j/K$$ (2)

or

$$m_j = Kx_{i,j}.$$ (3)

This approximation is used below in the pre-experimental analysis of confidence regions.

In the method developed below, the first step in deriving the confidence bounds is to consider each jump probability $p_{i,j}$ as the parameter in a Bernoulli trial, where a successful outcome is a jump from state $i$ to $j$, and an unsuccessful outcome is any other jump (including jump back to $i$). Confidence bounds for the estimate are obtained from Chebychev’s inequality. If $p$ is the true value of the binomial parameter, $x$ is its estimate, $n$ is the number of trials, and $100(1-a)\%$ is the desired confidence level, then the appropriate Chebychev inequality is

$$\text{Prob} \left( |x-p| < \frac{[p(1-p)]^{1/2}}{n^{1/2}} \right) > 1 - \frac{1}{\epsilon^2}. \quad (4)$$

where

$$\alpha = 1/\epsilon^2.$$ (5)

The solution for $p$ in terms of $x$, which is useful after the data is available, is

$$p = \frac{2x + \epsilon^2/n + \left[ 2x + \epsilon^2/n \right]^2 - 4[1+\epsilon^2/n]^{1/2}}{2[1+\epsilon^2/n]}.$$ (6)

The solution for $x$ in terms of $p$, which is useful for pre-experimental work, is

$$x = p \pm \frac{[p(1-p)]^{1/2}}{n^{1/2}}. \quad (7)$$

After the experiment has been performed, the number of trials $n$ is set equal to the actual number of visits to a state. Before the experiment has been performed, the number of trials $n$ is set equal to $Kx_{i,j}$ if the parameter under consideration corresponds to a jump out of state $j$.

The multivariable confidence intervals are obtained from the single variable confidence intervals with the following generalization of a standard theorem (Wilks[12]). This theorem does not require the random variables to be independent.

**Theorem:** Suppose $(a_1,b_1)$ is a $100(1-\alpha)_1\%$ confidence interval for $x_1$ for $1 \leq i \leq k$. Then $[(a_1,...,a_k),(b_1,...,b_k)]$ is a $100(1-\alpha,...-\alpha)_k\%$ simultaneous confidence interval for $(x_1,...,x_k)$.

**A Simple Example**

The general ideas will be illustrated by an example consisting of three programs performing the same cyclic process-control application. Majority voting is performed at the end of each cycle. Since we are not considering any hardware failures, the controller produces the correct output if at least two of the programs are correct. Mission time for the plant (the system being controlled) is one hour. The application control cycle lasts one-tenth of a second. Hence, there are 36,000 control cycles in a single mission. Because of the physical inertia of the plant, it takes four consecutive incorrect outputs from the controller to cause plant failure. The reliability requirement for the multiple version application software is that it causes plant failure during a one-hour mission with a probability less than $10^{-5}$. The requirement for the validation procedure is that it demonstrate this reliability at the 99% confidence level.

Solving the formula $(1-P(F))^n = \alpha$ for $P(F)=10^{-5}$ and $a=0.01$ gives $N=500,000$ for the number of natural life tests required. The total number of program cycles in this many natural tests is $500,000*36,000 = 18*10^8$. We will obtain the confidence region assuming an order of magnitude fewer cycles, $18*10^6$. The final gain in efficiency depends on the speed and cost of
The reliability Markov chain model is shown in figure 2. The controller passes an incorrect output to the plant at the end of a cycle if two or three of the program versions are incorrect, which is state 3 in the experimental model. The criterion for plant failure is four consecutive failures of the controller. The reliability model captures this condition by four replications of state 3 of the experimental model. Most of the transitions remain the same in the two models. The major change is that the \( P_{3,3} \) transition moves from state to state in the reliability model in order to account for the inertial memory of the plant.

The pre-experimental analysis begins by choosing a reasonable range of values for the jump probabilities. Since the sum of the jump probabilities leaving a state must sum to 1, there are actually six variables in figure 1, two for each state. We concentrate on the transitions that are detrimental to reliability in the hope that the analysis will then need to only consider upper confidence bounds instead of entire confidence intervals. Obvious choices are \( P_{1,2}, P_{1,3}, P_{2,2}, \) and \( P_{3,3} \) since these transitions move toward or stay in a failure state. Plant failure is likely to monotonically increase as these four variables increase. Less obvious choices are \( P_{2,2} \) and \( P_{2,3} \). Plant failure may or may not be monotonic in these two variables.

The range of values examined for the variables is

\[
10^{-8} \leq P_{i,j} \leq 10^{-3}
\]

This range appears reasonable for thoroughly tested, but not perfect, software. Extensive computation has shown that the most significant variable is \( P_{3,3} \), followed by \( P_{1,2} \). The variables \( P_{1,3} \) and \( P_{2,2} \) are less significant. The variables \( P_{2,2} \) and \( P_{2,3} \) are much less significant. These numerical results agree with figure 2 where a direct path to failure is \( P_{1,1} \), followed by three occurrences of \( P_{3,3} \). The probability of failure increases as \( P_{1,3}, P_{1,3}, P_{2,2}, \) and \( P_{3,3} \) increase. It is not always monotonic in \( P_{1,2} \) and \( P_{3,3} \).

The experimental Markov chain model is shown in figure 1. In state 1, all three programs have given a correct answer for the current control cycle. In state 2, exactly one program has given an incorrect answer. In state 3, two or three programs have given an incorrect answer. As mentioned before, the parameters to be estimated are the jump probabilities between the states, the \( P_{i,j} \)'s in figure 1. It is possible to have a more detailed experimental model, for example a model that tracks each program. However, a more detailed model has more parameters to estimate, and more parameters make it harder to obtain tight multivariable-confidence-bounds.

The simulation compared to natural life testing.

The experimental Markov chain model is shown in figure 1. In state 1, all three programs have given a correct answer for the current control cycle. In state 2, exactly one program has given an incorrect answer. In state 3, two or three programs have given an incorrect answer. As mentioned before, the parameters to be estimated are the jump probabilities between the states, the \( P_{i,j} \)'s in figure 1. It is possible to have a more detailed experimental model, for example a model that tracks each program. However, a more detailed model has more parameters to estimate, and more parameters make it harder to obtain tight multivariable-confidence-bounds.

The reliability Markov chain model is shown in figure 2. The controller passes an incorrect output to the plant at the end of a cycle if two or three of the program versions are incorrect, which is state 3 in the experimental model. The criterion for plant failure is four consecutive failures of the controller. The reliability model captures this condition by four replications of state 3 of the experimental model. Most of the transitions remain the same in the two models. The major change is that the \( P_{3,3} \) transition moves from state to state in the reliability model in order to account for the inertial memory of the plant.

The pre-experimental analysis begins by choosing a reasonable range of values for the jump probabilities. Since the sum of the jump probabilities leaving a state must sum to 1, there are actually six variables in figure 1, two for each state. We concentrate on the transitions that are detrimental to reliability in the hope that the analysis will then need to only consider upper confidence bounds instead of entire confidence intervals. Obvious choices are \( P_{1,2}, P_{1,3}, P_{2,2}, \) and \( P_{3,3} \) since these transitions move toward or stay in a failure state. Plant failure is likely to monotonically increase as these four variables increase. Less obvious choices are \( P_{2,2} \) and \( P_{2,3} \). Plant failure may or may not be monotonic in these two variables.

The range of values examined for the variables is

\[
10^{-8} \leq P_{i,j} \leq 10^{-3}
\]

This range appears reasonable for thoroughly tested, but not perfect, software. Extensive computation has shown that the most significant variable is \( P_{3,3} \), followed by \( P_{1,2} \). The variables \( P_{1,3} \) and \( P_{2,2} \) are less significant. The variables \( P_{2,2} \) and \( P_{2,3} \) are much less significant. These numerical results agree with figure 2 where a direct path to failure is \( P_{1,1} \), followed by three occurrences of \( P_{3,3} \). The probability of failure increases as \( P_{1,3}, P_{1,3}, P_{2,2}, \) and \( P_{3,3} \) increase. It is not always monotonic in \( P_{1,2} \) and \( P_{3,3} \).

The reliability Markov chain model is shown in figure 2. The controller passes an incorrect output to the plant at the end of a cycle if two or three of the program versions are incorrect, which is state 3 in the experimental model. The criterion for plant failure is four consecutive failures of the controller. The reliability model captures this condition by four replications of state 3 of the experimental model. Most of the transitions remain the same in the two models. The major change is that the \( P_{3,3} \) transition moves from state to state in the reliability model in order to account for the inertial memory of the plant.

The pre-experimental analysis begins by choosing a reasonable range of values for the jump probabilities. Since the sum of the jump probabilities leaving a state must sum to 1, there are actually six variables in figure 1, two for each state. We concentrate on the transitions that are detrimental to reliability in the hope that the analysis will then need to only consider upper confidence bounds instead of entire confidence intervals. Obvious choices are \( P_{1,2}, P_{1,3}, P_{2,2}, \) and \( P_{3,3} \) since these transitions move toward or stay in a failure state. Plant failure is likely to monotonically increase as these four variables increase. Less obvious choices are \( P_{2,2} \) and \( P_{2,3} \). Plant failure may or may not be monotonic in these two variables.

The range of values examined for the variables is

\[
10^{-8} \leq P_{i,j} \leq 10^{-3}
\]

This range appears reasonable for thoroughly tested, but not perfect, software. Extensive computation has shown that the most significant variable is \( P_{3,3} \), followed by \( P_{1,2} \). The variables \( P_{1,3} \) and \( P_{2,2} \) are less significant. The variables \( P_{2,2} \) and \( P_{2,3} \) are much less significant. These numerical results agree with figure 2 where a direct path to failure is \( P_{1,1} \), followed by three occurrences of \( P_{3,3} \). The probability of failure increases as \( P_{1,3}, P_{1,3}, P_{2,2}, \) and \( P_{3,3} \) increase. It is not always monotonic in \( P_{1,2} \) and \( P_{3,3} \).
Figures 3 and 4 display the reliability envelope (10^-5 probability of failure) for the two important variables p1,3 and p3,3. In figure 3, the least significant variables p1,2, p2,3, and p3,2 are set equal to 0.001, while p1,2 takes on the values 10^-4, 10^-6, 10^-8, and 10^-3. The curves in figure 5 show the solutions for p1,3 and p3,3 that give a probability of failure of 10^-5. Figure 4 is the same except that the four least significant variables are set equal to 0.1.

The next item to consider is the pre-experimental determination of an inner confidence region. This inner region is computed from the reliability envelope, and it has the property that if the estimates (from the experiment) for the parameters lie within the inner confidence region then (at the given confidence level) the true values for the parameters lie within the reliability envelope. The overall level desired in this example is 99% confidence that the probability of failure is less than or equal to 10^-5. The 1% margin of error can be distributed over the parameters as described by the quoted theorem in section 4. The distribution used below is

P1,3, P1,2, P3,3 at 99.7%
P2,2, P2,3, P3,2 at 99.97%.

The reason for this nonsymmetric distribution is to get tighter bounds on the more important parameters by requesting a lower confidence level.

The solid line is the [P3,3(1-P3,3)]1/2 and the dotted line is the [P1,3(1-P1,3)]1/2. The parameter K=18*10^8 is the total number of visits to the states (that is, the total number of cycles) during the experiment and -18.2574 is the value for the Chebychev inequality that gives a confidence level of 99.7%.

There is a lower boundary for the confidence region because as x1,3 decreases, the number of visits to state 3 becomes too small to support a tight confidence interval for the estimate x3,3 of p3,3. When this happens, the value for x1,3 must be very small in order to say that p3,3 is small at a high confidence level.

Running the pre-experiment design analysis was rather computationally intensive, due mainly to the calculation of reliability at 36000 iterations. About 10^9 floating point operations were performed.
Conclusions

A procedure for testing multiple-version process-control software has been presented that preserves the advantages of natural life testing while trying to be more efficient. The advantages of natural life testing are preserved by observing the software during simulations of the process-control mission. This method yields a realistic sequence of inputs. Observing all the multiple versions at once, instead of separately, eliminates the need to assume independence of failure in order to combine the separate observations into one model. The gain in efficiency arises from close observation of the error states of the programs combined with a knowledge of the inertia of the controlled system (the plant). The error states of the programs are modeled as states in a Markov chain, and the parameters observed are the transition probabilities between the states. Knowledge of the inertia of the controlled system is used to expand the experimental Markov model into the reliability Markov model.

The major problem with the method is that it appears to improve upon natural life testing only if the parameters fall within a narrow range. This problem is magnified by the need for an extremely high multi-variable confidence interval.

Acknowledgements

The authors are indebted to G.E. Migneault at NASA Langley Research Center for several informative conversations, and to Sandra Howell of Old Dominion University for assistance with the graphics.

References

Biographies

Allan L. White
NASA Langley Research Center
Hampton, VA 23665-5225

Allan L. White received a Ph. D. in mathematics from New Mexico State University. He has taught at the University of Colorado. He was an applied mathematician at Kentron International Inc. He is now at NASA Langley Research Center. His interests include the design and analysis of reliable process control systems.

Jon A. Sjogren
USAAVRADA
Hampton, VA 23665-5225

Jon A. Sjogren holds a doctorate in mathematics from the University of California (Berkeley). He held post-doctoral positions at the University of Bonn (Germany) and IHES (France). After teaching in the U.S. and abroad, he joined the Fault Tolerant Computing Group directed by K. S. Trivedi at Duke University. He is currently a Research Electronics Engineer with the Joint Research Programs Office, U. S. Army Aviation Systems Command attached to the NASA Langley Research Center.