Combinatorial Coverage as an Aspect of Test Quality

D. Richard Kuhn, NIST kuhn@nist.gov
Raghu N. Kacker, NIST raghu.kacker@nist.gov
Yu Lei, University of Texas Arlington ylei@uta.edu

Abstract There are relatively few good methods for evaluating test set quality, after ensuring basic requirements traceability. Structural coverage, mutation testing, and related methods can be used if source code is available, but these approaches may entail significant cost in time and resources. This paper introduces an alternative measure of test quality that is directly related to fault detection, simple to compute, and can be applied prior to execution of the system under test. As such, it provides an inexpensive complement to current approaches for evaluating test quality.

Introduction

How thorough are your tests? This is a vitally important question for mission critical systems, but very difficult to answer with confidence, especially if tests were produced by third-party test developers.

Generally it must be shown that tests track to enumerated requirements, but this is a coarse grained metric. Structural coverage criteria such as statement or branch coverage may also be applied, if source code is available. Mutation testing – developing multiple versions of the code with mutations, or seeded faults – may be used to compare the fault detection capacity of alternative test suites, or evolve a test suite that produces a sufficiently high score on detecting differences between mutated versions of the code. Such an approach naturally is dependent on the mutations chosen.

Evaluating test quality is a particularly difficult and imprecise process for "black box" testing, where no source code is used. A test goal may be to positively demonstrate a collection of specified features, often by a single test for each feature or option. But simply showing that a particular input can demonstrate the feature does little to prove that an application is adequate for the wide range of inputs likely to be encountered in real-world use. Alternatively, an operational profile may be developed which tests the system according to the statistical distribution of inputs that occur in operational use. This process can provide reasonable confidence for the system's behavior in normal operation, but may miss the rare input configurations that can result in a failure.

A common approach for high assurance is to include tests designed to exercise the system with rare scenarios, based on experience or engineering judgment. This approach is clearly dependent on the skill of testers, and it may leave a large proportion of the possible input space untested. It also provides no quantitative measure of the proportion of significant input combinations that have been tested. Therefore, if test services are to be contracted out, there is little sound basis for developers to specify the level of testing required, or for testers to prove that testing has been adequate for the required assurance level. This paper describes measurement methods derived from combinatorial testing that can be used in analyzing the thoroughness of a test set, based on characteristics of the test set separate from its coverage of executable code.

Distribution of Faults

Empirical data show that most failures are triggered by a single parameter value, or interactions between a small number of parameters, generally two to six [1], a relationship known as the *interaction rule*. An example of a single-value fault might be a buffer overflow that occurs when the length of an input string exceeds a particular limit. Only a single condition must be true to trigger the fault: *input length* > *buffer size*. A 2-way fault is more complex, because two particular input values are needed to trigger the fault. One example is a search/replace function that only fails if both the search string and the replacement string are single characters. If one of the strings is longer than one character, the code does not fail, thus we refer to this as a 2-way fault. More generally, a *t*-way fault involves *t* such conditions.

Figure 1 shows the cumulative percentage of faults at t = 1 to 6 for various applications [1]. We refer to the distribution of t-way faults as the *fault profile.* Figure 1 shows the fault profile for a variety of fielded products in different application domains, and results for initial testing of a NASA database system. As shown in Figure 1, the fault detection rate increases rapidly with interaction strength, up to t=4. With the medical device applications, for example, 66% of the failures were triggered by only a single parameter value, 97% by single values or 2-way combinations, and 99% by single values, 2-way, or 3-way combinations. The detection rate curves for the other applications studied are similar, reaching 100% detection with 4 to 6-way interactions. Studies by other researchers have been consistent with these results. Thus, the impossibility of exhaustive testing of all possible inputs is not a barrier to high assurance testing. That is, even though we cannot test all possible combinations of input values, failures involving more than six variables are extremely unlikely because they have not been seen in practice, so testing all possible combinations provides very little benefit beyond testing 4 to 6-way combinations.

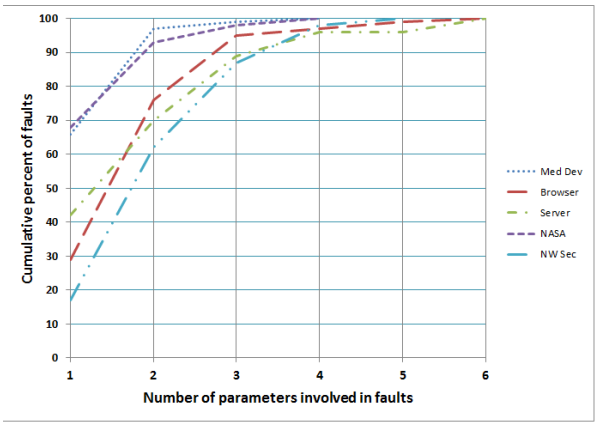

Figure 1. Cumulative fault distribution

Matrices known as *covering arrays* can be computed to cover all *t*-way combinations of variable values, up to a specified level of *t* (typically $t \leq 6$), making it possible to efficiently test all such *t*-way interactions [2]. The effectiveness of any software testing technique depends on whether test settings corresponding to the actual faults are included in the test sets. When test sets do not include settings corresponding to actual faults, the faults will not be detected. Conversely, we can be confident that the software works correctly for *t*-way combinations contained in passing tests.

As with all testing, it is necessary to select a subset of values for variables with a large number of values, and test effectiveness is also dependent on the values selected, but testing t-way combinations has been shown to be highly effective in practice. This approach is known as *combinatorial testing*, an extension of the established field of statistical Design of Experiments (DoE), endorsed by the Department of Defense Office of Test and Evaluation in 2009 [3], and used by commercial firms with demonstrated success.

Coverage Implications of Fault Distribution

The distribution of faults reported above suggests that testing which covers a high proportion of 4-way to 6-way combinations can provide strong assurance. If we know that *t* or fewer variables are involved in failures, and we can test all *t*-way combinations, then we can have reasonably high confidence that the application will function correctly. As shown above, the distribution of faults varies among applications, but two important facts are apparent: a consistently high level of fault detection has been observed for 4-way and higher strength combinations; and no interaction fault discovered so far, in thousands of failure reports, has involved more than six variables.

Any test set, whether constructed as a covering array or not, contains a large number of combinations. Measuring *combinatorial coverage*, i.e., the coverage of *t*-way combinations in a test set, can therefore provide valuable information about test set quality. Combinatorial coverage includes a number of advantages for assessing test quality:

- *Computed independently of other evaluations of test quality.* Combinatorial coverage provides additional information for decision-makers, and may be used in conjunction with structural coverage, mutation testing, or other approaches.
- *Direct relationship with fault detection.* The size of the input space spanned by the test set, a significant aspect of fault detection, can be measured by the number of *t*-way combinations up to an appropriate level of *t*. The proportion of *t*-way combinations covered measures the fractional size of the input space that is tested.
- *Simple to compute and interpret.* Because it is based on the input space of test values, there is no need to run the system under test to compute this measure of test set quality. Freely available tools can be used on any test set expressed as a matrix where rows are tests and columns are parameter values.

Measuring Coverage of Fault-triggering Combinations

Combinatorial testing is based on covering all *t*-way combinations for some specified level of *t*, but this form of testing may not always be practical because of established test practices, legal or contractual test requirements, or use of legacy test sets. An alternative to creating a combinatorial test set from scratch is to investigate the combinatorial coverage properties of an existing test set, possibly supplementing it with additional tests to ensure thorough coverage of system variable combinations. Determining the level of input or configuration state-space coverage can help in understanding the degree of risk that remains after testing. If a high level of coverage of state-space variable combinations has been achieved, then presumably the risk is small, but if coverage is much lower, then the risk may be substantial. This section describes some measures of combinatorial coverage that can be helpful in estimating this risk.

Variable-value configuration: For a set of *t* variables, a variable-value configuration is a set of *t* valid values, one for each of the variables, i.e., the variable-value configuration is a particular setting of the variables.

Example. Given four binary variables *a, b, c*, and *d*, for a selection of three variables *a, c*, and *d* the set {*a*=0, *c*=1,

d=0} is a variable-value configuration, and the set {*a*=1, *c*=1, *d*=0} is a different variable-value configuration.

Simple t-way combination coverage: For a given test set of *n* variables, simple *t*-way combination coverage is the proportion of *t*-way combinations of *n* variables for which all valid variable-value configurations are fully covered.

***Example*.** Table I shows four binary variables, *a*, *b*, *c*, and *d*, where each row represents a test. Of the six possible 2-way variable combinations, *ab, ac, ad, bc, bd, cd*, only *bd* and *cd* have all four binary values covered, so simple 2-way coverage for the four tests in Table 1 is 2/6 = 33.3%. There are four 3-way variable combinations, *abc, abd, acd, bcd*, each with eight possible configurations: 000, 001, 010, 011, 100, 101, 110, 111. Of the four combinations, none has all eight configurations covered, so simple 3-way coverage for this test set is 0%. As shown later, test sets may provide strong coverage for some measures even if simple combinatorial coverage is low.

a	b	c	d
0	0	0	0
0	1	1	0
1	0	0	1
0	1	1	1

Table 1. Test array with four binary components

It is also useful to measure the number of *t*-way combinations covered out of all possible settings of *t* variables.

Total variable-value configuration coverage: For a given combination of *t* variables, total variable-value configuration coverage is the proportion of all *t*-way variable-value configurations that are covered by at least one test case in a test set. This measure may also be referred to as total *t*-way coverage.

The number of *t*-way combinations in an array of *n* variables is $C(n,t) = n!/(n-t)!t!$, or "*n* choose *t*", the number of combinations of *n* things taken *t* at a time without repetition. If each variable has *v* values, then each set of *t* variables has v^t configurations, so the total number of possible combination settings is $v^t \times C(n, t)$. Any test set covers at least some fraction of this amount. (Note that there is a natural extension of this formula for the case where variables do not all have the same number of values.) For the array in Table I, there are C(4,2) = 6 possible variable combinations and $2^2 \times C(4,2) = 24$ possible variable-value configurations. Of these, 19 variable-value configurations are covered and the only ones missing are *ab*=11, *ac*=11, *ad*=10, *bc*=01, *bc*=10, so the total variable-value configuration coverage is 19/24 = 79%. But only two, *bd* and *cd*, out of six, are covered with all 4 value pairs. So for simple *t*-way coverage, we have only 33% (2/6) coverage, but 79% (19/24) for total variable-value configuration coverage. Although the example in Table 1 uses variables with the same number of values, this is not essential for the measurement, and the same approach can be used to compute coverage for test sets in which parameters have differing numbers of values.

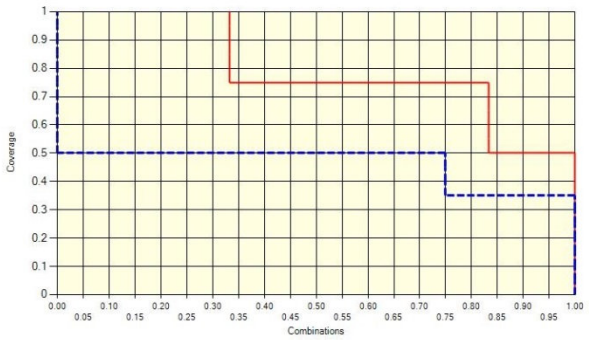

Figure 2. Graph of coverage for Table 1 tests

Figure 2 shows a graph of the 2-way (red/solid) and 3-way (blue/dashed) coverage data for the tests in Table 1. Coverage is given as the Y axis, with the percentage of combinations reaching a particular coverage level as the X axis. For example, the 2-way line (red) reaches Y = 1.0 at X = .33, reflecting the fact that 2/6 of the six combinations have all 4 binary values of two variables covered. Similarly, Y = .5 at X = .833 because one out of the six combinations has 2 of the 4 binary values covered. The area under the curve for 2-way combinations is approximately 79% of the total area of the graph, reflecting the total variable-value configuration coverage.

Practical Examples

The methods described in this paper were originally developed to analyze the input space coverage of tests for spacecraft software [4][5]. A set of 7,489 tests had been developed, although at that time combinatorial coverage was not the goal. With such a large test suite, it seemed likely that a huge number of combinations had been covered, but how many? Did these tests provide 2-way, 3-way, or even higher degree coverage?

The original test suite had been developed to verify correct system behavior in normal operation as well as a variety of fault scenarios, and performance tests were also included. Careful analysis and engineering judgment were used to prepare the original tests, but the test suite was not designed according to criteria such as statement or branch coverage. The system was relatively large, with the 82 variable configuration $1^3 2^{75} 4^2 6^2$ (three 1-value, 75 binary, two 4-value, and two 6-value). Figure 3 shows

combinatorial coverage for this system (red = 2-way, blue = 3-way, green = 4-way, orange = 5-way). This particular test set is not a covering array, but pairwise coverage is still relatively good, because 82% of the 2-way combinations have 100% of possible variable-value configurations covered and about 98% of the 2-way combinations have at least 75% of possible variable-value configurations covered.

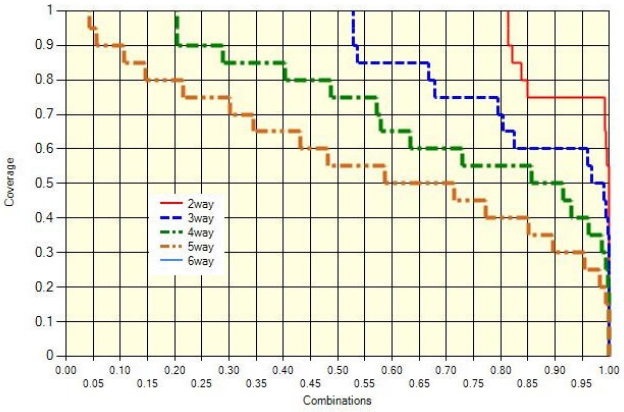

Figure 3. Configuration coverage for spacecraft example.

interaction	combinations	settings	coverage
2-way	3321	14761	94.0
3-way	88560	828135	83.1
4-way	1749060	34364130	68.8
5-way	27285336	603068813	53.6

Table 2. Total t-way coverage for Fig. 3 configuration.

Figure 4 shows a smaller example based on a US Air Force test plan [6] with seven parameters in a $2^4 3^1 4^2$ (four 2-value, one 3-value, and two 4-value) configuration, with 2-way through 6-way coverage for 122 tests. Coverage is remarkably high, with nearly 100% of all 2-way through 4-way combinations covered. Note that the 2-way and 3-way lines are not visible because with 100% coverage they appear as vertical lines on the right side of the chart.

Figure 5 shows how coverage declines with 25% of the tests removed. Although the smaller test set has less coverage for all but 2-way combinations, coverage is still relatively high, so a test manager might consider this comparison in reviewing the cost/benefit tradeoffs of adding or removing tests.

Computing Combinatorial Coverage

Tools are available to compute the measures discussed in this article. Several covering array generators can compute total coverage, and NIST-developed tools that are freely available can compute a variety of additional measures, and produce the reports included in examples above. The tools also include embedded constraint solvers, making it possible to produce counts of covered combinations excluding those that are not possible physically, or should be excluded because of constraints among variables. This is an essential feature for real-world use. It is also possible to generate additional tests to supplment those analyzed, to bring coverage up to any desired level.

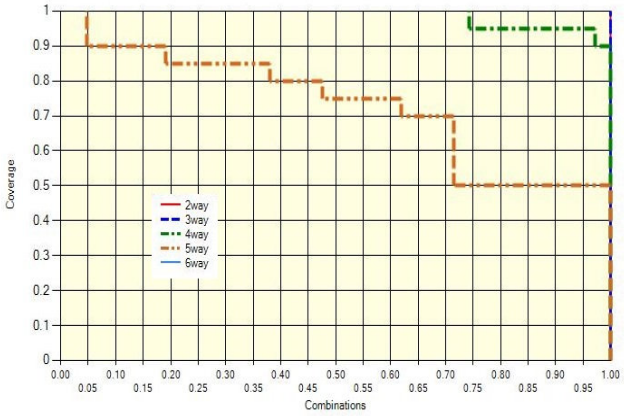

Figure 4. Configuration coverage for USAF test plan.

interaction	combinations	settings	coverage
2-way	21	152	100
3-way	35	664	100
4-way	35	1690	98.7
5-way	21	1818	69.7

Table 3. Coverage for Fig. 4 configuration.

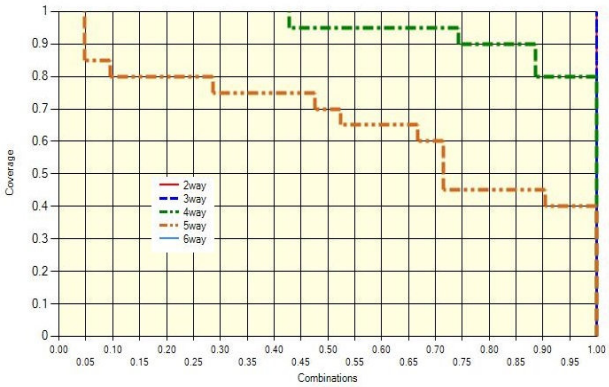

Figure 5. Configuration coverage, 75% of tests in Fig. 4.

interaction	combinations	settings	coverage
2-way	21	152	100
3-way	35	664	99.5
4-way	35	1690	90.0
5-way	21	1818	56.7

Table 4. Coverage for Fig. 5 configuration.

The methods and tools introduced above were developed for analysis of NASA software tests, and additional NASA usage has suggested the following areas of utility [7]: 1) as an inline analysis tool for evaluating developer tests, 2) as a planning tool during test development to ensure adequate coverage, 3) as an IV&V audit tool for auditing completed IV&V analysis or multi-project test plans.

Conclusions

Combinatorial coverage provides valuable information for decision-makers because it measures the proportion of the input space that is covered relevant to testing. Because only a small number of variables are involved in failures, testing all settings of 4-way to 6-way combinations can provide strong assurance. For example, if we measure the *t*-way coverage of tests, and find that all 4-way combinations are covered, 90% of 5-way combinations, and 70% of 6-way combinations are covered, we can reasonably conclude that very few potential failure-triggering combinations have been left untested. Conversely, we can also have confidence that the system has been shown to work correctly for almost all of the relevant input space. Thus, combinatorial coverage can provide significant value in evaluating test quality.

Acknowledgements

We are grateful to Greg Hutto at Eglin AFB for providing a copy of the 53d Wing tech report on design of experiments in test plan design.

Disclaimer

Certain products may be identified in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology or other agencies of the US Government, nor does it imply that the products identified are necessarily the best available for the purpose.

References

1. D.R. Kuhn, D.R. Wallace, A.J. Gallo, Jr., "Software Fault Interactions and Implications for Software Testing", IEEE Trans. on Software Engineering, vol. 30, no. 6, June, 2004.
2. NIST Special Publication 800-142*, Practical Combinatorial Testing*, Oct. 2010.
3. C. McQuery, “Design of Experiments in Test and Evaluation”. Memo, Office of the Secretary of Defense, May 1, 2009.
4. J.R. Maximoff, M.D. Trela, D.R. Kuhn, R. Kacker, "A Method for Analyzing System State-space Coverage within a t-Wise Testing Framework*", IEEE International Systems Conference 2010*, Apr. 4-11, 2010, San Diego.
5. D.R. Kuhn, I. Dominguez, R.N. Kacker and Y. Lei. "Combinatorial Coverage Measurement Concepts and Applications", *2nd Intl Workshop on Combinatorial Testing*, Luxembourg, IWCT2013, IEEE, Mar. 2013.
6. G. Hutto, “53d Wing Test Plan Examples”, Tech. Rpt., Eglin AFB, 2012.
7. C. Price, R. Kuhn, R. Forquer, A. Lagoy, R. Kacker, “Evaluating the t-way Combinatorial Technique for Determining the Thoroughness of a Test Suite”, NASA IV&V Workshop, 2013.