

Partition Testing With Usage Models

Kirk Sayre
University of Tennessee
sayre@cs.utk.edu

J.H. Poore
University of Tennessee
poore@cs.utk.edu

Abstract

The fundamental statistical strategy of improving sampling efficiency through partitioning the population is applied to software testing. Usage models make it possible to apply this strategy to improve the efficiency of testing. The testing budget is allocated to the blocks of the partition, and the software is executed on the sample of uses drawn from each block or sub-population of potential uses. Usage models support many strategies for automated partitioning and generating test cases from the partitioned population. Two strategies are shown here with the efficiency gains demonstrated.

1. Statistical testing

Statistical testing is the comprehensive application of statistical science to the solution of software testing problems. The population of interest is all possible uses of the software under test, and this population must be represented in a form that facilitates posing and solving problems about the software and the testing process. The representation we use is the Markov chain usage model, which is described below. Further information about Markov chain usage models is presented in [4], which is reprinted in these conference proceedings.

The statistical technique presented here is the use of partitioning to improve sampling efficiency. There is a “partition testing” literature [2,6] in software engineering which varies somewhat in terminology and conclusions from the statistical literature on partition sampling [1,3]. In this paper we will follow the statistical terminology and apply it to two goals of software testing, the estimation of software reliability and failure detection potential.

All possible uses of the software under test can be represented by a usage model [5,7]. This model has the structure of a discrete parameter, finite state, time homogeneous Markov chain. States of use of the software are represented as states of the Markov chain and transitions from one state of use to another are represented by state transitions of the Markov chain. Each state transition has a fixed probability of occurrence. Usage models have two distinguished states, a single invocation state and a single termination state; an arc from the termination state back to the invocation state makes the

chain recurrent. Significant analytical results for usage models are readily available [8].

2. 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 paper 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 [1,3]. 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.

2.1. Basic results

The following are basic results of sampling theory [1]:

- 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, \dots, p_K , where p_i is the probability mass associated with block i .
- Estimated failure rates f_1, f_2, \dots, f_K , where f_i is the 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, \dots, c_K , are needed.

2.2. 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 :

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

2.3. 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 a_h , the number of failures observed during testing in block h , is recorded.

The overall failure rate of the software is estimated as

$f = \sum_{h=1}^K p_h f_h$, where $f_h = \frac{a_h}{n_h}$ is the estimated failure rate of block h .

2.4. Variance of estimate of overall failure rate

The variance of the estimate of the overall failure rate is

$$V(f) = \sum_{h=1}^K p_h^2 \left(\frac{f_h(1-f_h)}{n_h - 1} \right).$$

2.5. Probability of finding a failure

The estimated probability of finding at least one failure

during testing can be calculated. Given f_1, f_2, \dots, f_K , the estimated failure rates of the partition blocks, and n_1, n_2, \dots, n_K , the number of tests allocated to the blocks, the estimated probability of finding at least one failure

during testing is $1 - \prod_{h=1}^K (1 - f_h)^{n_h}$.

2.6. Expected number of failures found

The expected number of failures to be found during testing can also be calculated. Given f_1, f_2, \dots, f_K and n_1, n_2, \dots, n_K , as defined above, the expected number of

failures to be found during testing is $\sum_{h=1}^K f_h n_h$.

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 [1,2,3,6].

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 state- i to state- 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.

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.
4. Make the termination state absorbing.
5. Calculate P_S , 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 termination, i.e., leave the uncertain arcs to termination if any exist.
3. For each arc (i,j) in model A , change the arc to (i,j') where j' is the state corresponding to j in model 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 an arc based reliability model calculated using M_U and M_S and arc failure estimates W_U and 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 to termination is

$$c_i = \sum_{j=1}^N M_{i,j}(x_{i,j} + c_j)$$

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 termination. Given $c_{termination} = 0$, it is now possible to define a system of linear equations that can be solved for $c_{invocation}$. 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.

5.1. Example: analytical automatic partitioning scheme

Consider the usage model of Figure 1. Uncertain arcs are marked as bold lines in the graph. The model has the transition probabilities and pre-test estimates of the arc failure rates as given in Table 1. The uncertain arcs have failure rates of 0.2 and the background failure rate is .0001. All arcs have unit testing cost.

Table 1: Transition probabilities and failure rates

From State	To State	Transition Probability	Failure Rate
S1	S2	1.0	.0001

Table 1: Transition probabilities and failure rates

From State	To State	Transition Probability	Failure Rate
S2	S3	0.069767	.2
S2	S4	0.697675	.0001
S2	S5	0.232558	.0001
S3	S2	1.0	.0001
S4	S6	0.833333	.0001
S4	S9	0.166667	.0001
S5	S10	0.5	.0001
S5	S11	0.5	.0001
S6	S7	1.0	.0001
S7	S8	1.0	.0001
S8	S2	1.0	.0001
S9	S7	1.0	.0001
S10	S11	0.5	.0001
S10	S15	0.5	.0001
S11	S5	0.697674	.0001
S11	S10	0.056221	.2
S11	S12	0.050801	.0001
S11	S13	0.056221	.0001
S11	S14	0.037481	.0001
S11	S15	0.050801	.0001
S11	S16	0.050801	.0001
S12	S11	1.0	.0001
S13	S11	1.0	.0001
S14	S11	0.5	.0001
S14	S16	0.5	.0001
S15	S11	1.0	.0001
S16	S17	1.0	.0001
S17	S1	1.0	.0001

Tables 2 and 3 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).

Table 2: Partition characteristics

Block	Probability Mass	Estimated Failure Rate
U	.5746	.3260
S	.4254	.0053

Table 3: Test case allocation

Block	Number of Tests	Expected Test Cost
U	51	89.8263
S	7	53.6868

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.

Given the allocation, Tables 4 and 5 can be computed for partition testing and simple random testing.

Table 4: Comparison by failure rate and variance

	Expected Failure Rate	Expected Variance
Partition Testing	.1896	.0003
Simple Random Testing	.1919	.0023

Table 5: Comparison by detection of failures

	Probability of Finding a Failure	Expected Number of Failures
Partition Testing	~1.0	16.7012
Simple Random Testing	~1.0	12.8573

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.

5.2. 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.

$p_1, p_2, \dots, p_K, f_1, f_2, \dots, f_K$, and c_1, c_2, \dots, 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.

5.3. Example: simulation-based automatic partitioning scheme

Using the same model as before, Tables 6 and 7 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.

Table 6: Partition characteristics

Block	Probability Mass	Estimated Failure Rate
F	.5860	.3314
S	.4140	.0036

Table 7: Test case allocation

Block	Number of Tests	Expected Test Cost
F	65	72.3874

Table 7: Test case allocation

Block	Number of Tests	Expected Test Cost
S	8	36.0338

Given the allocation, Tables 8 and 9 can be computed for partition testing and simple random testing, with the same number of tests run in each case.

Table 8: Comparison by failure rate and variance

	Expected Failure Rate	Expected Variance
Partition Testing	.1957	.0001
Simple Random Testing	.1919	.0023

Table 9: Comparison by Detection of Failures

	Probability of Finding a Failure	Expected Number of Failures
Partition Testing	~1.0	21.5577
Simple Random Testing	~1.0	12.8573

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.

6. Conclusions

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.

The partitioning schemes presented in this paper use pre-test estimates of the failure rate of the software 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.

7. References

- [1] Cochran, W.G., *Sampling Techniques*, Wiley, New York, 1977.
- [2] D. Hamlet and R. Taylor, "Partition Testing Does not Inspire Confidence", *IEEE Trans. Software Eng.*, December 1990, pp. 1402-1411.
- [3] V.N. Nair, D.A. James, W.K. Ehrlich, and J. Zevallos, "Statistical Issues in Assessing Software Testing Strategies", *Statistica Sinica*, January 1998.
- [4] 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.
- [5] 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.
- [6] E.J. Weyuker and B. Jeng, "Analyzing Partition Testing Strategies", *IEEE Trans. Software Eng.*, July 1991, pp. 97-108.
- [7] J.A. Whittaker and J.H. Poore, "Markov Analysis of Software Specifications", *ACM Transactions on Software Engineering and Methodology*, January 1993, pp. 93-106.
- [8] J.A. Whittaker, and M.G. Thomason, "A Markov Chain Model for Statistical Software Testing", *IEEE Trans. Software Eng.*, October 1994, pp. 812-824.

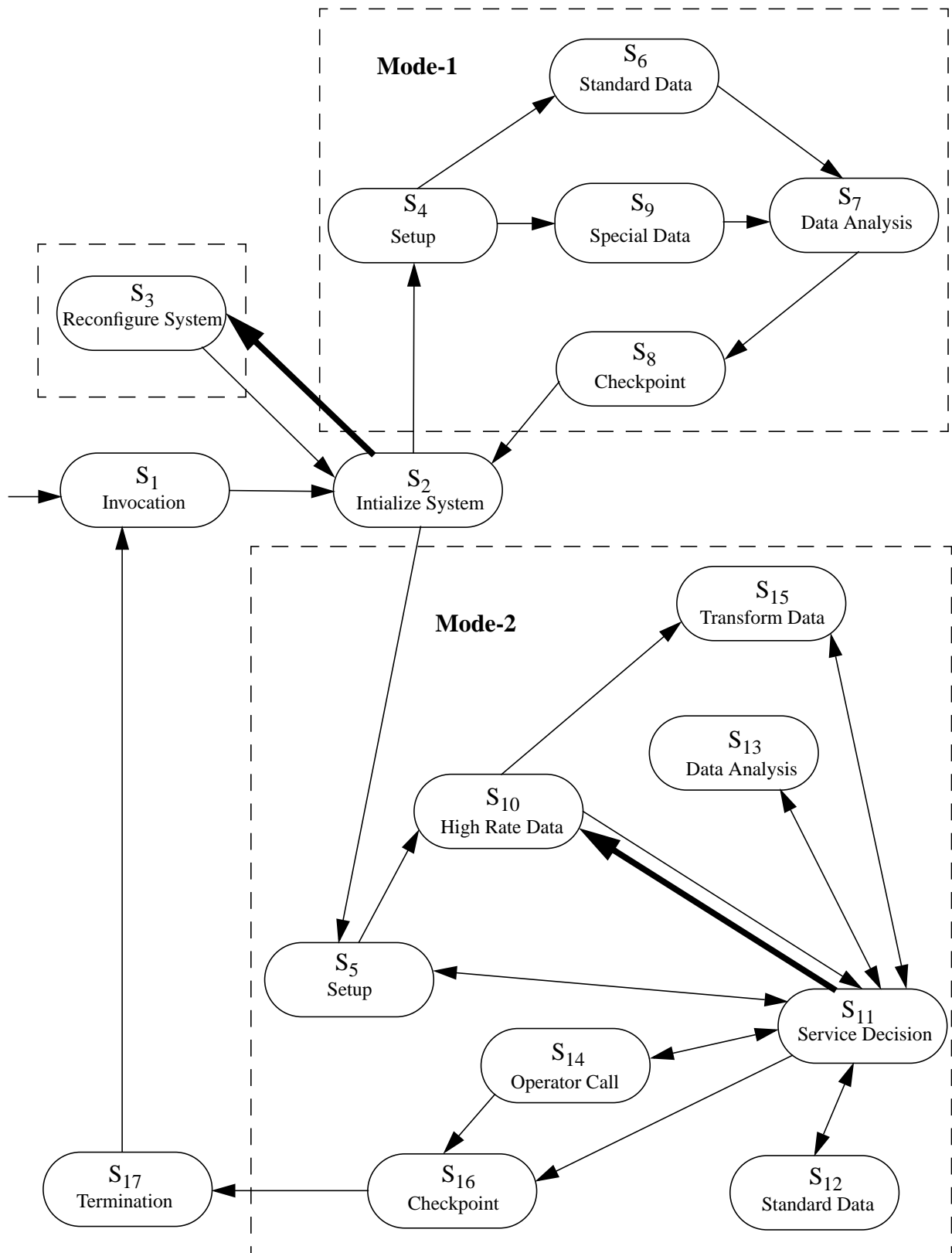


Figure 1: Example model