IMPROVED TECHNIQUES FOR SOFTWARE TESTING BASED ON MARKOV CHAIN USAGE MODELS

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Kirk Sayre

December 1999

Copyright © 1999 by Kirk Sayre

All rights reserved

Acknowledgments

I would like to thank Dr. Jesse Poore for all of the hard work and good advice he has given me during the creation of this dissertation. I would also like to thank Dr. Ramon Leon, Dr. Bruce MacLennan, and Dr. Michael Vose for their helpful suggestions and careful review of this dissertation. Lastly I would like to thank the members of the SQRL lab for their assistance in my research.

Contents

List Of Tables

List of Figures

Chapter 1

Introduction

1.1 Statistical Testing Based on a Usage Model

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 [16]. Test cases are drawn from the sample population of possible uses according to the sample distribution and run against the software under test. Various statistics of interest, such as the estimated failure rate and mean time to failure of the software are computed. The testing performed is evaluated relative to the population of uses to determine whether or not to stop testing.

The model used to represent use of the software in this work is a finite state, time homogeneous, discrete parameter, irreducible Markov chain [10]. In a Markov chain usage model the states of use of the software are represented as states in the Markov chain [1, 18, 21, 22]. User actions are represented as state transitions in the Markov chain. The

probability of a user performing a certain action given that the software is in a particular state of use is represented by the associated transition probability in the Markov chain. The usage model always contains two special states, the *(Invoke)* state and *(Terminate)* state. The *(Invoke)* state represents the software prior to invocation and the *(Terminate)* state represents the software after execution has ceased. All test cases start from the *(Invoke)* state and end in the *(Terminate)* state.

Given a Markov chain based usage model it is possible to analytically compute a number of statistics useful for validation of the model, test planning, test monitoring, and evaluation of the software under test. Example statistics include the expected test case length and associated variance, the probability of a state or arc appearing in a test case, the long run probability of the software being in a certain state of use [22]. Test cases are randomly generated from the usage model, i.e., randomly sampled based on the use distribution. These test cases are run against the software and estimates of reliability are computed.

1.1.1 Suitability of Statistical Testing Based on a Usage Model

The suitability of usage model-based statistical testing of software, henceforth referred to simply as statistical testing, to real world software testing problems has been demonstrated by a number of successful applications of the testing methodology. Statistical testing has been used on a wide variety of software systems ranging from CASE tools to real-

time weapons command and control systems. In all cases benefits were realized from the application of statistical testing. Noteworthy applications include those listed in Table 1.

1.1.2 Necessity of Statistical Testing Based on a Usage Model

Real world applications of statistical testing show that software can be successfully tested using this methodology. The necessity of statistical testing may be shown by an analysis of the two primary goals of software testing, the estimation of the field reliability of the software and the discovery of software faults.

The field reliability, or reliability of the software from the user's point of view, depends greatly on the manner in which the software is used. For example, consider a system that performs a number of functions correctly and one function incorrectly. If the incorrect function is rarely used few field failures will be observed and the software will have high field reliability. However, if the incorrect function is frequently used, the software will exhibit a large number of field failures and have low field reliability. In general the reliability of software depends both on the faults contained in the software and on the manner in which the software is used. It is impossible to accurately estimate the field reliability of software if the environment of use is not taken into account when testing.

Not all software failures have an equal impact on the user. The impact of a failure on the user depends on the likelihood of encountering the failure and on the severity of the failure. For example, a failure that occurs in a function that is rarely used will have less impact on the perceived quality of the software than a failure that occurs in a function that

Table 1: Users of Statistical Testing

Organization	Application Type	Number of States
IBM SSD	Device Controllers	400 - 5,000
CTI	PET Scanners	~500
FAA	Air Traffic Monitoring System	~1,000
SET	CASE Tools	100-500
Navy	Missile Ballistic Control System	~200
Army	Mortar Ballistic Control System	~200
Air Force	Satellite Monitoring System	~500
Ericsson Telecommunications	Central Office Telephone Switch	~800
Raytheon	Embedded Real-time System	200 - 2,000

is frequently used. In this case the likelihood of encountering the failure has a significant affect on the importance of the failure to the user. As another example, consider an item of software for the control of a nuclear reactor that contains two faults, one that causes a failure on a data reporting screen and another that causes a failure when attempting to shut down the reactor core. From a user's viewpoint the failure that occurs when attempting to shut down the reactor core is more important than the failure that occurs on a data reporting screen.

Because some failures are more important to the user than others, it is necessary to focus testing so as to reveal the more important faults, i.e., test the software according to some model that highlights critical use. A software organization runs a number of risks if testing is not directed towards revealing the faults that have the greatest impact on the user.

- The testing performed by the organization may reveal many unimportant faults but miss some important faults during testing and release the software with serious defects. This will result in a low field reliability and run the risk of dissatisfying customers.
- The testing performed by the organization may reveal the important faults and a large number of relatively unimportant faults which cause the release of the software to be delayed. In this case the software organization might incur needless testing and development costs and possibly miss a market window.

If all failures are considered to be equally severe, the importance of a failure is completely dictated by the failure's probability of occurrence. In this case testing should be performed according to the expected use of the software. If failures are not of equal sever-

ity it may be necessary to use importance sampling to shift the probability distribution of the test cases to focus testing according to both the profile of use and the cost of failure [7].

1.1.3 State of Practice

Current statistical testing processes assume that enough testing will be performed to see long run statistical effects come into play. This may require a great deal of testing, therefore making some results primarily applicable to software organizations with extensive test automation. Many of the results presented herein are concerned with situations where not enough testing will be performed to justify an assumption of long run statistical effects.

The typical process followed when performing statistical testing is presented in Figure 1. An overview of new results will be provided in conjunction with discussion of the stage of the testing process impacted by the new results (indicated by shaded boxes).

1. **Get Specification** - Some form of a specification detailing the correct behavior of the software is needed to develop the usage model. The correct behavior of the software may be defined by a formal specification, requirements documentation, user's manual, or a predecessor system.

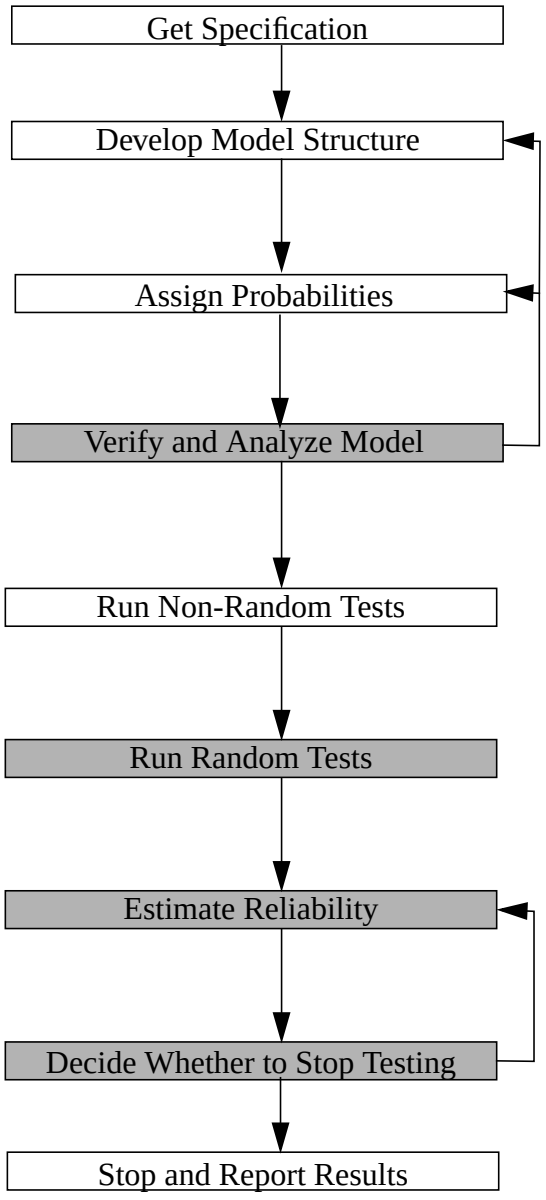


Figure 1: Testing Process

2. **Develop Model Structure** - The states of use and arcs connecting the states are identified. In current practice this stage is manual, i.e., the structure of the usage model cannot be derived automatically from a specification or other artifact. However, work is in progress to show how to derive the model structure from a sequence-based specification of the software.
3. **Assign Probabilities** - The state transition probabilities of the usage model are assigned. The probabilities may be assigned manually or they may be represented as a set of constraints and automatically calculated to satisfy some testing goal [17].
4. **Verify and Analyze Model** - Analytical results are calculated to aid in test planning and verifying that the model properly represents the expected use of the software [22]. The following new analytical results will be presented (chapter 2).
   - A means for calculating the expected number of test cases needed to cover a state or arc with associated variance will be presented. The variance of this statistic is a newly derived result.
   - A lower bound on the expected number of test cases needed to cover all states or arcs with associated variance will be presented.
   - The calculation for the probability of an arc appearing in a test case will be given. (This corrects an error in the literature [22].)
5. **Run Non-Random Tests** - Non-random test cases are crafted or generated from the usage model and then executed on the software under test. Examples of non-random tests currently being used are hand crafted tests, test cases generated in order of probability of occurrence, and test cases generated to cover all arcs in the usage model in the

minimum number of testing steps. Non-random tests may be run to satisfy contractual obligations, explore a particular use of the software, help validate the usage model and test facility, or determine whether the software is stable enough for full scale testing. Current practice does not include use of non-random tests in reliability estimation.

6. **Run Random Tests** - Test cases are generated randomly from the usage model and executed on the software under test. The test cases may be run automatically or by hand. Methods for partitioning the test cases, allocating testing effort to blocks in the partition so as to minimize reliability estimator variance, and generation of test cases from particular blocks in the partition are new results (chapter 4).
7. **Estimate Reliability** - The testing record containing information as to which tests were run and where failures were observed in the test cases is used to estimate the reliability of the software.

   Currently, the reliability is estimated using the Whittaker reliability model [22] or the binomial reliability estimator [15]. In the Whittaker model the testing record is stored as a testing chain. The testing chain is a Markov chain that contains all states from the usage model that have been visited during testing. The probabilities of the outgoing arcs of a state in the testing chain are calculated by normalizing the number of times each outgoing arc of a state has been visited during testing. Every time a unique error is encountered during testing a new state, a failure state, and arcs to and from the failure state are added to the testing chain. The reliability is computed as the probability of going from *(Invoke)* to *(Terminate)* in the testing chain without entering a failure state.

While this model has served well in many applications, field experience has revealed several limitations of the Whittaker model.

- The model does not make use of pretest information about the reliability of the software. For example, if a software organization is testing a new release of a product they may have a great deal of prior testing information from the previous releases indicating which portions of new release are likely to be reliable and which portions of the new release have not been proven to be reliable. Making use of this information in the reliability estimates of the current release will allow the organization to leverage their investment in the testing of the prior versions.
- The model behaves erratically during the early stages of testing. It assumes that enough testing will be performed to allow the reliability estimator to stabilize. This may require more testing than the organization can afford.
- The model provides no information in the absence of failures. Mission critical applications such as weapons control systems, power plant control software, etc. are unlikely to exhibit failures when subjected to final testing. Thus, the model may not be used to estimate the software's reliability in such situations.
- The model does not have an associated variance. Therefore it is impossible to define a confidence interval around the reliability predicted by the model.

New reliability estimators will be presented that allow prior reliability information to be used, behave in a predictable manner, and make more efficient use of the testing budget (chapters 3 and 4).

8. **Decide Whether to Stop Testing**- The testing record is evaluated to determine whether testing should continue or stop.

   Currently two methods are used to measure the degree to which testing matches the expected use of the software, the Euclidean distance between the usage chain and the testing chain and the Kullback discriminant of the usage chain to the testing chain.

   - The Euclidean distance can always be calculated. However, the Euclidean distance is not always an accurate measure of the similarity of the usage chain and the testing chain.
   - The Kullback discriminant provides a more accurate indication of the degree to which the testing chain matches the usage chain [2,20] but the Kullback discriminant is not defined until all arcs in the usage model have been visited at least once during testing. In many cases it may not be feasible to continue testing until all arcs in the usage model have been covered, constraining the usefulness of the Kullback discriminant as a stopping criterion. A method for approximating the Kullback discriminant prior to full arc coverage will be presented.
   - The Kullback discriminant provides no direct information about the stability of the testing record, i.e., about whether the testing chain has converged and is unlikely to change in the future. A specific definition of convergence and a means for assessing the convergence of the testing chain will be presented (chapter 5).

9. **Stop and Report Results** - After testing is finished the test results may be used for a number of purposes, such as deciding whether to release the product, evaluating

whether the software development process is under control, or evaluating the performance of a new piece of technology used in the product.

1.2 Notation

As work in the Software Quality Research Lab accumulates, material from the fields of probability theory, Markov chain theory, graph theory, information theory, sampling theory, abstract algebra, language theory and elsewhere is being used. We are seeing the notations of these fields collide. This dissertation represents a first step in an effort to unify notation. The “standard” notation of each field will be used where possible, and collisions will be resolved in the most reasonable way while avoiding overloading any uses. All notation is case and font specific.

Notation	Name	Comments
U=(*U*,**U**)	Markov Chain Usage Model	A model U is a pair defined by: (i) a directed graph *U*, (ii) a one-step transition (stochastic) matrix **U** of a discrete parameter, time homogeneous, aperiodic, irreducible Markov chain (iii) with a one-one correspondence between the nodes of the graph and states of the MC, edges of the graph and transitions of the MC
T, U, V, W		names of different usage models

Notation	Name	Comments
$U=(U,\underline{U})$	Directed graph of U	*U* is defined by: (i) an indexed state set *U* of state names, with indexes corresponding to row indexes of **U** (ii) an arc (edge) set $\underline{U}$ of arc names The short-hand name for $(U,\underline{U})$ is *U*, i.e., a graph is known by the name of its state set
u	number of states in *U*	u = |*U*| = rank of **U**
$\underline{u}$	number of arcs in $\underline{U}$	
U(*name*)	state name in *U*	corresponds to row of **U**
(name)	state name	abbreviated form when state set is understood
$\underline{U}(i,j)$	arc name in $\underline{U}$	*i* and *j* are state names, arc corresponds to element of **U**
(i,j)	arc name	abbreviated form when arc set is understood
Pr[event]	probability of an event	
$u_{i,j}$	row i, column j element of **U**	$Pr[X_{n+1}=j \mid X_n= i] = u_{i,j}$ in **U**
$\pi(\mathbf{U})$	long run state occupancies for **U**	given the properties of the markov chains this is also the stationary distribution
π	long run state occupancies	abbreviated form when model is understood
$\pi(\mathbf{U}, name)$	long run state occupancy for state *(name)* in **U**	long run occupancy of state *(name)*, may be named by index of **U**, for example $\pi(\mathbf{U}, Invoke)$ or $\pi(\mathbf{U}, 0)$ or $\pi(\mathbf{U}, 1)$ depending on the indexing base
π(*name*)	long run state occupancy for state *(name)*	abbreviated form when model is understood
$\pi(\underline{U})$	long run arc occupancies for **U**	

Notation	Name	Comments
$\underline{\pi}$	long run arc occupancies	abbreviated form when model is understood
$\pi(\underline{U},i,j)$	long run arc occupancy for arc *(i,j)* in model **U**	long run occupancy of arc *(i,j)*
$\pi(i,j)$	long run arc occupancy	abbreviated form when model is understood
X, Y, Z	random variables	not to be confused with a graph
E(*X*)	expectation of *X*	
$\hat{E}(X)$	estimate of the expectation of X	
Var(*X*)	variance of *X*	
$\hat{V}ar(X)$	estimate of the variance of *X*	
H(**U**)	source entropy of **U**	
H	source entropy	abbreviated form when model is understood
H(U(*name*))	entropy of state U(*name*)	
H(*name*)	entropy of state *(name)*	abbreviated form when model is understood
K(**U**,**V**)	Kullback discriminant	
d(**U**,**V**)	Euclidean distance	
<**S**,**F**,*n*>	Testing record	A testing record consists of success matrix **S**, where s_{ij} = the number of times arc *(i,j)* was taken successfully in testing, failure matrix **F**, where f_{ij} = the number of times a failure was observed on arc *(i,j)* during testing, and *n*, the number of tests run

Chapter 2

Quantitative Analysis of Models for Test Planning

2.1 Models From Industry

The usage models in Table 2 are taken from real industrial use, except for the NAS model, and will be used for examples throughout the dissertation. The models are listed by number of states in the following table and in the appendices. Even larger models are in use in industry, the largest approaching 5,000 states. It is not practical to present the full model or complete analytical information on each model, but it is desirable to establish a standard way of characterizing models.

For each model a data sheet with a comprehensive quantitative characterization of the model is given in Appendix A. Some of the statistics in the data sheets were developed in this research and will be discussed in section 2.4. This information, which can be used for

Table 2: Example Models

Model Name	Application	Number of States	Number of Arcs
NAS	Example	17	29
CDU_Inc2	Ballistic Command and Control, Navy	51	300
Certify_Inc3A	CASE Tool, SET	56	151
MSDS	Material Safety Information System, Lockheed Martin Energy Systems	130	565
Tucson2	Mass Storage Device Controller, IBM	392	573

test planning, is more extensive than heretofore used in industry. Each data sheet contains the following information about its model.

- **Number of States** - The state count remains the most basic fact about models because it determines the order of the transition matrix and the computation time for analyses that require inversion of a matrix of this order. The state count includes the distinguished states *(Invoke)* and *(Terminate)*.
- **Number of Arcs** - The number of arcs, $\underline{u}$, is also of fundamental importance and determines the density (or sparsity) of the transition matrix. The arc count includes arcs from *(Invoke)* and to *(Terminate)*, as well as the single arc *(Terminate, Invoke)*.
- **Density** - The density of the transition matrix **U** provides an indication of path complexity, i.e., a dense transition matrix indicates a large number of branching choices at each state, which in turn means that there could be a relatively diverse population of paths. The density is calculated as the proportion of non-zero probabilities in the transition matrix, $\underline{u} / u^2$.
- **Entropy** - The source entropy, H, is the probability weighted average of the state entropies (state entropy is the entropy of the probability vector over the exit arcs of the state) and is an information theoretic measure of the uncertainty in the model [2]. Source entropy is used in other calculations and for comparing models of the same structure but different probabilities.
- **Trajectories** - The number of statistically typical paths from *(Invoke)* to *(Terminate)* is calculated as $2^{H/(\pi(Invoke))}$ [2], where $\pi(Invoke)$ is the long run probability of *(Invoke)*. The exponent is the minimum average number of yes-no questions that

would be necessary to ask and answer in order to identify a randomly selected path, thus the estimate of the number of paths that it takes to account for all the probability mass.

- **Order-cube** - Some statistics computed for a usage model require inversion of a matrix of the order of the transition matrix. Inversion of a matrix by direct methods is an order u-cube calculation and an indication of cost of computation.
- **Expected Sequence Length** - The sequence length (the number of state transitions in a sequence) of a model is a random variable whose expectation represents the average sequence length measured in transitions in the long run.
- **Variance of Sequence Length** - Usage models may contain a high degree of variation, making the expectations of many random variables based on the usage model misleading for planning purposes unless associated variance is considered.
- **Simulation Cost in Sequences To Approximate Expected Sequence Length** - Average sequence length can be approximated by simulation. The number of sequences that would be necessary to generate and average for length, in order to have 95% confidence of being within 10% of the analytical expectation, is given.
- **Simulation Cost To Approximate Expected Sequence Length** - A data point is a single sequence. The numbers reported are the 95% confidence interval of the cost in transitions of simulating an adequate number of data points to give 95% confidence that the estimated sequence length is within 10% of the analytical value.
- **State Mean First Passage in Transitions** - The random variable is the number of transitions to reach the state in question, given that the trajectory began in *(Invoke)*.

For each state, the expectation is the mean time in transitions before the state will first be visited. The states having the least and greatest mean first passage times are identified by number and the expectations are given.

- **Variance of State Mean First Passage** - The variances of the above random variables are given.
- **Simulation Cost To Approximate State Mean First Passage in Transitions** - A data point is the number of transitions in a sequence from the recurrent chain that begins in *(Invoke)* and ends in the first occurrence of the state. The numbers reported are the 95% confidence interval of the cost in transitions of simulating an adequate number of data points to give 95% confidence that the estimated mean first passage is within 10% of the analytical value. This cost to estimate the mean first passage time in simulation is given for states having the least and greatest values.
- **State Mean First Passage in Sequences** - The random variable is the number of sequences to reach the state in question. For each state, the expectation is the mean number of sequences before the state will first be visited. The states having the least and greatest mean first passage times are identified by number and the expectations are given.
- **Variance of State Mean First Passage** - The variances of the above random variables are given.
- **Simulation Cost To Approximate State Mean First Passage in Sequences** - A data point is the number of sequences until first occurrence of the state. The numbers reported are the 95% confidence interval of the cost in transitions of simulating an ade-

quate number of data points to give 95% confidence that the estimated mean first passage is within 10% of the analytical value. This cost to estimate the mean first passage time in simulation is given for states having the least and greatest values.

- **Arc Mean First Passage in Transitions** - The random variable is the number of transitions to reach the arc in question, given that the sequence began in *(Invoke)*. For each arc, the expectation is the mean time in transitions before the arc will first be visited. The arcs having the least and greatest mean first passage times are identified by number and the expectations are given.
- **Variance of Arc Mean First Passage** - The variances of the above random variables are given.
- **Simulation Cost To Approximate Arc Mean First Passage in Transitions** - A data point is the number of transitions in a sequence from the recurrent chain that begins in *(Invoke)* and ends in the first occurrence of the arc. The numbers reported are the 95% confidence interval of the cost in transitions of simulating an adequate number of data points to give 95% confidence that the estimated mean first passage is within 10% of the analytical value. This cost to estimate the mean first passage time in simulation is given for arcs having the least and greatest values.
- **Arc Mean First Passage in Sequences** - The random variable is the number of sequences to reach the arc in question. For each arc, the expectation is the mean number of sequences before the arc will first be visited. The arcs having the least and greatest mean first passage times are identified by number and the expectations are given.

- **Variance of Arc Mean First Passage** - The variances of the above random variables are given.
- **Simulation Cost To Approximate Arc Mean First Passage in Sequences** - A data point is the number of sequences until first occurrence of the arc. The numbers reported are the 95% confidence interval of the cost in transitions of simulating an adequate number of data points to give 95% confidence that the estimated mean first passage is within 10% of the analytical value. This cost to estimate the mean first passage time in simulation is given for arcs having the least and greatest values.
- **State Long Run Probability** - The stationary distribution is calculated analytically, and gives the probability of being in each state in the long run. The data sheets report this value for *(Invoke)* and for the states with the least and greatest values.
- **Simulation Cost to Approximate State Long Run Probability** - The number of state transitions needed to approximate the long run probability of a state by simulation so as to have 95% confidence that the approximated value is within 10% of the analytical value is given.
- **Arc Long Run Probability** - The probability of each arc being traversed, or transition occurring, in the long run is given for the arcs with the least and greatest values.
- **Simulation Cost to Approximate Arc Long Run Probability** - The number of state transitions needed to approximate the long run probability of an arc by simulation so as to have 95% confidence that the approximated value is within 10% of the analytical value is given.

- **Estimate of the Number of Sequences to Achieve State Coverage** - The random variable is the number of sequences needed to cover all states. This is one of the most requested statistics for usage models, but analytical values are not available. The expected value and variance are estimated by simulation, and the cost of the simulation is also reported.
- **Estimate of the Number of Sequences to Achieve Arc Coverage** - The random variable is the number of sequences needed to cover all arcs. This is one of the most requested statistics for usage models, but analytical values are not available. The expected value and variance are estimated by simulation, and the cost of the simulation is also reported.

2.2 Derivation of Sampling Results for Simulations

The number of data points needed in simulation to approximate the expectation of a random variable known by analytical methods is of interest for two reasons. First, since some statistics of interest do not have analytical solutions (known by us) and must be approximated through simulation, it might be just as well to approximate those of known analytical values at the same time. Second, as the order of the transition matrix increases analytical solutions might become computationally prohibitive, forcing the use of approximations determined by simulation.

Given a random variable X, its expected value and variance, the number of data points needed to approximate $E(X)$ can be calculated for a confidence level c% and an error tolerance $(0 < \varepsilon < 1)$.

Theorem 2.1 For X a random variable and $0<\varepsilon<1$, an approximation in the closed interval $[(1-\varepsilon)E(X),(1+\varepsilon)E(X)]$ is given by $n = \frac{z^2 \cdot \mathrm{Var}(X)}{\varepsilon^2 \cdot (E(X))^2}$ data points with c% confidence, where z is the appropriate standard normal tail-end value for a c% confidence interval.

Proof: The size of the desired interval is $2\varepsilon E(X)$. The size of a c% confidence interval around the expected value of a normally distributed random variable is $2z \cdot \frac{\sqrt{\mathrm{Var}(X)}}{\sqrt{n}}$ [5]. The goal is for the two intervals to have equal size. Therefore, equate the two and solve for n. □

When approximating the value of the previously discussed Markov chain statistics through simulation, the cost of collecting a single data point (either a sequence or a transition) is the number of simulated state transitions required to collect that data point. Each data point will have a different collection cost, which means that the cost of collecting a fixed number n of data points will vary between groups of data points. Therefore, the cost of collecting data points can be viewed as a random variable X which is the sum of independent and identically distributed random variables $X_1,...,X_n$, where X_i is the cost of col-

lecting the *i*-th data point. Because all X_i are independent and identically distributed, $E(X)=n\,E(X_1)$ and $Var(X)=n\,Var(X_1)$. By the central limit theorem X has an approximately normal distribution and the standard confidence interval calculations apply [5].

2.3 Example Model

Figure 2 shows the graph of an example usage model that will be used throughout this dissertation. The model is called the NAS model because it is taken from [16]. The probabilities for the model are shown in Table 3.

The expected length of sequences generated from the NAS model is 56.3 state transitions with a variance of 1620. The variance indicates that the sequence length will vary significantly from sequence to sequence. The probability of state *(S3)* appearing in a sequence is 0.231, making *(S3)* the least likely state to appear in a sequence. *(S3)* will also require the largest number of test sequences, 4.33 on average, until its first appearance. In the long run the probability of being in *(S3)* is 0.00523, meaning over the long run about half of one percent of the time will be spent in that state. Other information may be read from Table 15 in the appendix.

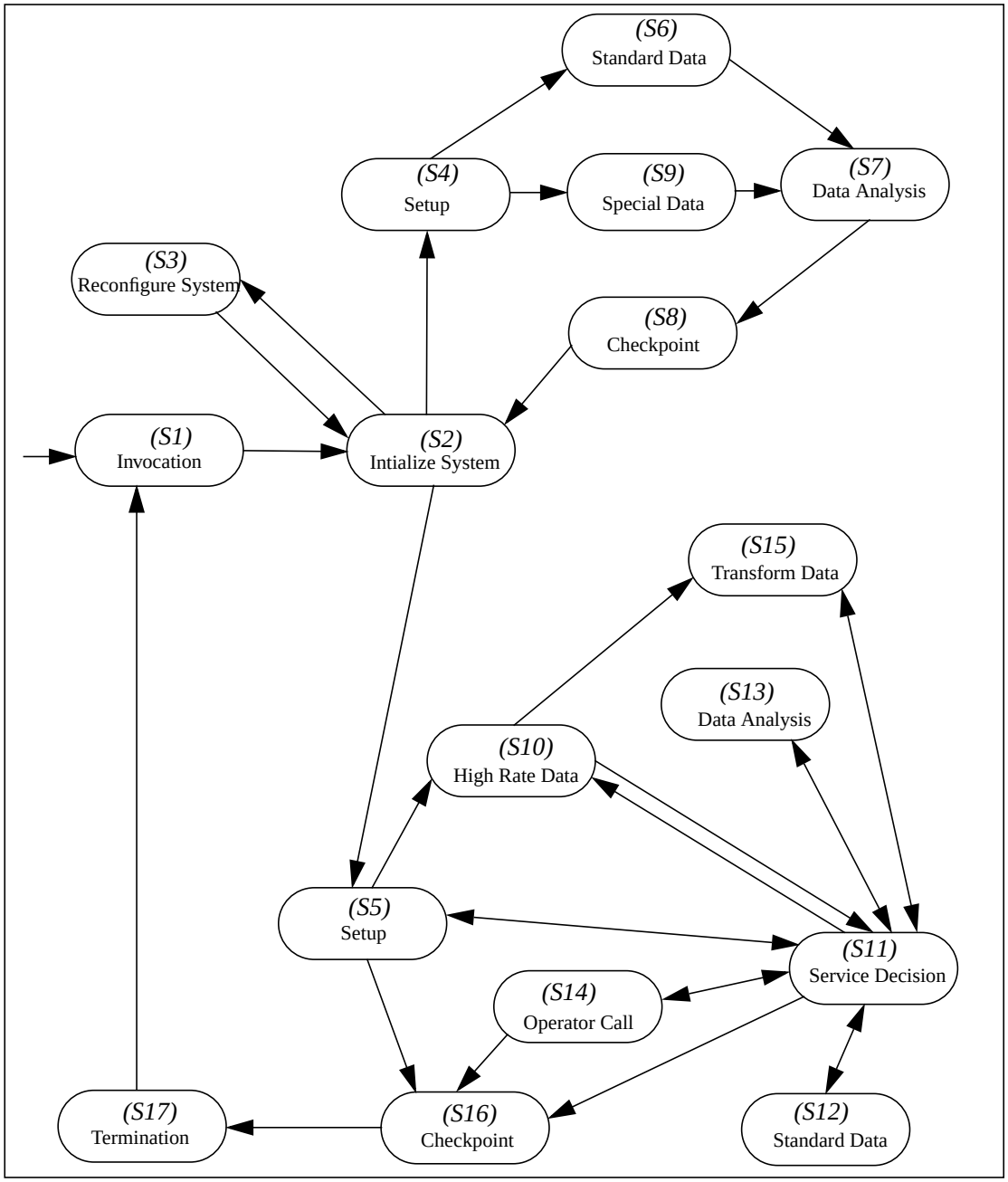


Figure 2: NAS Model

Table 3: NAS Model, Transition Probabilities

From State	To State	Transition Probability	From State	To State	Transition Probability
(S1)	*(S2)*	1.00	*(S11)*	*(S5)*	0.698
(S2)	*(S3)*	0.0698	*(S11)*	*(S10)*	0.0562
(S2)	*(S4)*	0.698	*(S11)*	*(S12)*	0.0508
(S2)	*(S5)*	0.233	*(S11)*	*(S13)*	0.0562
(S3)	*(S2)*	1.00	*(S11)*	*(S14)*	0.0375
(S4)	*(S6)*	0.833	*(S11)*	*(S15)*	0.0508
(S4)	*(S9)*	0.167	*(S11)*	*(S16)*	0.0508
(S5)	*(S10)*	0.500	*(S12)*	*(S11)*	1.00
(S5)	*(S11)*	0.500	*(S13)*	*(S11)*	1.00
(S6)	*(S7)*	1.00	*(S14)*	*(S11)*	0.500
(S7)	*(S8)*	1.00	*(S14)*	*(S16)*	0.500
(S8)	*(S2)*	1.00	*(S15)*	*(S11)*	1.00
(S9)	*(S7)*	1.00	*(S16)*	*(S17)*	1.00
(S10)	*(S11)*	0.500	*(S17)*	*(S1)*	1.00
(S10)	*(S15)*	0.500	*(S11)*	*(S5)*	0.698

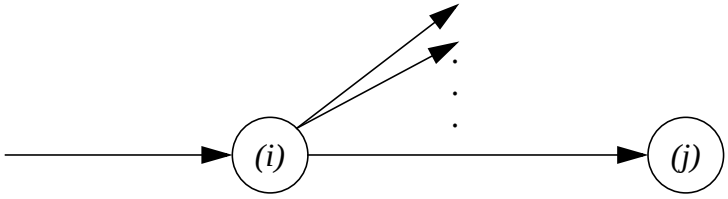
2.4 New Statistics for Test Planning

Several items described in section 2.1 are newly developed and are discussed here along with suggestions on how to use them in test planning.

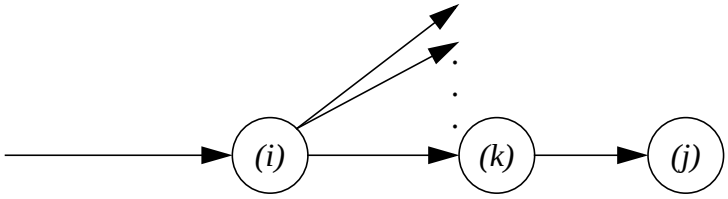
2.4.1 Probability of an Arc Appearing in a Sequence

The probability of a given arc appearing in a sequence may be calculated in the same manner as the probability of a given state appearing in a sequence by making some changes to the usage model. The probability of an arc *(i,j)* appearing in a sequence is computed as follows:

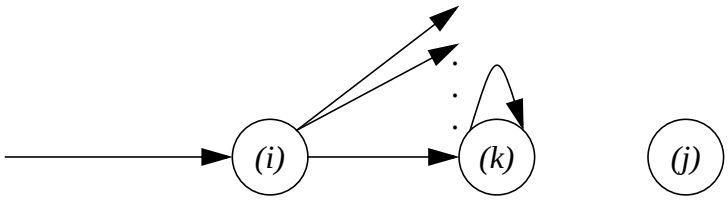
1. The usage model containing arc *(i,j)* is given.



2. The placeholder state *(k)* is added.



3. The placeholder state and *(Terminate)* are made absorbing.



4. The probability of absorption in the placeholder state is computed. This is the probability of arc *(i,j)* appearing in a randomly generated sequence.

Because the event of a placeholder state appearing in a sequence is analogous to the arc corresponding to the placeholder state appearing in a sequence, the probability of a placeholder state appearing in a sequence is equal to the probability of the respective arc appearing in a sequence.

The probabilities of the arcs in the NAS model appearing in a randomly generated sequence are reported in Table 4.

2.4.2 Expected Number of Sequences to Cover a State or Arc

One useful analytical result that may be computed from a usage model is the expected number of sequences generated until a state or arc's first appearance, known as the mean first passage of the state or arc [20]. This result aids in test planning when testing goals involve knowing whether certain states or arcs are covered during random testing.

The number of sequences generated until the first appearance of a specified state or arc can be viewed as a discrete random variable X that can take any integer value greater than

Table 4: NAS Model, Probability of Arc Appearing in Sequence

Arc	Probability	Arc	Probability	Arc	Probability
(S1, S2)	1.00	*(S7, S8)*	.750	*(S11, S15)*	.422
(S2, S3)	.230	*(S8, S2)*	.750	*(S11, S16)*	.731
(S2, S4)	.750	*(S9, S7)*	.333	*(S12, S11)*	.422
(S2, S5)	1.00	*(S10, S11)*	.808	*(S13, S11)*	.447
(S3, S2)	.230	*(S10, S15)*	.808	*(S14, S11)*	.212
(S4, S6)	.714	*(S11, S5)*	.909	*(S14, S16)*	.269
(S4, S9)	.333	*(S11, S10)*	.447	*(S15, S11)*	.838
(S5, S10)	.917	*(S11, S12)*	.422	*(S16, S17)*	1.00
(S5, S11)	.917	*(S11, S13)*	.447	*(S17, S17)*	1.00
(S6, S7)	.714	*(S11, S14)*	.425		

zero. The expected value of *X* has been routinely computed for field use. The variance of *X* has not been routinely provided. Because the variance of *X* can be large, a specified state or arc may be covered much later or much earlier in testing than the time given by E(*X)*, and Var(*X)* should be taken into account when using this statistic for test planning. The variance of the mean first passage of a state or arc may be computed by making use of the fact that this random variable has a geometric distribution.

2.4.2.A Geometric Distribution

The geometric distribution is a discrete distribution with a single parameter p where p is the probability of the occurrence of a particular event [5]. The geometric distribution is applicable to random variables representing the occurrence of a particular event for the first time at a discrete time $t > 0$. If the probability p of the event occurring is constant over time and independent of any previous trials, the probability of the event occurring for the first time at time t is $f(t|x) = (1-p)^{t-1}p$ for all $t \in \mathbf{N}$. The expected value of a random variable with a geometric distribution is $1/p$ and its variance is $(1-p)/p^2$.

2.4.2.B Expectation and Variance of State or Arc First Passage

A standard analytical result computed from a usage model is the probability of a state or arc appearing in a randomly generated sequence. These events are independent, i.e. the fact that a state or arc did or did not appear in a previous sequence has no affect on the

probability of the state or arc appearing in the current sequence. Viewing a trial as the random generation of a sequence and the event of interest as the occurrence of the specified state or arc, the random variable modeling the number of sequences generated until the first occurrence, has a geometric distribution. An approximate $100(1-\alpha)\%$ confidence interval for the expectation may be computed as $\mathrm{E}(X) \pm z_{\alpha/2}\sqrt{\mathrm{Var}(X)}$.

Example 2.1 Mean First Passage, With Variance

The expected mean first passage time, associated variance, and an approximate 95% confidence interval for the state mean first passage times are shown in Table 5 for the NAS model.

2.4.3 Number of Sequences Needed to Cover All States or Arcs

Often test planners wish to know how many test cases will be needed to cover all states or arcs in a usage model. We do not have an analytical solution for this statistic, but can approximate its value through simulation. Obviously, the expected mean first passage time of the state least likely to appear in a sequence, is a lower bound on the number of sequences needed to cover all states in the model. Likewise, the expected mean first passage of the arc least likely to appear in a sequence is a lower bound for the number of sequences needed to cover all arcs in the model. These lower bounds for covering all states and arcs in the NAS model are shown in Table 6, along with the associated variances and

Table 5: NAS Model, State Mean First Passage

State	Mean First Passage	Variance	95% CI Lower	95% CI Upper
(S1)	1.00	0	1.00	1.00
(S2)	1.00	0	1.00	1.00
(S3)	4.33	14.4	1.00	11.8
(S4)	1.33	0.444	1.00	2.64
(S5)	1.00	0	1.00	1.00
(S6)	1.40	0.560	1.00	2.87
(S7)	1.33	0.444	1.00	2.64
(S8)	1.33	0.444	1.00	2.64
(S9)	3.00	6.00	1.00	7.80
(S10)	1.08	0.0850	1.00	1.65
(S11)	1.00	0	1.00	1.00
(S12)	2.37	3.24	1.00	5.90
(S13)	2.24	2.77	1.00	5.50
(S14)	2.36	3.19	1.00	5.86
(S15)	1.19	0.230	1.00	2.13
(S16)	1.00	0	1.00	1.00
(S17)	1.00	0	1.00	1.00

Table 6: NAS Model, Lower Bound of Expected Number of Sequences Until State/ Arc Coverage

Coverage	Expected Value (Lower Bound)	Variance (Lower Bound)	95% CI, Lower (Lower Bound)	95% CI, Upper (Lower Bound)	Expected Value (Simulation)	Variance (Simulation)
State	4.3	14.4	1.0	11.8	5.98	12.7
Arc	4.7	17.5	1.0	12.9	8.02	18.1

approximate 95% confidence intervals. These may be compared with the approximation based on simulation.

2.5 Conclusions About Quantitative Analysis of Models for Test Planning

Quantitative results used for test planning may be computed analytically, which typically involves a matrix inversion, or they may be estimated through simulation. Given the ease in which distributed solutions may be applied to the problem of estimating the quantitative analysis results through simulation, for some organizations it may make sense to estimate the quantitative analysis results through simulation rather than computing the results analytically. There are two primary reasons an organization may decide to estimate these values rather than computing them analytically:

- If the time required to estimate *x*, where *x* is a quantitative analytical result listed earlier in the chapter, is *n*, it is fairly simple to devise a distributed algorithm to estimate *x* through simulation that will reduce the computation time by a factor of *c*, where *c* is the number of processors used in the distributed solution. Given enough processors, a distributed solution could be significantly faster than an analytical solution.
- The testing organization may not need a high degree of accuracy in the quantitative analytical results. If this is the case it may be faster to estimate the quantitative analyt-

ical results to a relatively low degree of accuracy than to compute the exact values analytically.

The variances of the random variables being assessed by the quantitative analytical results have not been consistently used in practice. Because the variances of these random variables can be quite large, the probability is high of observing an instance of the random variable during testing that is significantly different than the random variable's expected value. Because of this, it is recommended that the variances of the random variables associated with the quantitative analytical results be used more consistently.

Chapter 3

Arc-Based Reliability Models

The Miller reliability model [12] can be used in conjunction with usage models to define software reliability estimators. Two different reliability estimators, a single-use reliability estimator and a single-action reliability estimator, and their associated variance calculations will be presented. A simulation will be given to illustrate reliability estimation of a software system using the two different estimators.

3.1 The Miller Reliability Model

The Miller model is based on Bayesian statistics and allows the user of the model to take advantage of prior knowledge of the system under test [12]. The Miller model assumes that the possible failure rates of the software have a standard beta distribution [9]. In the Miller model the expected value of the reliability R of the system under test is $\mathrm{E}(R) = 1 - \left(\frac{f + a}{f + s + a + b}\right)$, where s is the number of successful tests run, f is the number

of failures, and *a* and *b* are parameters representing prior information about the software failure rate. For the case of no prior information about the software failure rate $a=b=1$.

The variance of *R* is

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

While the Miller model can be used to calculate the expected value of the overall reliability for the system, it can also be used when the testing domain is partitioned into equivalence classes (also called blocks or bins). Reliability can then be calculated for each block of the partition as well as for the entire system.

When partitioning the testing domain, each block *i* has a probability p_i representing the probability of drawing a test case from that block of the partition. Within the block all test cases must be equally likely; however, Miller presents a general method for satisfying this requirement. Each block *i* has associated priors a_i, b_i and random variables F_i and $R_i = 1 - F_i$, where F_i represents the failure rate of block *i* and R_i represents the reliability of block *i*. These random variables have expected values

$$\mathrm{E}(F_i) = \frac{f_i + a_i}{(f_i + s_i + a_i + b_i)}$$

and $\mathrm{E}(R_i) = 1 - \mathrm{E}(F_i)$, respectively. F_i and R_i have variance

$$\mathrm{Var}(F_i) = \mathrm{Var}(R_i) = \frac{(f_i + a_i)(s_i + b_i)}{(f_i + s_i + a_i + b_i)^2(f_i + s_i + a_i + b_i + 1)} .$$

The expected value of the software reliability when testing from a multi-block partition is $\mathrm{E}(R) = 1 - \sum_i p_i(\mathrm{E}(F_i))$.

The variance of the reliability estimator when testing from a multi-block partition was not given in [12], but is derived as follows.

Theorem 3.1 The variance of the reliability estimator when testing from a multi-block partition is $\mathrm{Var}(R) = \sum_i p_i^2 \mathrm{Var}(R_i)$.

Proof: The random variable representing the reliability of the software when testing from a multi-block partition, $R = 1 - \sum_i p_i F_i$, is the sum of a family of independent random variables of the form $p_i F_i$. Therefore

$$\mathrm{Var}(R) = \mathrm{Var}\left(1 - \sum_i p_i F_i\right) = \mathrm{Var}(1) + \mathrm{Var}\left(\sum_i -p_i F_i\right) = \sum_i {p_i}^2 \mathrm{Var}(F_i)$$
$$= \sum_i {p_i}^2 \mathrm{Var}(1 - R_i) = \sum_i p_i^2 \mathrm{Var}(R_i).$$

□

3.2 The Terminal Arc Partitioning Strategy

A partitioning of the test sample space requires that each and every test case belongs to one and only one block of the partition. The first step in partitioning the test space is to define what is meant by a test case. In the arc-based reliability models a test case will be defined as a sequence from *(Invoke)* to any other state in the usage model without returning to *(Invoke)*. This is a somewhat unusual definition of a test case, since the final state is not necessarily *(Terminate)*. An example of this test case definition will be illustrated with the small usage model in Figure 3.

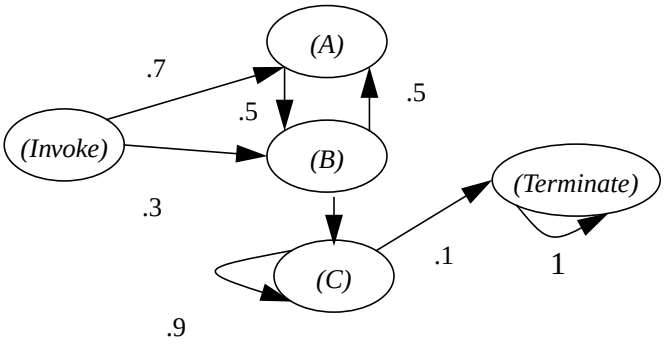


Figure 3: Example Usage Model

Given this definition of a test case, the partitioning of the test space is determined by the last arc taken in the test case. For example, with respect to Figure 3 *(Invoke, A, B)*, *(Invoke, A, B, A, B)*, and *(Invoke, A, B, A, B, C, Terminate)* are all valid, distinct test cases from the usage model. In the example, the first two tests would belong to the same block of the partition since the last arc taken in both tests was *(A,B)*. The third test case would belong to a different block since the last arc traversed in that test was *(C, Terminate)*. Exe-

cuted test cases generated from a usage model partitioned in this manner may be used to calculate the single-use and single-action reliability estimates of the software under test.

More formally, let **R** be the relation on the set of test cases T defined by:

$$(Invoke, \ldots, a, b)\mathbf{R}(Invoke, \ldots, c, d) \Leftrightarrow ((a = c) \wedge (b = d))$$

Clearly **R** is an equivalence relation and T/**R** is a partition.

3.3 Single-Use Reliability and Single-Action Reliability

The Whittaker model [20] estimates software reliability in terms of test cases as "uses", where a use is an executed sequence of actions from *(Invoke)* to *(Terminate)*. For example, a use of word processing software could be to invoke the software, load a document, print the document, and then exit the software. This view of software reliability, the *single-use reliability,* defines the reliability as the probability of the software executing a randomly selected use without a failure. A use is considered to have a failure if at least one failure occurs during the execution of that use. While preserving this definition of reliability, an alternative approach to its estimation is given that does not necessarily yield $R = 1.0$ when random testing reveals no failures. This definition is needed because Cleanroom development and testing often leads to testing results where no failures are seen in random testing. Current reliability estimators in use today, such as the Whittaker model and the sampling theory based model [4], do not provide a meaningful variance in the absence of failures. Therefore it is impossible to define a confidence interval around the estimated

reliability, which in turn means it is impossible to assess the trustworthiness of the estimated reliability.

The *single-action reliability* is introduced to provide an estimate of the probability that a single user action, a single state transition in the usage model, will occur without failure. For example, a single user action involving a word processor might be loading a file.

Field experience shows many testing situations in which pre-test information is known or asserted in terms of individual arcs of the usage model. Both of these reliability estimates make use of this type of information.

Note that it is possible for the single-use reliability and the single-action reliability of a system to be quite different. This is because the single-use reliability depends strongly on the length of a typical use of the software. The longer the typical use of the software, the more chances the software has to fail. Therefore it is possible for software with a high single-action reliability to have a relatively low single-use reliability. For example, consider a model where the probability of each individual user action succeeding is 0.99 and every use is 100 steps long (however rare such a model might be). The single-action reliability would be 0.99 since any given step has a 0.99 reliability. However, for a use to succeed every user action must succeed; therefore the single-use reliability is $(0.99)^{100} = .366$, a much lower value than the single-action reliability.

3.3.1 Testing Records

Five matrices are needed to compute the single-action and single-use reliabilities: the usage model transition matrix **U**, a success matrix **S**, a failure matrix **F,** and matrices of parameters of prior information, **A** and **B**. The transition matrix contains the arc transition probabilities of the usage model. It is created when the model is created.

A success matrix contains the counts of the number of times that each transition has been taken successfully during testing. Note that the success matrix does not contain information about failure states and that normalization of the success matrix would yield the *testing chain* only in the case of no failures.

The record of failures is maintained separately from the record of successes. The failure matrix **F** contains the counts of the number of times that a transition has failed during testing. If a failure is encountered while executing a test case that does not permit testing to continue (a halting failure), then no transitions beyond the failure will be counted in the testing record in either the success matrix or the failure matrix. Since those transitions were not executed during the test, they cannot be counted as successes or failures. Testing will continue with the sequence if possible.

The general process followed in testing using these reliability measures is as follows:

1. Generate sequences from the usage model.
2. Run the sequences until *(Terminate)* unless there is a halting failure.
3. Update the count of successful transitions in **S**.

4. Update the count of failed transitions in **F**.
5. Using the Miller failure rate calculation with test cases partitioned by final arc, estimate the failure rate of each arc in the model. The arc failure rate is defined to be zero if the state transition probability is zero.
6. Estimate the single-action reliability, E(R_a), and single-use reliability, E(R_u).

3.3.2 Arc Failure Rate Calculation

Following Miller, the expected values and variances of the arc failure rate random variables are computed using the Beta distribution.

$$\mathrm{E}(F_{(i,j)}) = \frac{\mathrm{f}_{i,j} + \mathrm{a}_{i,j}}{\mathrm{f}_{i,j} + \mathrm{s}_{i,j} + \mathrm{a}_{i,j} + \mathrm{b}_{i,j}}$$

$$\mathrm{Var}(F_{(i,j)}) = \frac{(\mathrm{f}_{i,j} + \mathrm{a}_{i,j})(\mathrm{f}_{i,j} + \mathrm{s}_{i,j} + \mathrm{b}_{i,j})}{(\mathrm{f}_{i,j} + \mathrm{s}_{i,j} + \mathrm{a}_{i,j} + \mathrm{b}_{i,j})^2(\mathrm{f}_{i,j} + \mathrm{s}_{i,j} + \mathrm{a}_{i,j} + \mathrm{b}_{i,j} + 1)}$$

3.3.3 Single-Action Reliability Estimator

The single-action failure rate can be viewed as the probability of failure of a randomly selected transition from the convergent sequence. In terms of the Miller model, the probability associated with each block of the partition is the long run probability of the arc that defines that block. The long run arc probabilities of U are defined by $\pi(i, j) = \pi(i)\mathrm{u}_{i,j}$.

Theorem 3.2 For each arc in a usage model U, the long run arc probability is equal to the probability of selecting the partition block under **R** identified with the arc, i.e $p_{i,j} = \pi(i)\mathrm{u}_{i,j}$.

Proof: Because the Markov chain representing the usage model is ergodic, in the convergent sequence the probability of selecting an arbitrary arc approaches that arc's long run probability [10]. Selecting an arc at random and taking the sequence beginning with the most recent *(Invoke)* is equivalent to selecting the bin and a test case from it. Therefore, the long run arc probability is equal to the probability of selecting the partition block under **R** identified with the arc. □

Theorem 3.3 The expected value of the single-action reliability is $\mathrm{E}(R_a) = 1 - \mathrm{E}(F_a)$, where $\mathrm{E}(F_a) = \sum_i \left(\sum_j p_{i,j} \mathrm{E}(F_{i,j}) \right)$ is the expected value of the single-action failure rate. (The variance is given by Theorem 3.1.)

Proof: The probability associated with block (*i*,*j*) is $p_{i,j}$. The failure rate associated with block (*i*,*j*) is $F_{(i,j)}$. The expected value of the single-action failure rate follows by the Miller model. □

Because the single-action reliability is the sum of random variables, by the central limit theorem the random variable representing the single-action reliability has an approx-

imately normal distribution. Therefore, given the expected value and variance of the single-action reliability it is possible to compute a c% confidence interval for the single-action reliability.

Example 3.1 Single-Action Reliability

Testing was simulated by assigning each arc an “actual” failure rate and then generating simulated test sequences from the model. The simulated test sequences were generated by taking random walks of the usage model based on the transition matrix. At each transition it was randomly determined whether a failure occurred; success and failure matrices were updated as discussed earlier.

All arcs were assumed to have a failure rate of 0.01 and no prior information was used in the arc failure rate estimates, i.e., all elements of **A** and **B** equal 1. A graph of the single-action reliability is shown in Figure 4.

A 95% confidence interval was computed for the estimated single-action reliability. The confidence interval tightens as testing progresses. The size of the c% confidence interval around the estimated single-action reliability could be used as a stopping criterion, i.e., stop testing when the size of the c% confidence interval is less than some constant ε.

In this example, the single-use reliability estimator converges to 0.99. This is because the actual failure rate of each arc in the usage model is 0.01. Therefore, any arc picked at random from the convergent sequence will have a failure rate of 0.01 and consequently a reliability of 0.99. This means that the actual single-use reliability is 0.99. Because the

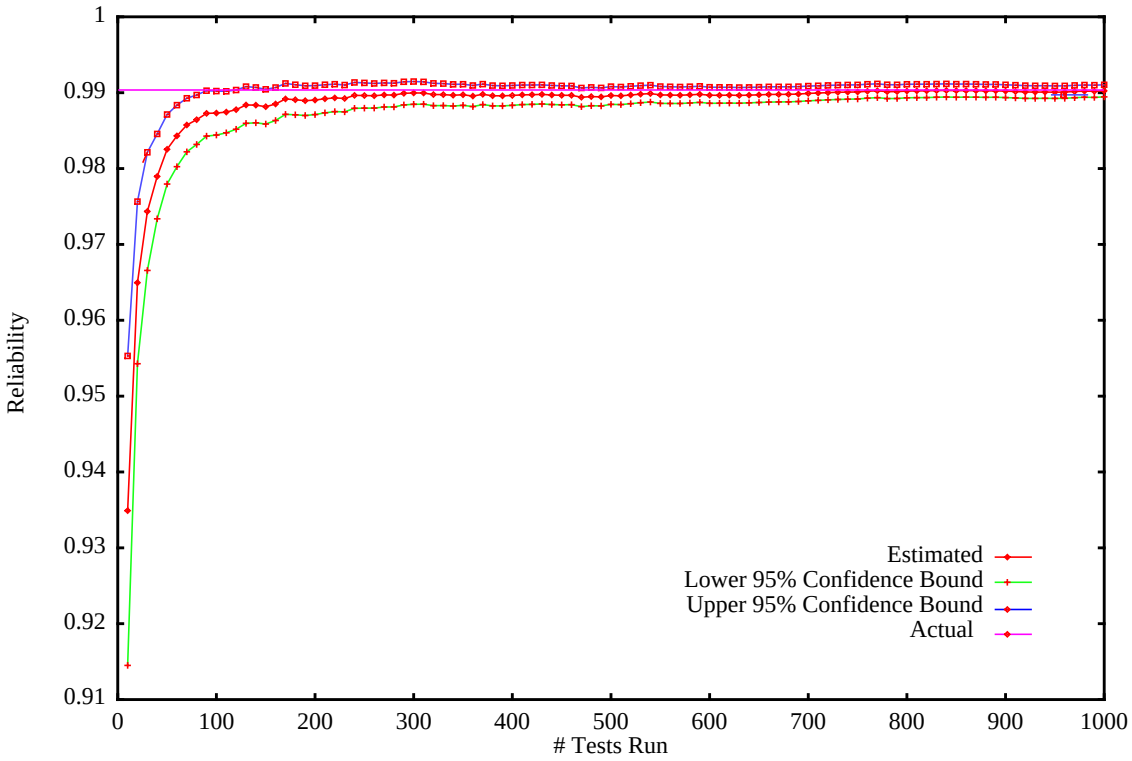


Figure 4: NAS Model, Single-Action Reliability

Miller reliability estimator is consistent and unbiased and the single-action reliability estimator is based on the Miller reliability estimator, the single-use reliability estimator will converge to the actual single-use reliability, which in this case is 0.99.

The variance of the single-action reliability estimator is shown in Figure 5.

3.3.4 Single-Use Reliability Estimator

The single-use reliability, R_u, is defined as the probability of going from *(Invoke)* to *(Terminate)* without encountering a failure. The single-use unreliability, F_u, is the probability of going from *(Invoke)* to *(Terminate)* and encountering at least one failure.

Theorem 3.4 For usage model U, let parameter $z_{(j)}$ denote the probability of encountering at least one failure in the random sequence *(j,k,...,Terminate)*. Then

$z_{(j)} = \sum_k u_{j,k} f_{(j,k)} + \sum_k u_{j,k}(1 - f_{(j,k)}) z_{(k)}$, where $z_{(Terminate)} = 0$ and $f_{(j,k)}$ is the failure rate parameter of arc *(j,k)*.

Proof: Let *(j,k,...,Terminate)* be a sequence generated randomly from usage model U. Let $\phi(i, \ldots, j)$ be a function that gives the number of failures in a sequence *(i,...,j)*. If $(j) = (Terminate)$ the sequence *(j,...,Terminate)* will contain no transitions. Therefore $z_{(Terminate)} = 0$.

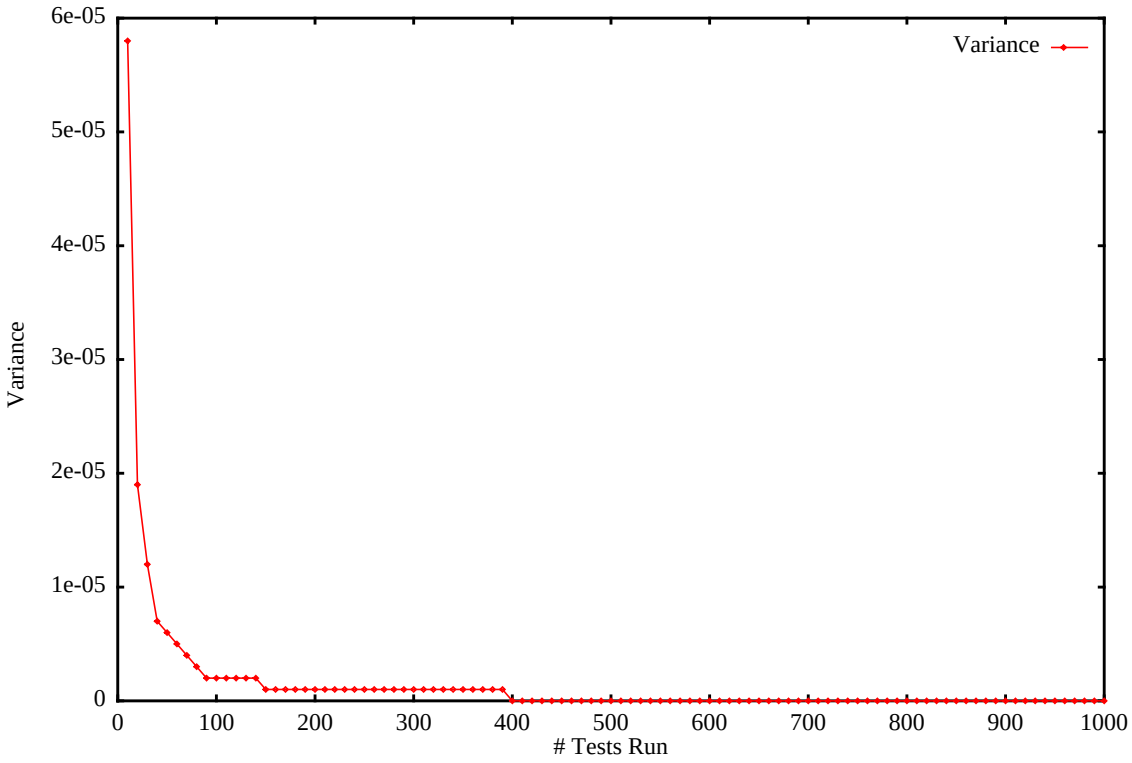


Figure 5: NAS Model, Single-Action Reliability Variance

Since either a failure occurs on the transition *(j,k)* or it does not, for the case of $(j) \neq (Terminate)$:

$$\begin{aligned} z_{(j)} &= \Pr[\phi(j, k, \ldots, Terminate) > 0] \\ &= (\Pr[\phi(j, k) = 1]) + (\Pr[\phi(j, k) = 0])\Pr[\phi(k, \ldots, Terminate) > 0]) \\ &= (\Pr[\phi(j, k) = 1]) + (\Pr[\phi(j, k) = 0])z_{(k)} \\ &= (\Pr[\phi(j, k) = 1]) + (1 - \Pr[\phi(j, k) = 1])z_{(k)} \end{aligned}$$

If *(j)* has outgoing arc probabilities $u_{j,1}, u_{j,2}, \ldots, u_{j,u}$, then the probability of choosing and failing on an outgoing arc is $u_{j,1} f_{(j,1)}, u_{j,2} f_{(j,2)}, \ldots, u_{j,u} f_{(j,u)}$, respectively. These are mutually exclusive events, since only one arc is chosen. Thus:

$$\Pr[\phi(j, k) = 1] = \sum_k u_{j,k} f_{(j,k)}$$

Substituting gives:

$$\begin{aligned} z_{(j)} &= \left(\sum_k u_{j,k} f_{(j,k)}\right) + \left(1 - \sum_k u_{j,k} f_{(j,k)}\right) z_{(k)} \\ &= \sum_k u_{j,k} f_{(j,k)} + \sum_k u_{j,k} (1 - f_{(j,k)}) z_{(k)}. \end{aligned}$$

□

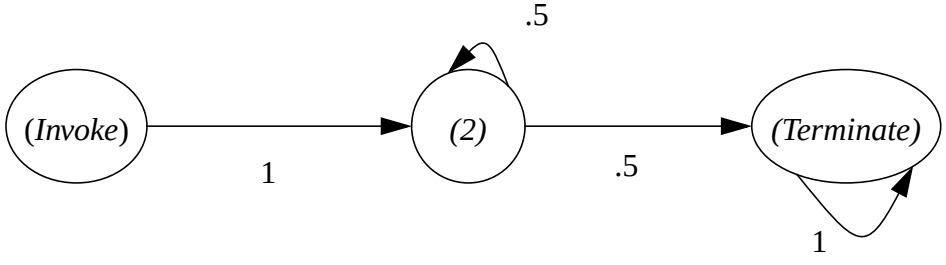
The actual single-use reliability of the software is defined as $r_u = 1 - z_{(Invoke)}$. However, the actual arc failure rates are unknown and must be represented as random variables. Replacing all arc failure rate parameters $f_{(j,k)}$, in the set of equations from Theorem 3.4 with corresponding random variables, $F_{(j,k)}$, results in the set of equations $Z_{(j)} = \sum_k u_{j,k} F_{(j,k)} + \sum_k u_{j,k} (1 - F_{(j,k)}) Z_{(k)}$ and $Z_{(Terminate)} = 0$. Because all $Z_{(j)}$ are functions of random variables, all $Z_{(j)}$ are random variables. $Z_{(j)}$ is a random variable

representing the probability of encountering at least one failure when executing a sequence beginning with *(j)* and ending with *(terminate)*.

3.3.4.A Expected Value of the Single-Use Failure Rate

We need the expected value of the single-use failure rate, $\mathrm{E}(F_u) = \mathrm{E}(Z_{(Invoke)})$. Given the expected value of the single-use failure rate, the expected value of the single-use reliability is defined as $\mathrm{E}(R_u) = 1 - \mathrm{E}(Z_{(Invoke)})$. A naive approach to analytically deriving the expected value of $Z_{(Invoke)}$ would be to replace all occurrences of random variables $F_{(j,k)}$ with their respective expected values. This replacement would result in the system of linear equations $\tilde{\mathrm{E}}(Z_{(j)}) = \sum_{k} \mathrm{u}_{j,k}\mathrm{E}(F_{(j,k)}) + \sum_{k} \mathrm{u}_{j,k}(1 - \mathrm{E}(F_{(j,k)}))\tilde{\mathrm{E}}(Z_{(k)})$ and $\tilde{\mathrm{E}}(Z_{(Terminate)}) = 0 = \mathrm{E}(Z_{(Terminate)})$, which could then be solved for an approximation of $\mathrm{E}(Z_{(Invoke)})$. However, the solution will generally not yield $\mathrm{E}(Z_{(Invoke)})$ because a cycle in the usage model will result in the violation of an independence assumption.

In illustration of the violation, consider the following model. It is possible to define the



following system of equations describing $Z_{(Invoke)}$.

$$Z_{(Invoke)} = F_{(Invoke,\,2)} + (1 - F_{(Invoke,\,2)})Z_{(2)}$$

$$Z_{(2)} = (0.5)F_{(2,\,2)} + (0.5)F_{(2,\,Terminate)} + (0.5)(1 - F_{(2,\,2)})Z_{(2)} + (0.5)(1 - F_{(2,\,Terminate)})Z_{(Terminate)}$$

$$Z_{(Terminate)} = 0$$

Solving these equations yields $Z_{(Invoke)} = F_{(Invoke,\,2)} + (1 - F_{(Invoke,\,2)})\left(\frac{F_{(2,\,2)} + F_{(2,\,Terminate)}}{1 + F_{(2,\,2)}}\right)$. Because the random variable $F_{(2,\,2)}$ appears in both the numerator and the denominator of the equation defining $Z_{(Invoke)}$, the random variables defined by the numerator and the denominator of the equation are not independent and hence the expected value of the fraction is not generally equal to the expected value of the numerator divided by the expected value of the denominator. Moreover, this is not a special case of equality as can be shown by a counter-example. Therefore the expected value of $Z_{(Invoke)}$ cannot be computed as suggested. In general, if the model contains cycles, some $F_{(j,\,k)}$ will appear in both the numerator and denominator of the equation defining $Z_{(Invoke)}$. Since most usage models contain cycles, the result is at best an approximation. It appears as if this might be a useful approximation, but we are unable to quantify the degree of error.

A value arbitrarily close to $\mathrm{E}(Z_{(Invoke)})$ may be approximated through simulation. Given the usage model U, associated matrices **F**, **S**, **A**, and **B**, and *j*, the number of iterations in the simulation (sequences to generate), the simulation algorithm is as follows.

```
s = 0
for x = 1 to j do
     seq = generate random sequence from U
     use seq to update F and S
     arc_rel = 1
     for each distinct arc (i,j) in seq do
          // Using the Beta distribution, randomly choose a
          // reliability for each arc.
          t = (1-Beta(F(i,j)+A(i,j),S(i,j)+B(i,j))
          arc_rel = arc_rel * t^(# times (i,j) is in seq)
     endfor
     s = s + arc_rel
endfor
exp_seq_rel = s/j
```

`exp_seq_rel` is an estimate of $\mathrm{E}(R_u) = 1 - \mathrm{E}(Z_{(Invoke)})$. The simulation can be viewed as taking samples from a sample space containing all possible sequence reliabilities. The reliability of a particular sequence s, or chance of successfully executing that sequence, is defined by the random variable $R_s = \prod_{i,j} (1 - F_{(i,j)})^{n_{(i,j)}}$, where $F_{(i,j)}$ is the random variable representing the failure rate of arc (i, j) and $n_{(i,j)}$ is the number of times arc (i, j) appears in the sequence. Therefore the value of each point in the sample space depends both on the path taken through the model (i.e. the sequence) and the values taken by the arc failure rate random variables along that path.

The sampling may be performed with a stratified sampling scheme. The sample space may be partitioned based on the sequence, i.e., all sequence reliabilities whose computation is based on the same sequence will go in the same block of the partition. The proba-

bility associated with each block in the partition is the probability of randomly generating the sequence associated with the block from the model. The probability of drawing an item from within a block is the joint probability of the failure rates of the arcs in the sequence. Because the random variables representing the arc failure rates are assumed to be independent, the joint probability of realizations of the arc failure rate random variables is simply the product of probabilities of the individual arc failure rates.

In the simulation the block (i.e. sequence) is chosen first. Then a sequence reliability is chosen from the block by independently sampling the failure rate of each arc in the sequence. The arc failure rates may be sampled independently because they are independent. As the sample size *j* grows, the average value of the sample approaches the expected value of the sample space, which in this case is $\mathrm{E}(R_u)$.

3.3.4.B Variance of the Single-Use Failure Rate

The variance of the single-use failure rate may also be estimated through simulation. In general the sample variance is computed as $s'^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$, where n is the sample size, y_i is the value of the ith data point, and $\bar{y}$ is the sample average [4]. As n grows larger, s'^2 approaches the population variance. In the case of the simulation, j is the sample size, y_i is the value of the ith sampled sequence reliability, and $E(r)$ is the sample

average. Therefore the sample variance of the single-use failure rate is

$$s'^2 = \frac{1}{j}\sum_{i=1}^{j}(y_i - E(r))^2 \ .$$

Note that there are two sources of variation within a sample, namely the selected sequences and the selected failure rates of the arcs in the sequences. The variation of the arc failure rates will decrease as testing progresses and more information about the arc failure rates is collected. However, the variation of selected sequences is purely a function of the usage model and hence will not change as testing progresses. This means that the sample variance will not necessarily approach zero as the number of tests run approaches infinity. However, the variance of the random variables representing the arc failure rates will approach zero as the number of tests run approaches infinity. Therefore, as the number of test cases increases the sample variance becomes dominated by the variation in selected sequences, which is constant. This implies that the sample variance will approach a constant value as the number of test cases increases.

Example 3.2 Single-Use Reliability

The testing based on the NAS model was simulated in the same manner as the example of the single-action reliability. All arcs were assumed to have a failure rate of 0.01 and no prior information was used in the arc failure rate estimates, i.e. a=b=1. The number of iterations used in the simulation of the expected value of the single-use failure rate was

60,000. A graph of the single use reliability and the Whittaker reliability is shown in Figure 6.

In this example both the single-use reliability and the Whittaker reliability were computed for the same set of test data. In the testing shown for this particular example the Whittaker reliability is always higher than the single-use reliability. However, this is will not always be true, i.e., the Whittaker reliability does not dominate the single-use reliability.

As stated earlier, the variance of the single-use reliability depends on two sources of variation, the variation of the makeup and length of randomly generated sequences and the variation of the random variables representing arc failure rates. As the variance of the arc failure rate random variables approaches zero, the variance of the single-use reliability will approach a constant. The estimated value of the constant to which the variance will converge is shown in Figure 7.

3.4 Use of Pretest Reliability Information

3.4.1 Decomposition of the Problem of Pretest Reliability Assessment

One of the strengths of the arc-based reliability models is their ability to make use of pretest reliability information. Because the pretest reliability information is associated with the reliability of a specific arc rather than with the reliability of the entire system, it

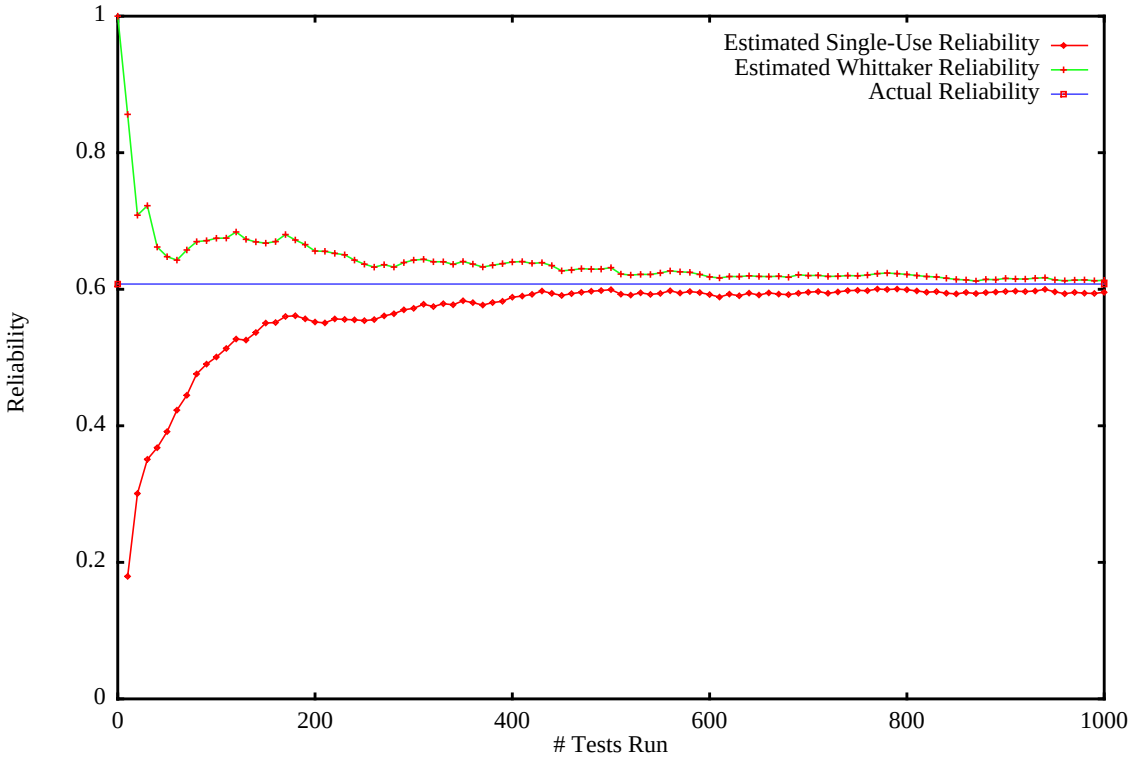


Figure 6: NAS Model, Single-Use Reliability

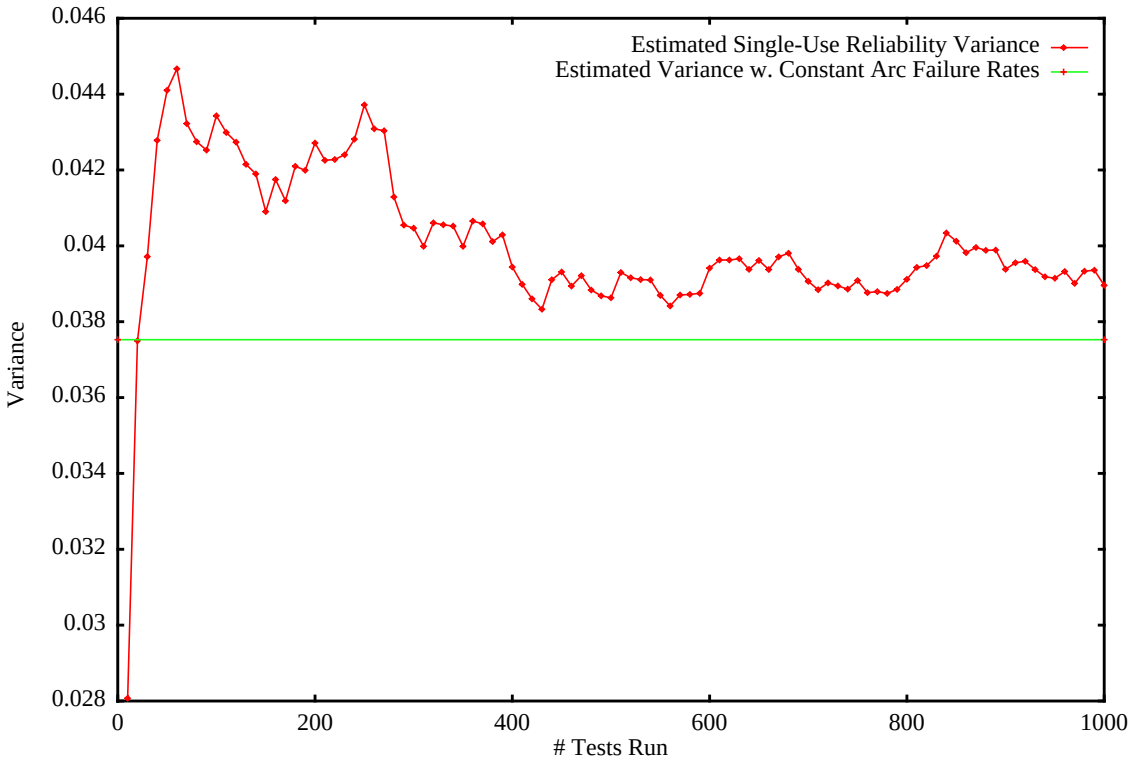


Figure 7: NAS Model, Single-Use Reliability Variance

may be possible for the tester to make a more accurate pretest estimate of the reliability of the software than if the pretest reliability information applied to the system as a whole. Each user action, represented by arcs in the usage model, will cause a portion of the system under test to be executed.

Given a mapping between arcs in the usage model and code implementing the user actions associated with the arcs, it is possible to derive a relatively fine grained pretest analysis of the reliability of the software. Each user action will be implemented by a unit of code that will usually be smaller, in many cases extremely smaller, than the size of the entire system. If the units of code to be analyzed are smaller than the entire system, there is a greater chance of making an accurate assessment of the pretest reliability of the small units of code than the system as a whole. In a sense the problem of assessing the pretest reliability of the system is decomposed into a set of smaller, possible more easily solved problems.

3.4.2 Sources of Pretest Reliability Information

Pretest reliability information can be gleaned from a number of sources, including the following [15].

- **Code Analysis** - Given a mapping between the usage model and the code, it is possible to analyze each piece of code to estimate the error-proneness of the code. Analyses include cyclomatic complexity metrics, function cohesion and strength, branching factor, and module size [6].

- **Design Records** - If reviews of the system design and code were conducted, the results of the reviews can be used to asses the reliability of the code. For example, a piece of code whose specification contained a large number of errors during the design phase may be more error prone than a piece of code whose specification contained few errors.
- **Previous Failure Data** - If modifications are being made to an existing, fielded system, failure data from the field and old testing records can be used to assess the reliability of the modified system. Pieces of code that exhibited low field reliability or contained many errors during previous testing are likely to be error prone in the modified system.
- **Skill of Personnel** - The abilities of the programmers can be taken into account when assessing the pretest reliability of the software. A piece of code programmed by team of new hires may be more error prone than a piece of code written by a team of experienced professionals.
- **Non-Random Tests** - Non-random tests run prior to random testing provide some anecdotal evidence of the systems' reliability.

Example 3.3 Single-Use Reliability Estimate With Prior Information

When using accurate prior information about the arc failure rates, the estimated reliability approaches the true reliability sooner in testing than when no prior reliability information is used. In this example, as shown in Figure 8, when using no prior reliability

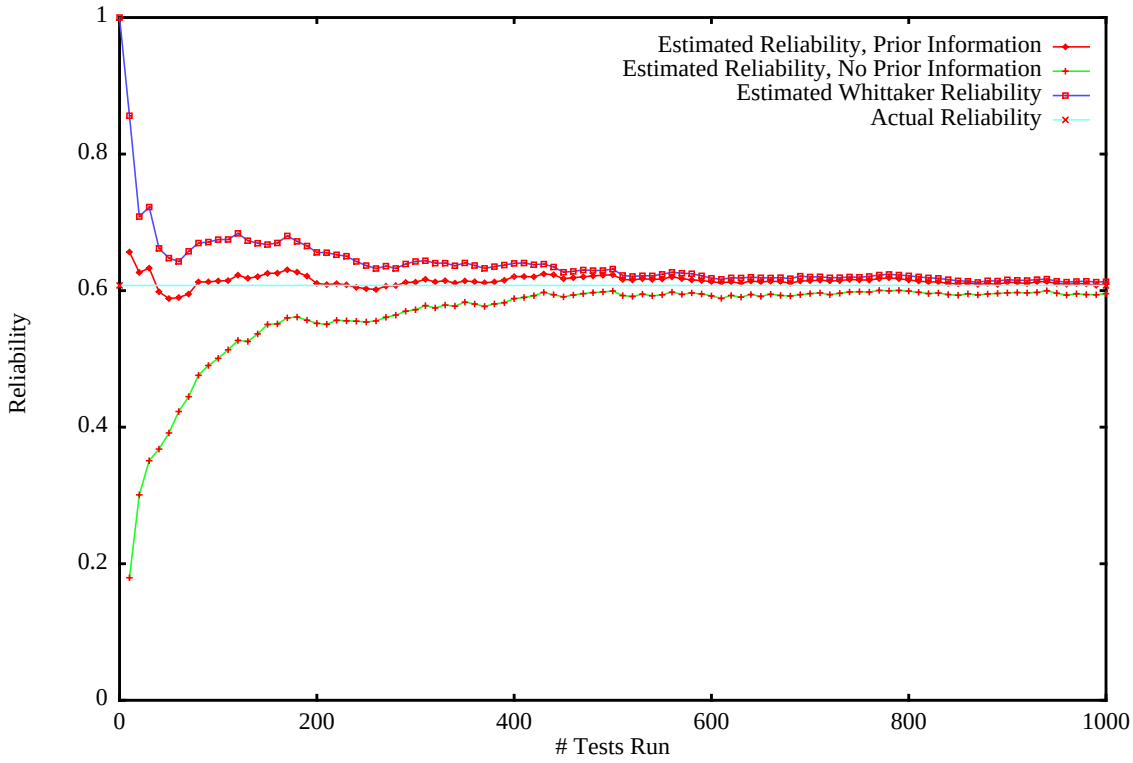


Figure 8: NAS Model, Single-Use Reliability, Prior Information

information, it takes approximately 400 tests for the estimated reliability to approach the actual reliability as closely as the estimated reliability using prior reliability information (a=1, b=100) does after 100 tests. This represents a savings of 300 test cases that can be realized if accurate reliability information is available prior to testing.

In this example the single-use reliability with prior information slowly converges to the single-use reliability with no prior information. As testing progresses actual testing experience will dominate the prior information. The time at which the single-use reliability with prior information converges to the single-use reliability without prior information will depend on the strength of the prior information.

As with the estimated reliability itself, the variance of the reliability estimated without prior information will eventually converge to the variance of the reliability estimated with prior information, as is shown in Figure 9.

3.5 Conclusions about Arc-Based Reliability Models

Product managers are interested in reliability estimates during the development process, to support product release decisions, for estimating maintenance budgets, and during maintenance activities. This is a wide range of uses and decisions to support. Arc based reliability models, used with a base of experience and maturity, offer great improvement over present practice. Use of prior information and the ability to make more decisions at the arc level will permit tailoring of test plans to better support each situation.

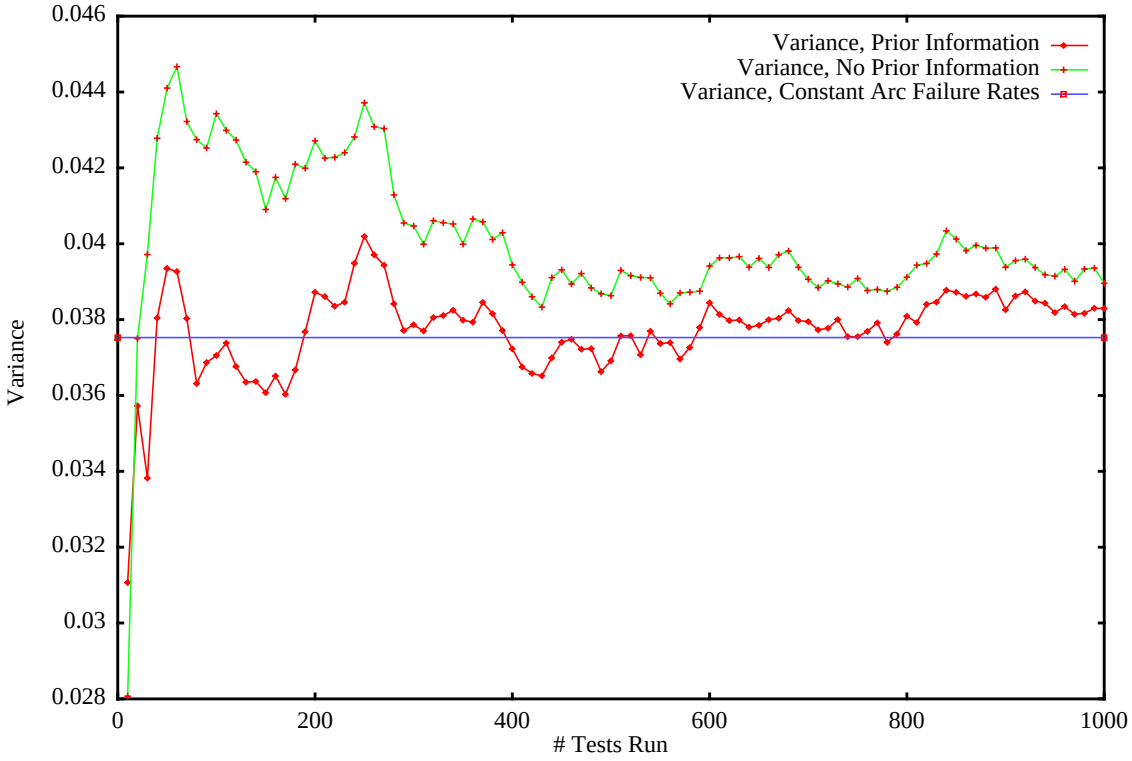


Figure 9: NAS Model, Single-Use Reliability Variance, Prior Information

Most developers and test managers continue to think in terms of actions and functions, rather than complete scenarios of use, even though their technical practices do not justify that point of view. Many will find the single-action reliability measures more intuitive, especially during development and early testing. Also, the availability of a variance associated with the reliability will assist in stopping decisions.

People associated with final system test and product release do tend to think in terms of scenarios of use and the usage patterns of various environments. The following features of the single-use reliability will be appreciated by practitioners:

- A starting value other than one
- Availability of a variance estimate
- Information in the absence of failures during testing
- Greater stability during early testing

Chapter 4

Usage Model-Based Partition Testing

The statistical technique presented here is the use of partitioning to improve sampling efficiency. There is a “partition testing” literature [8,19] in software engineering which varies somewhat in terminology and conclusions from the statistical literature on partition sampling [4,14]. In this work we will follow the statistical terminology and apply it to two goals of software testing, the estimation of software reliability and failure detection potential.

4.1 Partition Testing

Sampling theory is a mature area of statistics. To take advantage of stratified sampling methods it is necessary to define the sample in terms of a mathematical partition of the population. Sample allocations are then calculated for the blocks of the partition.

Testing software by using random samples of test cases drawn from blocks of a partition defined on the population of all test cases is called “partition testing.” In this work testing using a random sample drawn from the unpartitioned population will be called “simple random testing” to emphasize that both are based on random samples. Under proportional allocation of tests to blocks, partition testing will always perform at least as well as simple random testing, in terms of variances of estimators and failure detection probability [4,14]. Moreover, partition testing may outperform simple random testing, depending on the nature of the population and the degree of insight used in creating the partitioning strategy; which is to say that results of a given quality can be obtained with less testing or that a given amount of testing will produce higher quality results. Partitions allocate the set of all possible uses of the software under test into blocks, each of which can be tested independently of the other blocks. Simple random testing can be viewed as a special case of partition testing consisting of a single block.

The software engineering literature notes frustrations with partition testing that arise from difficulties in defining a mathematically correct partition, deriving analytical properties of partitions and test cases, expenses associated with constructing partitions and test cases for the partitions, and ineffective allocation of tests to blocks. As will be demonstrated below, these difficulties are resolved through application of graph theory and Markovian analysis to the usage model, thus facilitating greater efficiency in software testing through sampling theory.

4.2 Basic Results

The following are basic results of sampling theory [4]:

- Optimal allocation of samples to blocks of the partition based on a fixed total sampling budget.
- Optimal allocation of samples to blocks to achieve a given sample estimator variance.
- Estimate of the percentage of the sample population that falls into a certain category.
- Variance of the estimate of the percentage of the sample population that falls into a certain category.

To apply the basic results the following items of information are needed:

- A usage model M representing all possible uses of the software, at some level of abstraction.
- A partitioning rule in terms of M that creates K blocks.
- Probabilities $p_1, p_2, \ldots, p_K$, where p_i is the probability mass associated with block i.
- Estimated failure rates $f_1, f_2, \ldots, f_K$, where f_i is the pre-test estimated failure rate of block i.
- The number of tests to be run n, or the budget (allowable cost) c.
- If tests are to be allocated based on a fixed testing budget, then testing costs for each block, $c_1, c_2, \ldots, c_K$, are needed.

4.2.1 Optimal Test Allocation

The optimal test allocation can be computed to minimize the variance of the software reliability estimator given a constraint on the total number of tests to be run or the total cost to be allowed in testing. The optimal allocation of tests to the K blocks given a total fixed testing cost c makes the assumption that $c = c_0 + \sum_{h=1}^{K} c_h n_h$, where c_0 is the cost of general testing overhead, c_h is the cost of running a test from block h, and n_h is the number of tests run from block h. Note that the optimal allocation of tests to blocks based on a fixed testing cost takes into account the fact that it may be more expensive to run tests from one block than another.

The optimal test allocation for block h is calculated by:

$$n_h = n\left(\frac{p_h\sqrt{\frac{f_h(1-f_h)}{c_h}}}{\sum_{i=1}^{K} p_i\sqrt{\frac{f_i(1-f_i)}{c_i}}}\right)$$

This equation depends on the value of n, which is not directly defined when c is given. n may be calculated as:

$$n = \frac{(c - c_0)\sum_{h=1}^{K}\left(\frac{p_h\sqrt{f_h(1-f_h)}}{\sqrt{c_h}}\right)}{\sum_{i=1}^{K}(p_i\sqrt{f_i(1-f_i)}\sqrt{c_i})}$$

The optimal allocation of tests for block h given a fixed total number of tests is equivalent to testing where tests from all blocks have a fixed cost of one. The optimal allocation is computed by:

$$n_h = n\left(\frac{p_h\sqrt{f_h(1-f_h)}}{\sum_{i=1}^{K} p_i\sqrt{f_i(1-f_i)}}\right)$$

Given a target variance V, the minimum cost allocation of tests to achieve V is computed in the same manner as the optimum allocation of tests given a fixed testing budget. The only difference is the equation used to compute n:

$$n = \frac{\left(\sum_{i=1}^{K} p_i\sqrt{f_i(1-f_i)}\sqrt{c_i}\right)\sum_{h=1}^{K}\left(\frac{p_h\sqrt{f_h(1-f_h)}}{\sqrt{c_h}}\right)}{V}$$

4.2.2 Estimate of Overall Failure Rate

After the number of tests has been allocated to each block, test cases are selected randomly (generated from the usage model) based on the use distribution of each block. The

tests are then run and o_h, the number of failures observed during testing in block h, is recorded.

The overall failure rate of the software is estimated as $f = \sum_{h=1}^{K} p_h(o_h/n_h)$, where o_h/n_h is the post-test estimated failure rate of block h.

4.2.3 Variance of Estimate of Overall Failure Rate

The variance of the estimate of the overall failure rate is

$$\mathrm{Var}(f) = \sum_{h=1}^{K} p_h^2\left(\frac{(o_h/n_h)(1-(o_h/n_h))}{n_h - 1}\right).$$

4.2.4 Probability of Finding a Failure

The estimated probability of finding at least one failure during testing can be calculated. Given $f_1, f_2, \ldots, f_K$, the pre-test estimated failure rates of the partition blocks, and $n_1, n_2, \ldots, n_K$, the number of tests allocated to the blocks, the estimated probability of finding at least one failure during testing is $1 - \prod_{h=1}^{K} (1 - f_h)^{n_h}$.

4.2.5 Expected Number of Failures Found

The expected number of failures to be found during testing can also be calculated. Given $f_1, f_2, \ldots, f_K$ and $n_1, n_2, \ldots, n_K$, as defined above, the expected number of failures to be found during testing is $\sum_{h=1}^{K} f_h n_h$.

4.3 Model-Based Partitioning Strategies

A good partitioning strategy is one that results in blocks with different proportions of failures. The greatest gain in precision is realized when all blocks of the partition are homogeneous, i.e., all tests within a block either fail or all of the tests succeed [4,8,14,19].

Given usage model M, let **W** be a matrix where $w_{i,j}$ is a pre-test estimate of the probability of encountering a failure while executing the code to achieve transition from *(i)* to *(j)*, that is, while traversing arc *(i,j)*. M and **W** may be used to automatically partition the space of all possible uses (or test cases). All of the information needed to perform the basic stratified sampling calculations discussed in the previous section may be obtained from M and **W**. Since the partitioning and all required information are readily available, the additional cost associated with partition testing is negligible, thereby making partition testing potentially worthwhile even in cases where it provides only modest gains over simple random testing.

Two automatic partitioning schemes based on the usage model will be illustrated. Parameters are derived analytically for one and through simulation for the other. Additional partitioning schemes are certainly possible.

4.3.1 Analytical Automatic Partitioning Scheme

Suppose that certain arcs in the model M have (or potentially have) significantly higher estimated failure rates than the rest of the arcs. These arcs will be referred to as uncertain (dangerous, or high risk) arcs. Such arcs might be associated with features added in a new increment of code, code altered through maintenance, or a unit of code acquired from a third party, for example. Suppose further that the structure of the model (the graph) is such that it is possible to generate test cases from M that pass through one or more of the uncertain arcs as well as to generate test cases that do not pass through any of the uncertain arcs.

This makes it possible to automatically partition the test cases into two blocks, block U which contains all test cases that pass through at least one uncertain arc and block S which contains all the safe test cases that do not pass through any uncertain arcs. The blocks are created in an attempt to make the failure rate of block U significantly higher than the failure rate of block S and hence increase the efficiency of testing. Of course, in general, partitions with more than two blocks could be defined.

The probability mass associated with blocks U and S can be computed directly from the model. The probability of drawing a test case from block U can be calculated as follows:

1. Add one place holder state *(k)* to the model for each uncertain arc *(i,j)*.
2. Connect these states into the model by removing arcs *(i,j)* and then adding arcs *(i,k)* and *(k,j)*.
3. Make all place holder states absorbing; this removes arc *(k,j)*.
4. Make the termination state absorbing.
5. Calculate P_S = Pr[Absorbtion in *(Terminate)*], the probability of being absorbed in the termination state when starting from the invocation state, and therefore not traversing any uncertain arcs.
6. Calculate P_U, the probability of traversing an uncertain arc as $1-P_S$.

Model M will be used as the basis to create model M_S, which will be used to generate test cases from block S, and model M_U, which will be used to generate test cases from block U. M_S is created by removing all uncertain arcs from M and then normalizing the resulting probability matrix. M_U is created as follows:

1. Make two copies of M called A and B.
2. In model A remove all regular (not uncertain) arcs to *(Terminate)*, i.e., leave the uncertain arcs to termination if any exist.

3. For each uncertain arc *(i,j)* in model A, change the arc to *(i,j')* where *(j')* is the state corresponding to *(j)* in model B. This produces a new model M_U with *(Invoke)* of A and *(Terminate)* of B.
4. M_U is the model created in step three by connecting models A and B.
5. Normalize M_U.

The failure rates associated with blocks U and S, f_U and f_S respectively, may now be estimated by using the single use arc based reliability model calculated using M_U and M_S and arc failure estimates $\mathbf{W}_U$ and $\mathbf{W}_S$ (which can be derived from **W**).

If the number of tests to be run is given, all information needed to proceed with testing has been assembled. However, if a fixed cost for testing is given, it is necessary to compute the expected cost of running a test from blocks U and S. If the testing budget is constrained by the number of testing steps that may be taken rather than by the number of complete tests that may be run, allocation of tests based on a fixed cost c should be used. Allocation of tests based on a fixed cost rather than a fixed number of tests takes into account that some tests are more expensive (e.g., longer) to run than other tests.

The expected cost of running a test case from an arbitrary state *(i)* to *(Terminate)* is

$$d_{(i)} = \sum_{j=1}^{N} m_{i,j}(x_{i,j} + d_{(j)})$$

In the above equation, $x_{i,j}$ is the cost of taking arc *(i,j)* and $m_{i,j}$ is the probability of taking arc *(i,j)*. Note that if all arc costs $x_{i,j} = 1$ the above calculation is equivalent to the expected number of steps from state *(i)* to *(Terminate)*. Given $d_{(Terminate)} = 0$, it is now

possible to define a system of linear equations that can be solved for $d_{(Invoke)}$. The expected costs of tests from blocks U and S can be computed as the expected cost of going from invocation to termination in models M_U and M_S respectively.

Example 4.1 Analytical Automatic Partitioning Scheme

Consider the example usage model. Arcs *(S2,S3)* and *(S11,S10)* are uncertain. The uncertain arcs have failure rates of 0.2 and the background failure rate is .0001. All arcs have unit testing cost.

Tables 7 and 8 are computed analytically from the example usage model. The test allocation was calculated for a fixed testing budget of 5000 units (each unit being one transition in the usage model).

Note that a larger number of tests are allocated to block U than to block S. Factors that will increase the number of tests allocated to a block are:

- Less expensive to run tests in the block.
- Greater probability mass than other blocks.
- The block is more variable internally.

Table 7: Partition Characteristics

Block	Probability Mass	Estimated Failure Rate
U	.575	.326
S	.425	.00530

Table 8: Test Case Allocation

Block	Number of Tests	Expected Test Cost
U	51	89.8
S	7	53.7

Given the allocation, Tables 9 and 10 can be computed for partition testing and simple random testing.

Note that partition testing is expected to find approximately four more failures than simple random testing. Partition testing is also expected to reduce the variance by a factor of ten when compared to simple random testing.

4.3.2 Simulation-Based Automatic Partitioning Scheme

The simulation-based automatic partitioning scheme partitions the population of software uses into two blocks based on the estimated failure probability of each use. If a particular use has an estimated failure probability above a given threshold ϑ, it will be part of block F. All other uses will belong to block S.

$p_1, p_2, \ldots, p_K$, $f_1, f_2, \ldots, f_K$, and $c_1, c_2, \ldots, c_K$, may be estimated through simulation. The accuracy of estimates will increase as j, the number of sequences used in the simulation, increases. As j becomes sufficiently large, the estimates will approach their long-run values, although not at a uniform rate. The values computed in the simulation provide the information needed to support stratified sampling.

The generation of test cases from a particular block may be done by repeatedly generating sequences randomly from the model until a sequence is generated that belongs in the desired block as judged by its individual probability of failure. Alternatively, all sequences may be classified as they are generated and buffered for future use.

Table 9: Comparison by Failure Rate and Variance

	Expected Failure Rate	Expected Variance
Partition Testing	.190	.000300
Simple Random Testing	.192	.00230

Table 10: Comparison by Detection of Failures

	Probability of Finding a Failure	Expected Number of Failures
Partition Testing	~1.00	16.7
Simple Random Testing	~1.00	12.9

Example 4.2 Simulation-Based Automatic Partitioning Scheme

Using the same model as before, Tables 11 and 12 are computed analytically and by simulation of 1000 sequences. All test cases with estimated failure probabilities greater than 0.2 are members of block F. The test allocation was calculated for a fixed testing cost of 5000 units.

Given the allocation, Tables 13 and 14 can be computed for partition testing and simple random testing.

In this example partition testing is expected to find approximately nine more failures than simple random testing. Partition testing is also expected to reduce the variance by a factor of approximately twenty when compared to simple random testing.

4.4 Conclusions about Usage Model-Based Partition Testing.

If tests are allocated to partition blocks in the optimal manner, partition random testing will always perform at least as well as simple random testing. Gains are realized from partition testing when the partitioning strategy creates near-homogeneous blocks, i.e., blocks in which all tests either fail or succeed. Therefore the choice of a partitioning scheme is very important.

Table 11: Partition Characteristics

Block	Probability Mass	Estimated Failure Rate
F	.586	.331
S	.414	.00360

Table 12: Test Case Allocation

Block	Number of Tests	Expected Test Cost
F	65	72.4
S	8	36.0

Table 13: Comparison by Failure Rate and Variance

	Expected Failure Rate	Expected Variance
Partition Testing	.196	.000100
Simple Random Testing	.192	.00230

Table 14: Comparison by Detection of Failures

	Probability of Finding a Failure	Expected Number of Failures
Partition Testing	~1.00	21.6
Simple Random Testing	~1.00	12.9

The partitioning schemes presented in this chapter use pre-test estimates of the failure rate of the software in an attempt to create homogeneous partitions. In the event that completely homogeneous blocks cannot be created (as may often be the case in real world testing problems), partitions should be created so as to maximize the difference in failure rates between the partition blocks. Since the partitioning is completely automated, the cost of performing the partitioning is negligible. Thus, it is practical to consider partition testing even if the gains in efficiency and accuracy prove to be modest. However, partition testing has the potential to deliver significant gains in testing efficiency.

Chapter 5

Stopping Criteria

The decision to stop testing can be based on a number of criteria, such as:

- The confidence in a reliability estimate. If a reliability estimation technique is used that has an associated variance, a confidence interval can be computed around the reliability estimate. Testing can be stopped when a certain confidence in the reliability estimate has been achieved.
- The degree to which testing experience has converged to the expected use of the software. Given a testing record and a usage model it is possible to check the degree to which the testing experience matches the expected use of the software. Once the expected use of the software has been tested to a satisfactory degree testing may be stopped.
- Model coverage. Testing may be stopped based on a degree of state, arc, or path coverage during crafted and random testing.

In practice it is best to use multiple stopping criteria. For example, further evaluation of the testing performed is needed if the measure of correspondence between testing expe-

rience and expected use of the software indicates that the testing experience closely matches the expected use of the software yet the variance of the reliability estimate is unacceptably large.

5.1 Tests of Equality of Testing Experience and Expected Use

Currently, two methods are used to compare testing experience with expected use of the software, the Euclidean distance between the usage chain and the testing chain and the Kullback discriminant from the usage chain to the testing chain. Both of these methods compare the current testing chain with the usage chain and provide an indication of the degree to which the testing chain matches the usage chain. However, the Kullback discriminant provides a more accurate indication of the similarity of the usage chain and the testing chain.

5.1.1 The Euclidean Distance

The Euclidean distance is computed as $\sqrt{\sum_{i,j}(u_{i,j} - t_{i,j})^2}$ where $u_{i,j}$ and $t_{i,j}$ are the probabilities of going from state (i) to state (j) in the usage chain and the testing chain, respectively.

As stated earlier, the Euclidean distance can be an inaccurate measure of the similarity of two usage models. For example, consider the usage model shown in Figure 10.

Now suppose that two different testing chains with extreme differences resulted from two separate testing experiments. The testing chains are shown in Figures 11 and 12.

In testing chain **A** the probabilities of the arcs exiting (*Invoke*) match the corresponding arcs in the usage model. However, the probabilities of the arcs in the cloud containing the majority of the model structure of testing chain **A** do not match the corresponding arcs in the usage model. In testing chain **B** the situation is reversed. The arcs exiting (*Invoke*) in testing chain **B** do not match the corresponding arcs in the usage chain but the probabilities of the arcs in the cloud in testing chain **B** do match the probabilities of the corresponding arcs of the usage model. Because testing chain **B** has more arc probabilities in common with the usage model than testing chain **A**, the Euclidean distance will indicate that testing chain **B** is much closer to the usage chain than testing chain **A**, or in other words the testing performed to create testing chain **B** will be interpreted as being more representative of the expected use of the software than the testing experience represented by testing chain **A**. If the Euclidean distance is interpreted in this manner the software organization runs the risk of wasting time fixing relatively unimportant faults uncovered through testing of the cloud and runs the risk of missing important faults that would be exposed through testing of the transition from *(Invoke)* to *(State A)*. Thus, the Euclidean distance has the potential for misleading interpretation.

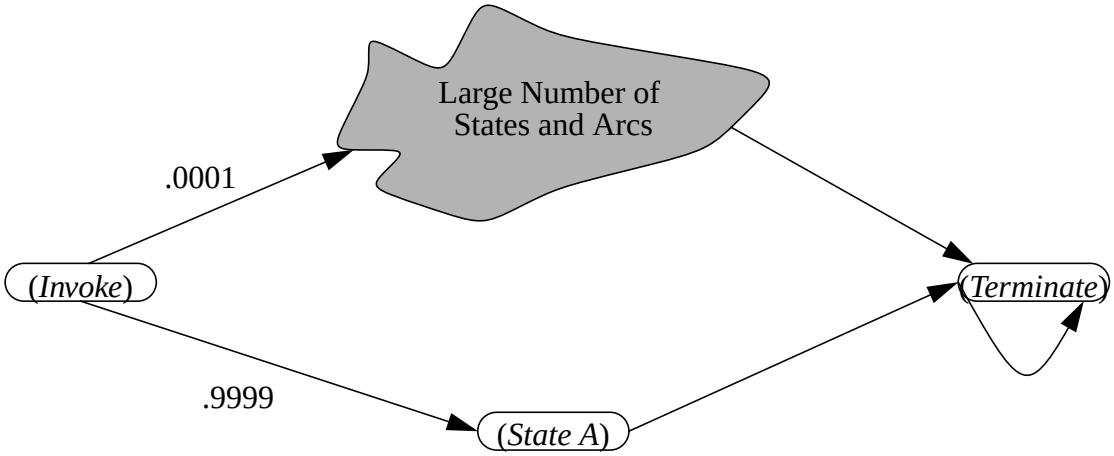


Figure 10: Euclidean Distance, Example Model

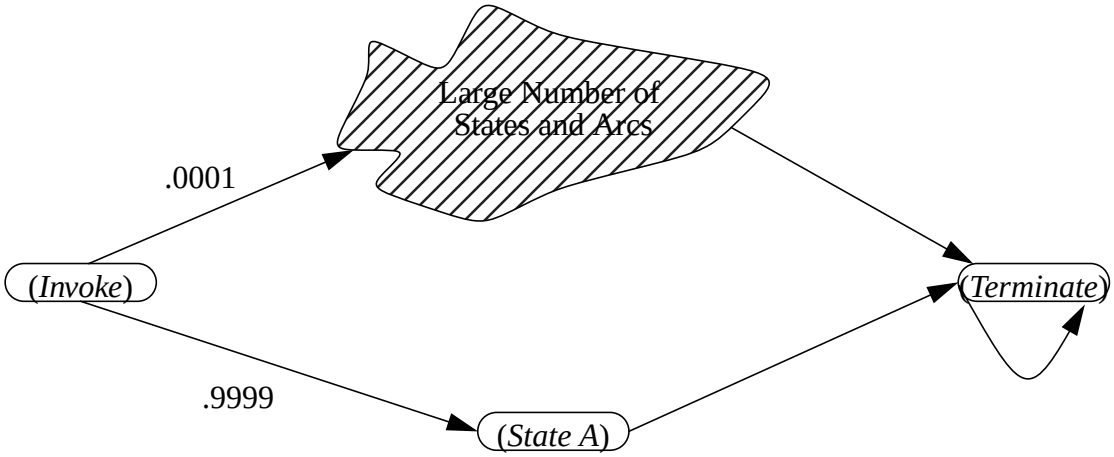


Figure 11: Euclidean Distance, Testing Chain A

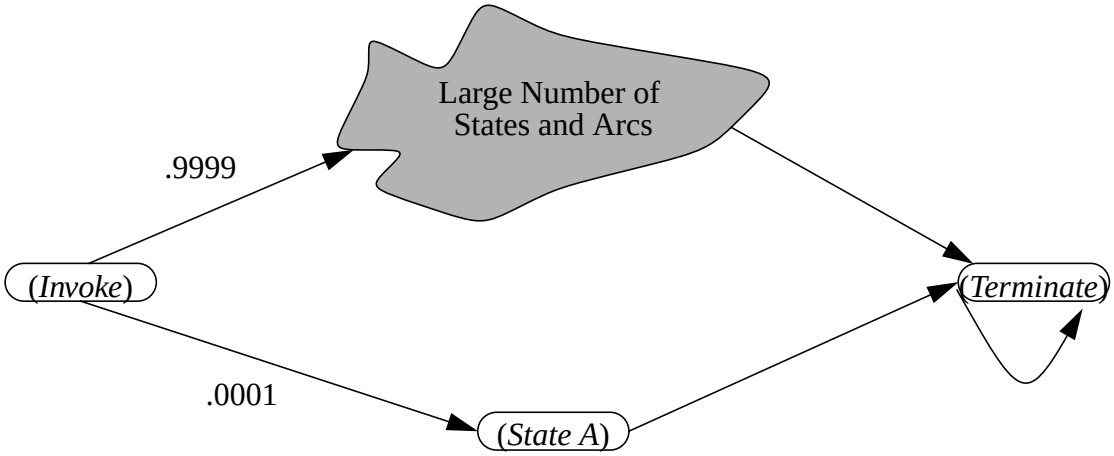


Figure 12: Euclidean Distance, Testing Chain B

5.1.2 The Kullback Discriminant

The Kullback discriminant [11] is the expected value of the log-likelihood ratio of two stochastic processes, i.e. $K(\mathbf{U}, \mathbf{T}) = \lim_{n \to \infty} \frac{1}{n} \left[\log \left(\frac{\Pr[X_0, X_1, \ldots, X_n | \mathbf{U}]}{\Pr[X_0, X_1, \ldots, X_n | \mathbf{T}]} \right) \right]$. In the specific case of comparing the usage chain to the testing chain, $X_0, X_1, \ldots, X_n$ is a sequence of length n generated by the usage chain and $K(\mathbf{U}, \mathbf{T}) = \sum_{i=1}^{u} \pi(i) \sum_{j=1}^{u} u_{i,j} \log \left(\frac{u_{i,j}}{t_{i,j}} \right)$. A problem arises in the computation of the discriminant when one or more arcs in the usage chain have not been covered in the testing chain. When some arcs have not been covered in the usage model, there exist state transitions that are possible in the usage chain but impossible in the testing chain. As the length of the sequence generated by the usage chain approaches infinity, the probability of one of the transitions not covered in the testing chain appearing in the sequence approaches one. Therefore, in the case where one or more arcs in the usage chain are not covered in the testing chain, the probability of the sequence generated by the usage chain being also generated by the testing chain approaches zero. This leads to a division by zero in the discriminant calculation. Therefore, the discriminant is not defined unless all arcs in the usage chain have been covered during testing.

The discriminant may be approximated by treating the probability of an arbitrarily long sequence generated from the usage chain being also generated from the testing chain, $\Pr[X_0, X_1, \ldots, X_n | \mathrm{T}]$, as being equal to ς, where ς is a positive value arbitrarily close but

not equal to zero. Setting $\Pr[X_0, X_1, \ldots, X_n|\mathrm{T}] = \varsigma$ in the discriminant calculation may be accomplished by setting the transition probabilities in the testing chain of arcs not covered during testing to ε, a positive value arbitrarily close to but not equal to zero. Setting $\Pr[X_0, X_1, \ldots, X_n|\mathrm{T}] = \varsigma$ in the discriminant calculation does not result in the true value of the discriminant; however, it may be better to calculate an approximate discriminant defined as $\tilde{\mathrm{K}}(\mathbf{U}, \mathbf{T}) = \sum_{i=1}^{\mathrm{u}} \pi(i) \sum_{j=1}^{\mathrm{u}} \mathrm{u}_{i,j} \log\left(\frac{\mathrm{u}_{i,j}}{\varepsilon - \varepsilon(\mathrm{sgn}(\mathrm{t}_{i,j})) + \mathrm{t}_{i,j}}\right)$, where $\mathrm{sgn}(x)$ is a function that returns zero if $x = 0$, one if $x > 0$, and negative one if $x < 0$, than nothing at all.

Theorem 5.1 If all arcs have been covered during testing, $\tilde{\mathrm{K}} = \mathrm{K}$.

Proof: If arc *(i,j)* has been covered in testing, $\mathrm{t}_{i,j} > 0$. Since all arcs *(i,j)* (where $\mathrm{u}_{i,j} > 0$) have been covered in testing, it is the case that all $\mathrm{t}_{i,j} > 0$ (where $\mathrm{u}_{i,j} > 0$). If $\mathrm{t}_{i,j} > 0$, $\varepsilon - \varepsilon(\mathrm{sgn}(\mathrm{t}_{i,j})) + \mathrm{t}_{i,j} = \mathrm{t}_{i,j}$. Therefore if all arcs have been covered during testing

$$\tilde{\mathrm{K}}(\mathbf{U}, \mathbf{T}) = \sum_{i=1}^{\mathrm{u}} \pi(i) \sum_{j=1}^{\mathrm{u}} \mathrm{u}_{i,j} \log\left(\frac{\mathrm{u}_{i,j}}{\mathrm{t}_{i,j}}\right) = \mathrm{K}(\mathbf{U}, \mathbf{T}). \square$$

Example 5.1 Discriminant Calculation With Incomplete Arc Coverage.

Figure 13 illustrates $\tilde{K}$. Note that the discriminant was calculated immediately.

5.2 Convergence of Testing Chain to Usage Chain

Neither the Kullback discriminant nor the Euclidean distance directly check whether the testing chain has followed a testing activity to converge to the usage chain. They simply provide a number used by the testing engineer to assess the degree to which the testing chain is currently in some sense equal to the usage chain.

The behavior of the Kullback discriminant and Euclidean distance over a series of tests have been used as heuristics to aid in the stopping decision. The testing chain is assumed to have converged, and hence further testing is likely to yield only modest amounts of new information, if the Kullback discriminant or Euclidean distance has changed very little over the last n test cases. A formal method for evaluating the convergence of the testing chain will be presented in the following sections.

Through the application of statistical sampling theory it is possible to compute the probability that the testing chain will remain essentially unchanged if more test cases are run. When estimating the mean of a population through sampling it is possible to estimate the variance of the sample mean. Given a sample of size n, the variance of the sample mean provides information on how the sample mean might vary from sample to sample. If the variance of the sample mean is small, repeated drawings of samples of size n are likely to yield the same sample mean.

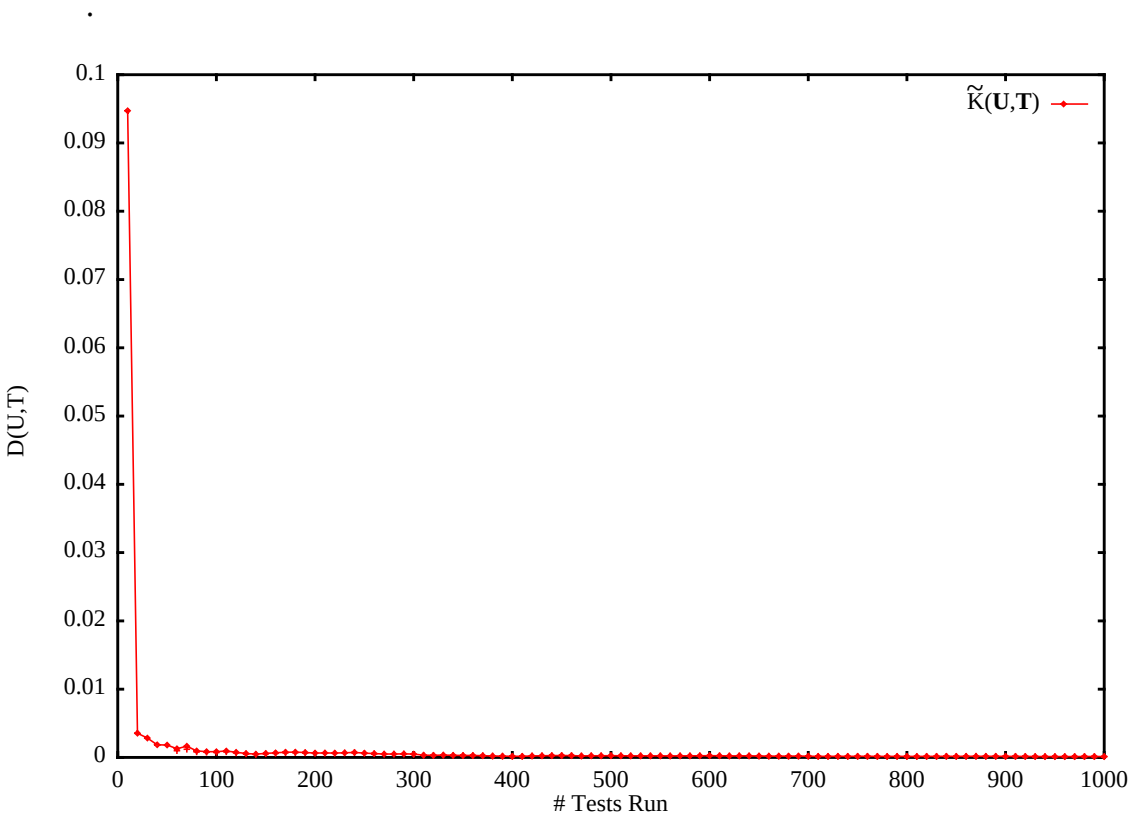


Figure 13: NAS Model, Approximate D(U,T)

It is necessary to characterize the construction of the testing chain as a sampling problem. The statistics of interest to be estimated through sampling from the usage chain are the long run arc occupancies, that is, the proportion of time over the long run spent taking each arc in the model. The long run arc occupancies were chosen for two reasons.

- The long run arc occupancies define a unique Markov chain. This implies that if the sample size is large enough that the long run arc occupancies as computed from the testing record have converged to their respective values as computed from the usage chain, then the single step transition probabilities that define the testing chain have also converged to their respective values in the usage chain.
- By focusing on the estimation of the long run arc occupancies, the problem may be treated as a case of cluster sampling for the estimation of proportions. In sampling terminology, the sampling unit is a single arc, the arcs are sampled in clusters, where each cluster is a sequence, and the proportion of each arc in relation to all other arcs is estimated.

Theorem 5.2 A matrix of long run arc occupancies $\underline{\pi}$ defines a unique Markov chain.

Proof: Let **U** and **V** be Markov chains with positive stationary distributions. Let **U** and **V** have the same long run arc occupancies, i.e., $\pi(\underline{U}) = \pi(\underline{V})$. Because

$\sum_{j=1}^{u} u_{i,j} = 1 = \sum_{j=1}^{v} v_{i,j}$ and $\pi(i, j) = u_{i,j}\pi(U, i) = v_{i,j}\pi(V, i)$, the following equalities hold:

$$u_{i,j} = \frac{\pi(i,j)}{\pi(U,i)} = \frac{\pi(i,j)}{\pi(U,i)\sum_{j=1}^{u} u_{i,j}} = \frac{\pi(i,j)}{\sum_{j=1}^{u} u_{i,j}\pi(U,i)} = \frac{\pi(i,j)}{\sum_{j=1}^{u} \pi(i,j)} =$$

$$\frac{\pi(i,j)}{\sum_{j=1}^{v} v_{i,j}\pi(V,i)} = \frac{\pi(i,j)}{\pi(V,i)\sum_{j=1}^{v} v_{i,j}} = \frac{\pi(i,j)}{\pi(V,i)} = v_{i,j}$$

The two Markov chains are equal. Therefore, the matrix of long run arc occupancies defines a unique Markov chain. □

5.2.1 Estimation of Long Run Arc Occupancies

As described in chapter 3, let **S** be the success matrix and **F** be the failure matrix generated from testing based on usage model U. The estimated long run arc occupancy, $\hat{\pi}(i, j)$, can be computed for each arc (i, j) as [4]:

$$\hat{\pi}(i, j) = \frac{s_{i,j} + f_{i,j}}{\sum_{k=1}^{u}\sum_{l=1}^{u} s_{k,l} + f_{k,l}}$$

Consider n sequences from usage model U. Let $X_k(a, b)$, for $k = 1, \ldots, n$, be the number of occurrences of arc *(a,b)* in the *k*th sequence. Let m_k be the length of the kth

sequence. Let $\overline{m}$ be the expected sequence length of U. Given this information, the estimated variance of $\hat{\pi}(i, j)$ is computed as:

$$\hat{\mathrm{V}}\mathrm{ar}(\hat{\pi}(i, j)) =$$

$$\left(\frac{1}{n\overline{m}^2}\right)\left(\frac{\sum_{k=1}^{n}\sum_{a=1}^{\mathrm{u}}\sum_{b=1}^{\mathrm{u}}(\mathrm{X}_k(a, b))^2 + 2\hat{\pi}(i, j)\sum_{k=1}^{n}\sum_{a=1}^{\mathrm{u}}\sum_{b=1}^{\mathrm{u}}\mathrm{X}_k(a, b)m_k + \sum_{k=1}^{n}m_k^2}{n-1}\right)$$

Because the distribution of sample means is approximately normal, given the estimated sample mean and variance, it is possible to calculate the probability that the sample mean will fall in a certain range by converting to a standard normal distribution and using the standard normal cumulative distribution function (CDF).

$\hat{\pi}(i, j)$ will be considered to have converged to $\pi(\underline{\mathrm{U}}, i, j)$ if $P[|\pi(i, j) - \hat{\pi}(i, j)| \leq \varepsilon_{i, j}] > p_{i, j}$, where $\varepsilon_{i, j}$ is a parameter indicating the tolerance to which $\hat{\pi}(i, j)$ must converge.

$\Pr[|\pi(i, j) - \hat{\pi}(i, j)| \leq \varepsilon_{i, j}]$ may be calculated as the area under the curve of the standard normal CDF in the interval $\left[\frac{(\pi(i, j) - \varepsilon_{i, j}) - \hat{\pi}(i, j)}{\sqrt{\mathrm{Var}(\hat{\pi}(i, j))}}, \frac{(\pi(i, j) + \varepsilon_{i, j}) - \hat{\pi}(i, j)}{\sqrt{\mathrm{Var}(\hat{\pi}(i, j))}}\right]$. Arc (i, j) in the testing chain is said to be *approximately equal* to the corresponding arc in the usage chain if $|\pi(i, j) - \hat{\pi}(i, j)| \leq \varepsilon_{i, j}$.

The testing chain is considered to have converged if $\Pr[\forall i \forall j(|\pi(i, j) - \hat{\pi}(i, j)| \leq \varepsilon_{i, j})] > p$, i.e. the probability that all the long run arc occupancies of the testing chain will be approximately equal to the long run arc occupancies of the usage chain is greater than p, if an equal number of tests were to be run again. Henceforth, the testing chain will be termed *approximately equal* to the usage chain if all of the long run arc occupancies as estimated from the testing record are approximately equal to the long run arc occupancies of the usage chain. Note that the probabilities of the long run arc occupancies as estimated by the testing record being approximately equal to the long run arc occupancies of the usage chain are not independent, as will be addressed next.

5.2.2 Estimation of the Probability of Approximate Equality

Because of the lack of independence of the long run arc occupancies, the probability of the estimated long run arc occupancies all being approximately equal to the actual long run arc probabilities cannot be analytically computed in a simple manner. However, this probability may be estimated through simulation.

The simulation is performed by repeating iterations of generating a fixed number, n, of sequences, updating the testing chain, and checking to see if the resulting testing chain is approximately equal to the usage chain. The probability of the testing chain being approximately equal to the usage chain after the generation of n sequences is estimated as

the proportion of times that the testing chain and usage chain were approximately equal to the total number of times the generation of *n* sequences was simulated.

In more detail, given **S** and **F** as the initial value of the testing record, *n*, the number of sequences to generate, U, the usage chain, and *j*, the number of simulation iterations, the simulation proceeds as follows:

```
Count_Of_Equal = 0
for p=1 to j do
     t_temp = S + F
     for q = 1 to n do
          s = Generate_Sequence(U)
          Update t_temp with s
     end for
     if (Estimate_Long_Run_Arc_Occ(t_temp) is approximately
          equal to Calc_Long_Run_Arc_Occ(U)) then
          Count_Of_Equal = Count_Of_Equal + 1
     end if
end for
Probability = Count_Of_Equal/j
```

As *j* becomes sufficiently large, `Probability` will approach the true probability of the testing chain being approximately equal to the usage chain after the generation of *n* sequences.

Note that in the simulation the testing record is initialized to an arbitrary state. This means that the simulation may be used to estimate the probability of the testing chain being approximately equal to the usage chain after the generation of *n* additional sequences given that the testing represented by the initial test record has already been per-

formed. Therefore, the same simulation algorithm may be used to compute both the probability of the testing chain being approximately equal to the usage chain after running n sequences given that no testing has been performed, and to compute the probability of approximate equality given that some specific testing has already been performed. To compute the probability of approximate equality given that some specific testing has already been performed, the simulation will be initialized with a non-empty testing record.

Example 5.2 Probability of Approximate Equality With a Fixed Testing Record

The graph in Figure 14 shows the probability of approximate equality of the testing chain, **T**, and the usage chain, **U**. During the simulation the testing record was initialized with an empty testing record, i.e., the same base of testing experience was used to compute the probability of approximate equality after running $n = 1 \ldots 1000$ additional tests. In this example the $\varepsilon_{i,j}$ for each arc (i, j) was set to 20% of the actual long run occupancy of the arc. Thus, the testing chain is considered to be approximately equal to the usage chain if all long run arc occupancies as estimated from the testing record are within 20% of their actual values. Values were computed every ten test cases

Given that no specific prior testing was performed, when testing from the NAS model there is approximately a 50% chance of the testing chain being approximately equal to the usage chain after running 190 test cases. After running 1000 test cases there is a 99.5% chance of the testing chain being approximately equal to the usage chain.

.

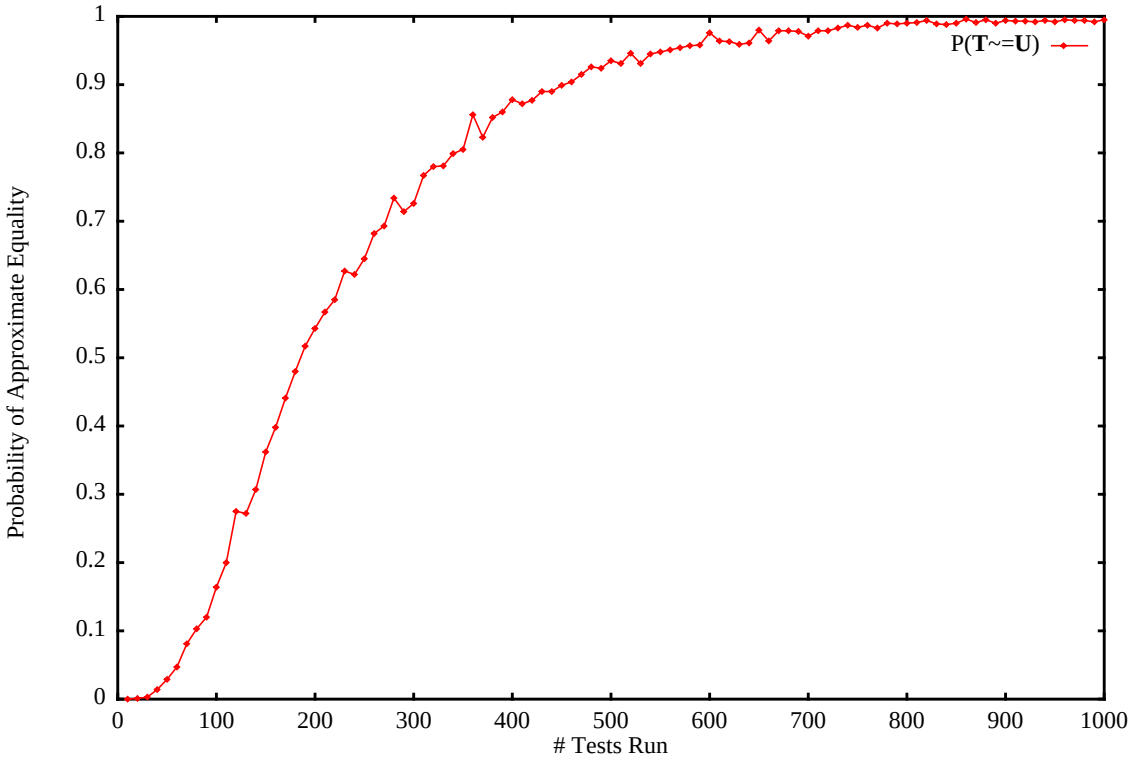


Figure 14: NAS Model, Convergence of Testing Chain to Usage Chain

In Figure 14 the graph of the probability of approximate equality given a fixed base of testing experience seems to be smoothly increasing, with a number of local rough points. These rough points in the graph will disappear if the number of iterations in the simulation is increased. Given a fixed base of testing experience, i.e., the simulation is initialized every time with the same testing record, the probability of approximate equality increases monotonically as the number of tests run increases.

Theorem 5.3 If $n_1 < n_2$, then

$$\Pr[(\pi(\underline{U}, i, j) = \hat{\pi}(i, j))|<\mathbf{S}_1, \mathbf{F}_1, n_1>] \leq \Pr[(\pi(\underline{U}, i, j) = \hat{\pi}(i, j))|<\mathbf{S}_2, \mathbf{F}_2, n_2>]$$

Proof: It is given that $\mathrm{Var}(\hat{\pi}(i, j)) = \dfrac{\pi(\underline{U}, i, j)(1 - \pi(\underline{U}, i, j))}{n}$ [4]. Since $\pi(\underline{U}, i, j)(1 - \pi(\underline{U}, i, j)) > 0$ is a constant, if $n_1 < n_2$ then $\dfrac{\pi(\underline{U}, i, j)(1 - \pi(\underline{U}, i, j))}{n_1} > \dfrac{\pi(\underline{U}, i, j)(1 - \pi(\underline{U}, i, j))}{n_2}$, i.e., $\mathrm{Var}(\hat{\pi}(i, j))$ decreases as the number of tests run increases.

$\hat{\pi}(i, j)$ is normally distributed with expected value $\mathrm{E}(\hat{\pi}(i, j)) = \pi(\underline{U}, i, j)$. Therefore, if $\mathrm{Var}(\hat{\pi}_2(i, j)) < \mathrm{Var}(\hat{\pi}_1(i, j))$, where $\hat{\pi}_1(i, j)$ is based on $<\mathbf{S}_1, \mathbf{F}_1, n_1>$ and $\hat{\pi}_2(i, j)$ is based on $<\mathbf{S}_2, \mathbf{F}_2, n_2>$, then

$\Pr[\pi(\underline{U}, i, j) = \hat{\pi}_1(i, j)] \leq \Pr[\pi_2(\underline{U}, i, j) = \hat{\pi}(i, j)]$.

Since $\mathrm{Var}(\hat{\pi}_2(i, j)) < \mathrm{Var}(\hat{\pi}_1(i, j))$ if $n_1 < n_2$ and

$\Pr[\pi(\underline{U}, i, j) = \hat{\pi}_1(i, j)] \leq \Pr[\pi_2(\underline{U}, i, j) = \hat{\pi}(i, j)]$ if $\mathrm{Var}(\hat{\pi}_2(i, j)) < \mathrm{Var}(\hat{\pi}_1(i, j))$,

$\Pr[(\pi(\underline{U}, i, j) = \hat{\pi}(i, j))|<\mathbf{S}_1, \mathbf{F}_1, n_1>] \leq \Pr[(\pi(\underline{U}, i, j) = \hat{\pi}(i, j))|<\mathbf{S}_2, \mathbf{F}_2, n_2>]$ if

$n_1 < n_2$.□

This proves, given a fixed base of testing experience, that $\Pr[\pi(\underline{U}, i, j) = \hat{\pi}(i, j)]$ increases as the number of additional tests run increases. Because the probability of $\pi(\underline{U}, i, j) = \hat{\pi}(i, j)$ for all arcs (*i,j*) increases as *n* increases and each $\hat{\pi}(i, j)$ is normally distributed, the probability of approximate equality increases as *n* increases. Therefore, any drops in the graph of the probability of approximate equality given a fixed base of testing experience as *n* increases are due to an insufficient number of iterations in the simulation (samples) used to estimate the probability of approximate equality.

Note that there are two basic ways of using the probability of approximate equality, (1) calculating the probability of approximate equality given a fixed prior testing record and varying the number of additional tests to run, or (2) calculating the probability of approximate equality given a fixed number of additional tests to run based on a successively updated testing record. Discussion up to this point has centered around the calculation of the probability of approximate probability given a fixed prior testing record and varying the number of additional tests to run. The probability of approximate equality is monotonically increasing in this case.

The next example is concerned with the second use of the probability of approximate equality. The testing record used in the estimation of the probability of approximate equality is updated after each executed test case. Given this updated base of testing experience, the probability of approximate equality after running some number of additional tests is estimated. As will be seen, the probability of approximate equality is not monotonic when used in this fashion.

Example 5.3 Probability of Approximate Equality With an Evolving Testing Record

Figure 15 displays the probability that the testing chain will be approximately equal to the usage chain after running n_1 additional test cases, given that n specific test cases have already been run. The probabilities presented illustrate the case for $n_1 = 10$ and n going from 0 to 1000.The testing record used in the simulation of the probability of approximate equality is updated after each test case is run, i.e., the base of testing experience is evolving over time.

The graph of Figure 15 does not follow an orderly curve. Until the testing record has stabilized to a certain degree, the probability of approximate equality of the testing chain and usage chain after running the next ten test cases given the updated testing experience is quite sensitive to the current state of the testing record. While testing chain **T** will eventually converge to usage chain **U**, **T** does not converge monotonically to **U**.

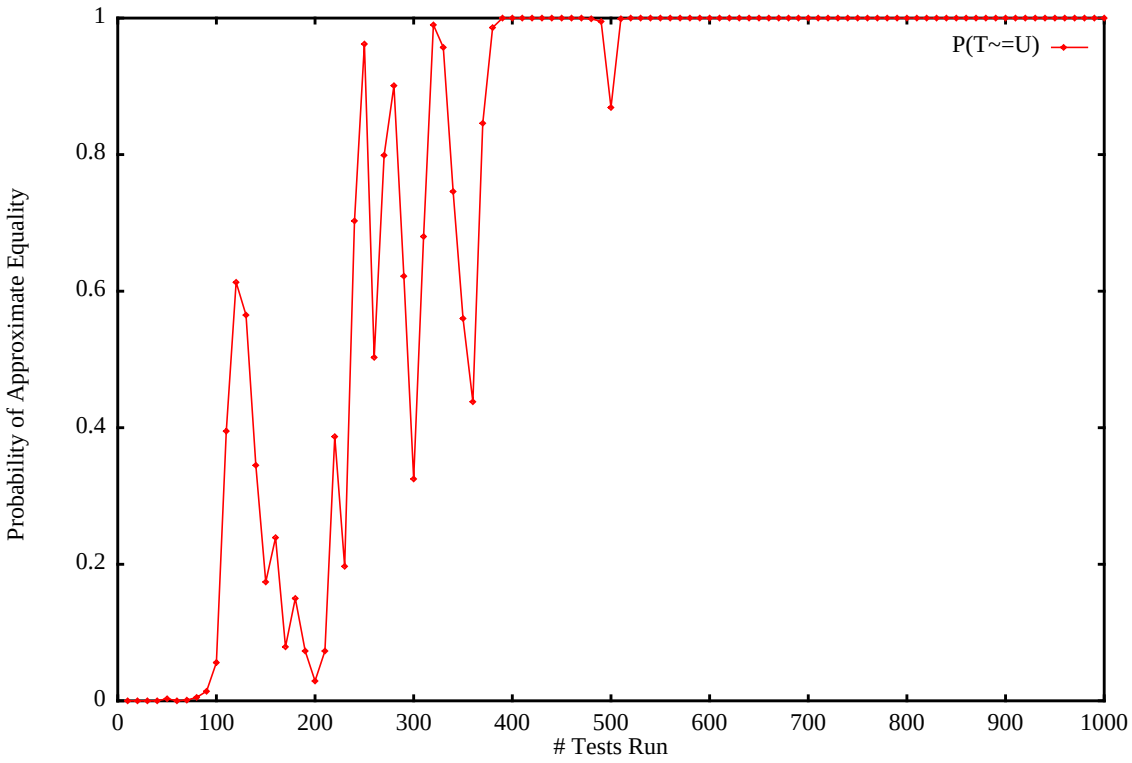


Figure 15: NAS Model, Convergence of Testing Chain to Usage Chain, Successively Updated Testing Record

In more detail, let $f(\langle \mathbf{S}_a, \mathbf{F}_a, a \rangle)$ be a function that computes the testing chain from a given testing record, $\mathbf{T}_n = f(\langle \mathbf{S}_n, \mathbf{F}_n, n \rangle)$, and $\mathbf{T}_{n+n_1} = f(\langle \mathbf{S}_n + \mathbf{S}_{n_1}, \mathbf{F}_n + \mathbf{F}_{n_1}, n + n_1 \rangle)$, where $\mathbf{S}_{n_1}$ and $\mathbf{F}_{n_1}$, represent testing experience gained from running n_1 additional test cases. If $\mathbf{T}_{n+n_1}$ is less similar to $\mathbf{U}$ than $\mathbf{T}_n$ then the probability of approximate equality after n_2 additional tests given $\langle \mathbf{S}_n + \mathbf{S}_{n_1}, \mathbf{F}_n + \mathbf{F}_{n_1}, n + n_1 \rangle$ as the base of testing experience will be less than the probability of approximate equality after n_2 additional tests given $\langle \mathbf{S}_n, \mathbf{F}_n, n \rangle$ as the base of testing experience. In other words, in this case the probability of approximate equality after running $(n + n_1) + n_2$ tests will be less than the probability of approximate equality after running $(n) + n_2$ tests. Therefore, the probability of approximate equality given an evolving testing record is not monotonically increasing.

A planning analysis using figure 14 might suggest that 700 test cases will need to be run before the probability of approximate equality nears 0.95. Figure 15 suggests that given the actual testing performed, the probability of approximate equality consistently exceeds 0.95 after 500 test cases have been run. Therefore, if the probability of approximate equality being greater than 0.95 was used as a stopping criterion, it may make sense to stop testing after 500 test cases have been run.

5.2.3 Setting the Arc Sensitivities

Each arc in the model has a distinct ε value. This allows the tester to adjust the sensitivity of the probability of convergence calculation in accordance with the importance of the arc from a testing standpoint. For example, when testing a new release of a software system it is often the case that new or modified code makes up only a small portion of the system. A great deal of the functionality may have been tested on previous releases. In this case it may not be necessary to test the old functionality to the same extent as the new functionality. This can be reflected in the probability of convergence calculation by setting the ε values of all arcs involving old functionality to relatively large values. The ε values associated with all arcs involving new functionality can be set to relatively small values. In general, the smaller ε, the more visits to that arc required before it is considered to have converged.

If an arc (i, j) is to be ignored in the consideration of approximate equality of the testing chain and usage chain, set $\varepsilon_{i,j} = 1$. Since both $\pi(i, j)$ and $\hat{\pi}(i, j)$ are probabilities, it is always the case that $|\pi(i, j) - \hat{\pi}(i, j)| \leq 1$. Therefore, if $\varepsilon_{i,j} = 1$ arc (i, j) in the testing chain will always be approximately equal to arc (i, j) in the usage chain.

5.3 Conclusions about Stopping Criteria

Because the Kullback discriminant and the probability of approximate equality are measuring two fundamentally different things, there is no direct correspondence between them. The Kullback discriminant measures the similarity of two stochastic processes, in this case the similarity of the usage chain to the testing chain. The probability of approximate equality gives the probability that the testing chain will be approximately equal to the usage chain after some number of additional tests have been run. Because the probability of approximate equality is the probability of the occurrence of an event and the Kullback discriminant is a measure of the similarity of two stochastic processes, not a probability, there is no direct correspondence between these stopping criteria.

In more detail, the Kullback discriminant is concerned with the specific current state of the testing chain. It provides no quantifiable information about the future behavior of the testing chain, i.e., the Kullback discriminant does not provide any quantifiable indication of whether the testing chain will significantly change during future testing (that information is vested in the testing record). The probability of approximate equality is based entirely on assessing the possible future states of the testing record. It provides no direct information about the current state of the testing record.

The probability of approximate equality and the Kullback discriminant may be used in conjunction. Once the testing chain has converged, the tester may wish to get some indication of the similarity of the current testing experience and the expected use of the software. The Kullback discriminant may be used to asses the degree that the current testing experi-

ence matches the expected use of the software, and the probability of approximate equality may be used to asses the chance that the testing record will significantly change in the future.

Given the availability of $\tilde{K}$ and the probability of approximate equality, it is recommended that use of the Euclidean distance as a stopping criterion be discontinued. Both $\tilde{K}$ and the probability of approximate equality are specifically concerned with the comparison of the behavior of two stochastic processes whereas the Euclidean distance is not.

Chapter 6

Summary and Conclusions

6.1 Recommended Changes in Engineering Practices

Based on this research, the following changes in testing practices are suggested.

- Estimate the quantitative analytical results through simulation when necessary. As industrial models increase in size, the time needed to calculate analytical results from the model will impede work flow in the use of statistical testing. Therefore distributed solutions for the calculation of these results will become desirable. Distributed solutions for the simulation of the quantitative analytical results listed in chapter 2 are easily realized.
- Use the correct calculation of the probability of an arc appearing in a sequence.
- Use the variances of the random variables associated with analytical results listed in chapter 2. Because the variances of these random variables can be quite large, the probability is high of observing an instance of the random variable during testing that

is significantly different than the random variable's expected value. Therefore a testing organization is incurring risk by ignoring these variances.

- Use extensive simulation in test planning.
- Make use of arc-based reliability estimators. A testing organization now has considerable ability to tailor the reliability estimation to the situation in order to make testing more efficient. If different reliability estimators yield radically different results when presented with the same testing data, further investigation of the testing problem is indicated.
- Use the arc-based reliability models to take advantage of pre-test reliability information. Testing savings can be realized if accurate pre-test reliability information is available.
- Use partition testing techniques in conjunction with usage models and arc-based reliability models to increase testing efficiency.
- Do not use the Euclidean distance as a stopping criterion. More suitable stopping criteria are now available, such as the approximation of the Kullback discriminant prior to full arc coverage and the probability of approximate equality.
- Use the probability of approximate equality in conjunction with the approximation of the Kullback discriminant as stopping criteria.

6.2 Future Work

The following issues are worthy of further study:

- It is possible to estimate the variance of the single use reliability based solely on the variance of the arc failure rates. As presented in chapter 3, the variance of the single use reliability is affected by both the variance of randomly generated sequences and on the variance of the arc failure rates.
- Further research into distributed solutions for the estimation or calculation of the quantitative analytical results listed in chapter 2 is needed. Some initial work in this area is presented in [3].
- Additional automatic partitioning schemes based on the framework discussed in chapter 4 should be investigated. Some possible schemes may take into account risk when designing the partition of the test cases.
- A formal relationship between the similarity of the testing chain and the usage chain and the estimated reliability should be established. Field evidence shows that a high degree of similarity between the testing chain and the usage chain indicates that the reliability estimated from the testing experience is accurate. However, a more formal relationship is needed.
- The distributions of various random variables based on the usage model (sequence length, number of sequences to cover all states or arcs, etc.) should be studied. Knowledge of the distribution underlying these random variables will allow for increased accuracy in test planning and model validation.

Bibliography

[1] K. Agrawal and J.A. Whittaker, “Experience in Applying Statistical Testing to a Real-Time, Embedded Software System”, *Proceedings of the Pacific Northwest Software Quality Conference*, 1994.

[2] R. Ash, *Information Theory*, John Wiley and Sons, New York, 1966.

[3] V. Chidamba, “Investigation of Parallel Processing to Facilitate Speedup of Markov Chain Simulations and Associated Statistic Calculations”, *University of Tennessee Department of Computer Science Technical Report*, ut-cs-99-427, 1999.

[4] W.G. Cochran, *Sampling Techniques*, John Wiley and Sons, New York, 1977.

[5] M.H. DeGroot, *Probability and Statistics*, Addison-Wesley, 1989.

[6] J. Gaffney et. al., *Software Measurement Guidebook*, International Thomson Computer Press, 1995.

[7] W.J. Gutjahr, “Importance Sampling of Test Cases in Markovian Software Usage Models”, Probability in the Engineering and Informational Sciences, Vol 11, 1997, pp 19-36.

[8] D. Hamlet and R. Taylor, “Partition Testing Does not Inspire Confidence”, *IEEE Transactions on Software Engineering*, Vol 16, No 1, December 1990, pp 1402-1411.

[9] N.L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, John Wiley and Sons, New York, 1995.

[10] J.G. Kemeny and J.L. Snell, *Finite Markov Chains*, D. Van Nostrand Company, Inc., 1960.

[11] S. Kullback, *Information Theory and Statistics*, John Wiley and Sons, New York, 1958.

[12] K.W. Miller, et. al., “Estimating the Probability of Failure When Testing Reveals No Failures”, *IEEE Transactions on Software Engineering*, Vol 18, January 1992, pp 33-42.

[13] H.D. Mills, “Certifying the Correctness of Software”, *Proceedings of the Hawaii International Conference on System Sciences, Vol II, Software Technology,* 1992, pp. 373-281.

[14] V.N. Nair, D.A. James, W.K. Ehrlich, and J. Zevallos, “Statistical Issues in Assessing Software Testing Strategies”, *Statistica Sinica*, January 1998.

[15] J.H. Poore, H.D. Mills, and D. Mutchler, “Planning and Certifying Software System Reliability”, IEEE Software, January 1993, pp 88-99.

[16] 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.

[17] G.H. Walton, *Generating Transition Probabilities for Markov Chain Usage Models*, Ph.D. Dissertation, Department of Compurter Science, University of Tennessee, Knoxville, 1995.

[18] G.H. Walton, J.H. Poore, and C.J. Trammell, “Statistical Testing of Software Based on a Usage Model”, *Software Practice and Experience*, January 1995, pp 97-108.

[19] E.J. Weyuker and B. Jeng, “Analyzing Partition Testing Strategies”, *IEEE Transactions on Software Eng*ineering, July 1991, pp 97-108.

[20] J.A. Whittaker, *Markov Chain Techniques for Software Testing and Reliability Analysis*, Ph.D. Dissertation, Department of Computer Science, University of Tennessee, Knoxville, 1992.

[21] J.A. Whittaker and J.H. Poore, “Markov Analysis of Software Specifications”, *ACM Transactions on Software Engineering and Methodology,* January 1993, pp 93-106.

[22] J.A. Whittaker and M.G. Thomason, “A Markov Chain Model for Statistical Software Testing”, *IEEE Transactions on Software Engineering*, October 1994, pp 812-824.

Appendix

Table 15: NAS Model

Name of Model: NAS					
General Statistics					
States	Arcs	Density	Entropy	Paths	N^3
17	29	0.100	0.587	$14x10^9$	4,913
Expected Sequence Length 56.3, Variance 1,622, Simulation Cost 197 sequences or 10,987-11,145 transitions.					
State Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (2)	1.0	0	0	0	
Greatest (3)	191.0	46,121	486	92,330-93172	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (17)	1.0	0	0	0	
Greatest (3)	4.3	14.5	296	71,805-72,405	
Arc Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,2)	1.0	0	0	0	
Greatest (14,11)	232.0	48,144	344	79,288-80,148	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (16,17)	1.0	0	0	0	
Greatest (14,11)	4.7	17.5	303	79,938-80,598	
Long run and stationary probability					
	Analytical	Simulation		Analytical	Simulation
State ID	Probability	Sequences	Arc ID	Probability	Sequences
Least (3)	0.00523	92,751	Least (14,11)	0.00470	86,394
Greatest (11)	0.25092	4,705	Greatest (11,15)	0.17506	6,318

Table 16: CDU_Inc2 Model

Name of Model: CDU_Inc2					
General Statistics					
States	Arcs	Density	Entropy	Paths	N^3
51	300	.115	.504	$5x10^6$	132,651
Expected Sequence Length 43.1, Variance 1779, Simulation Cost 368 sequences or 15,775 - 15,940 transitions.					
State Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (2)	1.0	0.0	0.0	0	
Greatest (44)	520,268	$2.7x10^{11}$	384	$1.9x10^8$-$2.0x10^8$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (1)	1.0	0.0	0	0	
Greatest (44)	11,800	$1.39x10^8$	385	$1.9x10^8$-$2.0x10^8$	
Arc Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,2)	1.0	0.0	0.0	0	
Greatest (44,41)	$8.93x10^6$	$8.0x10^{13}$	385	$3.41x10^9$-$3.44x10^9$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,2)	1.0	0.0	0	0	
Greatest (44,41)	202,538	$4.1x10^{10}$	385	$3.33x10^9$-$3.36x10^9$	
Long run and stationary probability					
	Analytical	Simulation		Analytical	Simulation
State ID	Probability	Sequences	Arc ID	Probability	Sequences
Least (44)	~0	$2.94x10^8$	(44,41)	~0	$3.4x10^9$
Greatest (15)	.36267	21,635	(15,15)	.34270	23,046

Table 17: Certify_Inc3A Model

Name of Model: Certify_Inc3A					
General Statistics					
States	Arcs	Density	Entropy	Paths	N^3
56	151	.0481	1.204	$4.76x10^{31}$	175,616
Expected Sequence Length 86.4, Variance 7277, Simulation Cost 375 sequences or 32,186-32,520 transitions.					
State Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (2)	7.3	201.6	1,471	10,645-10,700	
Greatest (50)	2054.2	$4.13x10^6$	377	768,982-776,951	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (56)	1.0	0.0	0	0	
Greatest (50)	24.4	570.4	369	772,383-780,370	
Arc Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (2,2)	35.5	1,547.0	471	16,653-16,807	
Greatest (50,49)	3,917.9	$1.52x10^7$	380	$1.48x10^6$-$1.5x10^6$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (2,2)	1.0	0.2	77	2057-3409	
Greatest (50,49)	45.3	2088.8	391	$1.38x10^6$-$1.68x10^6$	
Long run and stationary probability					
	Analytical	Simulation		Analytical	Simulation
State ID	Probability	Sequences	Arc ID	Probability	Sequences
Least (50)	.00064	963,139	(50,46)	.00032	$1.32x10^6$
Greatest (2)	.21700	6,426	(16,16)	.10850	37,155

Table 18: msds Model

Name of Model: msds					
General Statistics					
States	Arcs	Density	Entropy	Paths	N^3
130	565	.0334	.993	$2.97x10^{20}$	$2.2x10^6$
Expected Sequence Length 67.5, Variance 4867.8, Simulation Cost 411 sequences or 27,559-27,832 transitions.					
State Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (19)	1.0	0.0	1	0	
Greatest (16)	8707.4	$7.55x10^7$	383	$3.31x10^6$-$3.35x10^6$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (130)	1.0	0.0	0	0	
Greatest (78)	128.3	16324	382	$3.28x10^6$-$3.32x10^6$	
Arc Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,19)	1.0	0.0	1	0	
Greatest (33,33)	853,473	$7.28x10^{11}$	385	$3.26x10^8$-$3.3x10^8$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,19)	1.0	0.0	0	0	
Greatest (33,33)	12,457	$1.55x10^8$	385	$3.21x10^8$-$3.25x10^8$	
Long run and stationary probability					
	Analytical	Simulation		Analytical	Simulation
State ID	Probability	Sequences	Arc ID	Probability	Sequences
Least (16)	.00012	$3.43x10^6$	(16,16)	~0	$3.31x10^8$
Greatest (106)	.16106	2,511	(74,106)	.02879	10,699

Table 19: Tucson2 Model

Name of Model: Tucson2					
General Statistics					
States	Arcs	Density	Entropy	Paths	N^3
392	573	.00373	.353	$1.05x10^{85}$	$6.02x10^7$
Expected Sequence Length 799.3, Variance $6.54x10^6$, Simulation Cost 3,932 sequences or $3.14x10^6$-$3.15x10^6$ transitions.					
State Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (2)	1.0	0.0	0	0	
Greatest (262)	76,177	$6.48x10^9$	429	$3.25x10^7$-$3.28x10^7$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
State ID	Expectation	Variance	Sequences	Transitions	
Least (17)	1.0	0.0	0.0	0	
Greatest (3)	95.3	8,978.8	380	$2.85x10^7$-$2.94x10^7$	
Arc Statistics					
Mean first passage measured in transitions					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,2)	1.0	0.0	0	0	
Greatest (64,65)	117,399	$1.48x10^{10}$	414	$4.83x10^7$-$4.88x10^7$	
Mean first passage measured in sequences					
	Analytical		Simulation Costs		
Arc ID	Expectation	Variance	Sequences	Transitions	
Least (1,2)	1.0	0.0	0.0	0	
Greatest (64,67)	147.1	21479	382	$4.41x10^7$-$4.56x10^7$	
Long run and stationary probability					
	Analytical	Simulation		Analytical	Simulation
State ID	Probability	Sequences	Arc ID	Probability	Sequences
Least (262)	.00001	$3.57x10^7$	(64,65)	.00001	$4.84x10^7$
Greatest (387)	.21422	$2.06x10^6$	(387,388)	.21422	$2.06x10^6$

Vita

Kirk Sayre was born in Pittsburgh, Pennsylvania on April 15, 1970. He graduated from Moniteau High School in June of 1988. In 1992, he graduated from Bucknell University with a B.S. in Computer Science. In 1994 Mr. Sayre graduated from the American University with a M.S. in Computer Science. Kirk Sayre worked for Software Engineering Technology from 1994 to 1996.

Kirk Sayre was awarded the Doctor of Philosophy degree in Computer Science from the University of Tennessee in December of 1999.