NISTIR 7878

Combinatorial Coverage Measurement

D. Richard Kuhn
Raghu N. Kacker
Computer Security Division
Information Technology Laboratory

Yu Lei
University of Texas Arlington

NIST

**National Institute of
Standards and Technology**

U.S. Department of Commerce

NISTIR 7878

Combinatorial Coverage Measurement

D. Richard Kuhn
Raghu N. Kacker
Computer Security Division
Information Technology Laboratory

Yu Lei
University of Texas Arlington

October 2012

U.S. Department of Commerce
Rebecca Blank, Acting Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in Federal information systems.

Authority

This publication has been developed by NIST to further its statutory responsibilities under the Federal Information Security Management Act (FISMA), Public Law (P.L.) 107-347. NIST is responsible for developing information security standards and guidelines, including minimum requirements for Federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate Federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130, Section 8b(3), Securing Agency Information Systems, as analyzed in Circular A-130, Appendix IV: Analysis of Key Sections. Supplemental information is provided in Circular A-130, Appendix III, Security of Federal Automated Information Resources.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on Federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other Federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Interagency Report 7878
16 pages (September 2012)

COMBINATORIAL COVERAGE MEASUREMENT

Abstract. Combinatorial testing applies factor covering arrays [6, 7] to test all *t*-way combinations of input or configuration state space. In some testing situations, it is not practical to use covering arrays, but any set of tests with *n* parameters covers at least some proportion of *t*-way combinations up to $t \leq n$. This report describes measures of combinatorial coverage that can be used in evaluating the degree of *t*-way coverage of any test suite, regardless of whether it was initially constructed for combinatorial coverage.

1   Introduction

Because testing all possible combinations of input values is nearly always intractable, combinatorial testing using factor covering arrays is a reasonable alternative [8, 9, 10]. But it is not always practical to re-design an organization's testing procedures to use tests based on covering arrays. Testing procedures often develop over time, and employees have extensive experience with a particular approach. Units of the organization may be structured around established, documented test procedures. This is particularly true in organizations that must test according to contractual requirements, or relevant standards. And because much software assurance involves testing applications that have been modified to meet new specifications, an extensive library of legacy tests may exist. The organization can save time and money by re-using existing tests, which may not have been developed as factor covering arrays.

Short of creating new test suites from scratch, one approach to obtaining the advantages of combinatorial testing is to measure the combinatorial coverage of existing tests, then supplement as needed. Depending on the budget and criticality of the software, 2-way through 5-way or 6-way testing may be appropriate. Building covering arrays for some specified level of *t* is one way to provide *t*-way coverage. However, many large test suites naturally cover a high percentage of *t*-way combinations. If an existing test suite covers almost all 3-way combinations, for example, then it may be sufficient for the level of assurance that is required. Determining the level of input or configuration state space coverage can also help in understanding the degree of risk that remains after testing. If 90% - 100% of the relevant state space has been covered, then risk is likely to be smaller than would remain after testing that covers a much smaller portion of the state space. This report describes measures of combinatorial coverage that can be helpful in evaluating the degree of *t*-way coverage of any test suite, regardless of whether it was initially constructed for combinatorial coverage.

2   Software Test Coverage

Test coverage is one of the most important topics in software assurance. Users would like some quantitative measure to judge the risk associated with less than effectively exhaustive testing. A variety of measures have been developed to gauge the degree of test coverage [2]. The following are some of the better-known coverage metrics:

- **Statement coverage:** This is the simplest of coverage criteria – the percentage of statements exercised by the test set.

- **Decision or branch coverage:** The percentage of branches that have been evaluated to both *true* and *false* by a test set.
- **Condition coverage:** The percentage of conditions within decision expressions that have been evaluated to both true and false. Note that 100% condition coverage does not guarantee 100% decision

coverage. For example, "`if (A || B) {do something} else {do something else}`" is tested with [0 1], [1 0], then A and B will both have been evaluated to 0 and 1, but the *else* branch will not be taken because neither test leaves both A and B false.

- **Modified condition decision coverage (MCDC):** This is a strong coverage criterion that is required by the US Federal Aviation Administration for Level A (catastrophic failure consequence) software; i.e., software whose failure could lead to loss of function necessary for safe operation. It requires that every condition in a decision in the program has taken on all possible outcomes at least once, and each condition has been shown to independently affect the decision outcome, and that each entry and exit point have been invoked at least once [4].

3 Combinatorial Coverage

Note that the coverage measures discussed above depend on access to program source code. Combinatorial testing, in contrast, is a black box technique. Inputs are specified and expected results for assessment are determined from some form of specification. The program is then treated as simply a processor that accepts inputs and produces outputs.

Suppose we have a program that accepts two inputs, x and y, with 10 values each. Then the input state space consists of the $10^2 = 100$ pairs of x and y values, which can be pictured as a checkerboard square of 10 rows by 10 columns. With three inputs, x, y, and z, we would have a cube with $10^3 = 1,000$ points in its input state space, and so on. Exhaustive testing would require inputs of all 10^n combinations, but combinatorial testing could be used to reduce the size of the test set while ensuring that all t-way combinations for some specified level of t are covered.

Looking closely at the nature of combinatorial testing leads to several measures of state space that are useful. We begin by introducing what will be called a *variable-value configuration*.

Definition. 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.*

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 $a=1$, $c=1$, $d=0$ is a different variable-value configuration.

3.1 Simple t-way combination coverage

Of the total number of *t*-way combinations for a given collection of variables, what percentage will be covered by the test set? If the test set is a covering array, then coverage is 100%, by definition, but many test sets not based on covering arrays may still provide significant *t*-way coverage. If the test set is large, but not designed as a covering array, it is possible that it provides 2-way coverage or better. For example, random input generation may have been used to produce the tests, and good branch or condition coverage may have been achieved. In addition to the structural coverage figure, for software assurance it would be helpful to know what percentage of 2-way, 3-way, etc. coverage has been obtained.

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

Example. Figure 1 shows an example with 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 Figure 1 is $1/3 = 33.3\%$. 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%.

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

Figure 1.   An example test array for a system with four binary components

3.2   (*t* + *k*)-way combination coverage

A test set that provides full combinatorial coverage for *t*-way combinations will also provide some degree of coverage for (*t*+1)-way combinations, (*t*+2)-way combinations, etc. This statistic may be useful for comparing two combinatorial test sets. For example, different algorithms may be used to generate 3-way covering arrays. They both achieve 100% 3-way coverage, but if one provides better 4-way and 5-way coverage, then it can be considered to provide more software testing assurance.

Definition. (*t* + *k*)-way combination coverage: *For a given test set that provides 100% t-way coverage for n variables, (t+k)-way combination coverage is the proportion of (t+k)-way combinations of n variables for which all variable-values configurations are fully covered.*

Example. Suppose the test set in Figure 1 is extended as shown in Figure 2 then all four combinations of pairs of variables are covered, so 2-way coverage is 100%. Out of the four 3-way combinations *abc, abd, acd, bcd* only the combination *bcd* has all 8 variable value combinations so (2+1)-way = 3-way coverage is 25%.

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

Figure 2.   Eight tests for four binary variables.

3.3   Measures of Variable-Value Configuration coverage

So far we have only considered measures of the proportion of combinations of variables for which all configurations of *t* variables are fully covered. But when *t* variables with *v* values each are considered, each *t*-tuple has v^t possible variable-value configurations. For example, in pairwise (2-way) coverage of binary variables, every 2-way combination has four configurations: 00, 01, 10, 11. We define two measures with respect to such variable-value configurations:

Definition. Variable-value configuration coverage: *For a given combination of t variables, variable-value configuration coverage is the proportion of variable-value configurations that are covered by at least one test case in a test set.*

For (*t*+*k*)-way coverage where *k* = 1, Chen and Zhang [3] have proposed the *tuple density* metric. A special metric for (*t*+1)-way coverage is useful because (1) the coverage of higher strength tuples for *t'* > *t*+1 is much lower (because the number of *t*-way combinations to be covered grows exponentially with *t*), (2) the coverage at *t*+1 provides some information for coverage at *t'* > *t*+1 because (*t*+1)-way tuples are subsumed by higher strength tuples, and (3) the number of additional faults triggered by *t*-way combinations drops rapidly with *t* > 2 [10].

Definition. Tuple Density: *Sum of t and the fraction of the covered (t+1)-tuples out of all possible (t+1)-tuples* [3].

Example. As shown in the previous example, the test set in Figure 2 provides 100% coverage of 2-way combinations and 25% coverage of 3-way combinations, so the tuple density of this test set is 2.25.

Definition. (*p, t*)-completeness: *For a given set of n variables, (p, t)-completeness is the proportion of the C(n, t) combinations that have configuration coverage of at least p* [11].

Example. For Figure 1 above, there are C(4, 2) = 6 possible variable combinations and C(4, 2) × 2^2 = 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. But only two, *bd* and *cd*, are covered with all 4 value pairs. So for the basic definition of simple *t*-way coverage, we have only 33% (2/6) coverage, but 79% (19/24) for the variable-value configuration coverage metric. For a better understanding of this test set, we can compute the configuration coverage for each of the six variable combinations, as shown in Figure 3. So for this test set, one of the combinations (*bc*) is covered at the 50% level, three (*ab*, *ac*, *ad*) are covered at the 75% level, and two (*bd*, *cd*) are covered at the 100% level. And, as noted above, for the whole set of tests, 79% of variable-value configurations are covered. All 2-way combinations have at least 50% configuration coverage, so (.50, 2)-completeness for this set of tests is 100%. Although the example in Figure 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.

Vars	Configurations	Config
a b	00, 01, 10	.75
a c	00, 01, 10	.75
a d	00, 01, 11	.75
b c	00, 11	.50
b d	00, 01, 10, 11	1.0
c d	00, 01, 10, 11	1.0

*total 2-way cove*rage = 19/24   = .79167
(.50, 2)-completeness = 6/6   = 1.0
(.75, 2)-completeness = 5/6   = 0.83333
(1.0, 2)-completeness = 2/6   = 0.33333

Figure 3.   The test array covers all possible 2-way combinations of a, b, c, and d to different levels.

The graph in Figure 4 shows a graphical display of the coverage data for the tests in Figure 3. Coverage is given as the Y axis (ordinate), with the percentage of combinations reaching a particular coverage level as the X axis (abscissa). Note from Fig. 1 that 100% of the combinations are covered to at least the .50 level, 83% are covered to the .75 level or higher, and a third covered 100%. Thus the rightmost horizontal line on the graph corresponds to the smallest coverage value from the test set, in this case 50%. Thus (.50, 2)-completeness = 100%.
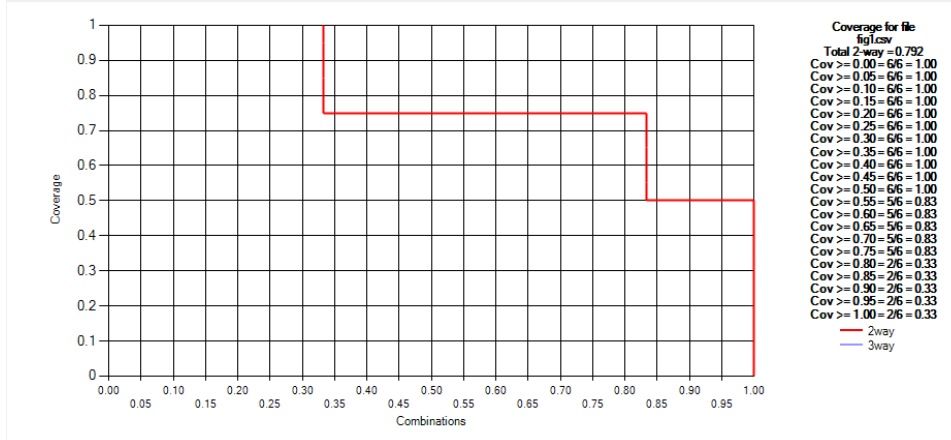

Figure 4.   Graph of coverage from Figure 3 test data.

Note that the total 2-way coverage is shown as 0.792. This figure corresponds approximately to the area under the curve in Figure 4. (The area is not exactly equal to the true figure of 0.792 because the curve is plotted in increments of 0.05.)   Additional tests can be added to provide greater coverage. Suppose an additional test [1,1,0,1] is appended to figure 1 Coverage then increases as shown in Figure 5. Now we can see immediately that all combinations are covered to at least the 75% level, as shown by the rightmost horizontal line (in this case there is only one horizontal line). The leftmost vertical line reaches the 1.0 level of coverage, showing that 50% of combinations are covered to the 100% level.
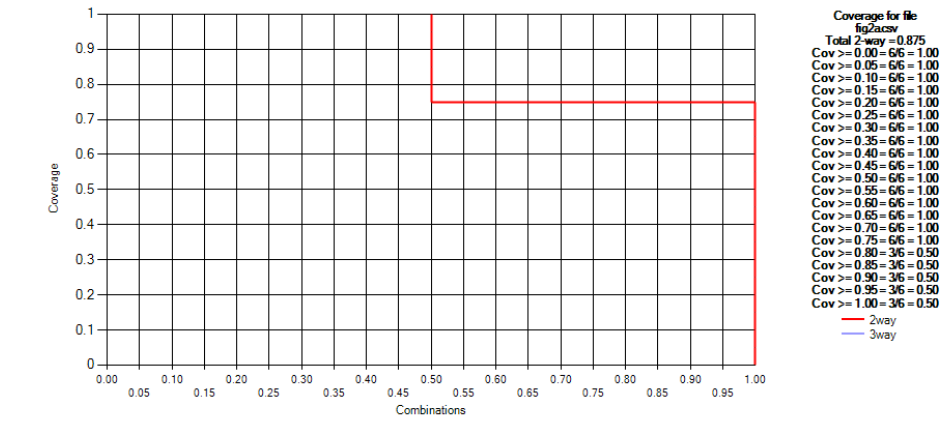

Figure 5.   Coverage after test [1,1,0,1] is appended.

Note that the upper right corner includes roughly 25 squares, so the area under the curve is 175/200, or 87.5%, matching the Total 2-way coverage figure. Appending one more test, [1,0,1,1] results in the coverage shown in Figure 6.
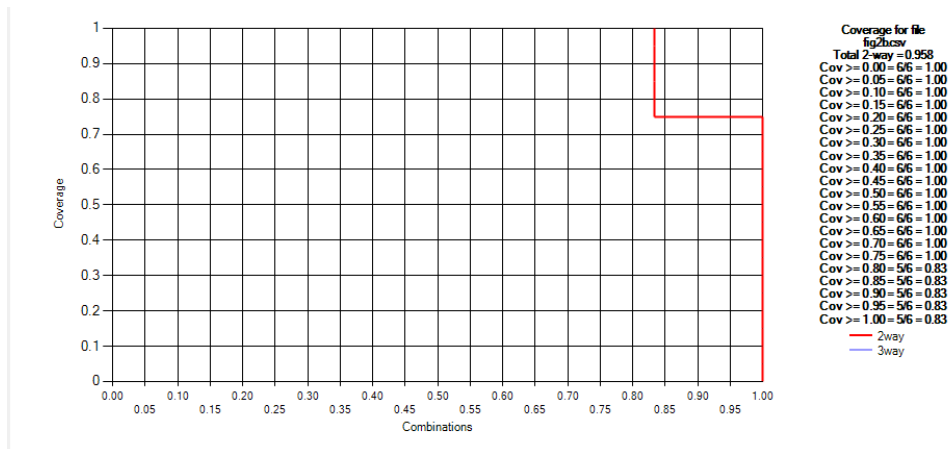
Figure 6.   Coverage after test [1,0,1,1] is appended.

If a test [1,0,1,0] is then appended, 100% coverage for all combinations is reached, as shown in Figure 7. The graph can thus be thought of as a "coverage strength meter", where the red indicator line moves to the right as higher coverage levels are reached. A covering array, which by definition covers 100% of variable-value configurations will always produce a figure as below, with full coverage for all combinations.
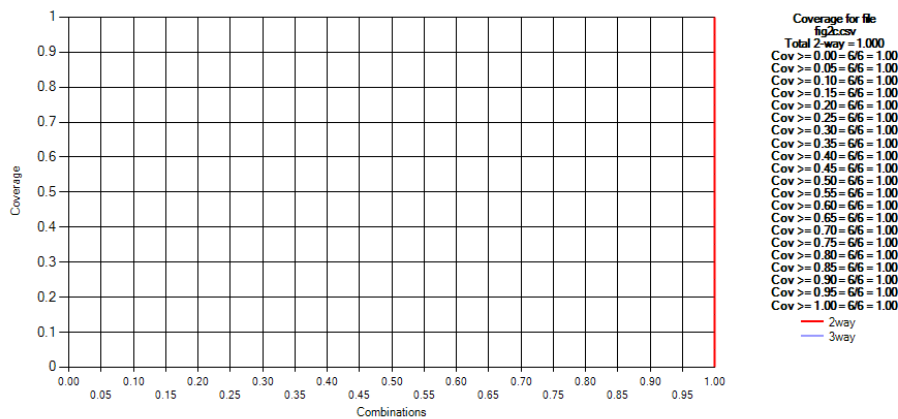

Figure 7.   Appending test [1,0,1,0] provides 100% coverage, i.e., a covering array.

Consider the graph in Figure 8 corresponding to test set in figure 1. The symbol Φ indicates the proportion of combinations with 100% variable-value coverage, and M indicates the minimum proportion of coverage for all *t*-way variable combinations. In this case 33% (Φ) of the combinations have full variable-value coverage, and all variable combinations are covered to at least the 50% level (M). So (1.0, 2)-completeness = Φ and (M, 2)-completeness = 100%, but it is helpful to have more intuitive terms for the points Φ and M. Note that Φ is the level of simple *t*-way coverage.   Since all combinations are covered to at least the level of M, we will refer to M as the "*t-way minimum coverage*", keeping in mind that "coverage" refers to a proportion of variable-value configuration values. Where the value of *t* is not clear from the context, these measures are designated Φ_t and M_t. Using these terms we can analyze the relationship between total variable-value configuration coverage, *t*-way minimum coverage and simple *t*-way coverage.
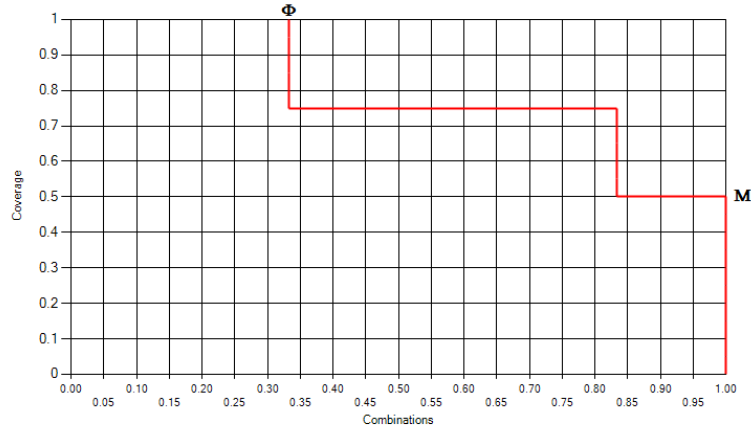
Figure 8.   Example coverage graph for t = 2.

Let S_t = total variable-value coverage, the proportion of variable-value configurations that are covered by at least one test. If the area of the entire graph is 1(i.e., 100% of combinations), then

$$S_t \geq 1 - (1- \Phi_t)(1- M_t)$$
$$S_t \geq \Phi_t + M_t - \Phi_t M_t \tag{1}$$

If a test suite has only one test, then it covers $C(n, t)$ combinations. The total number of combinations that must be covered is $C(n,t) \times v^t$, so the coverage of one test is $1/v^t$. Thus,

$$t\text{-way minimum coverage}: \qquad M_t \geq \frac{1}{v^t} > 0. \tag{2}$$

Thus, for any non-empty test suite, t-way minimum coverage $\geq 1/v^t$. This can be seen in Figure 9, which shows coverage for a single test with ten binary variables, where 2-way minimum coverage is 25%. With only one test, 2-way full coverage is of course 0, and S = total variable-value coverage (denoted "Total t-way" on the graph legend) is 25%.
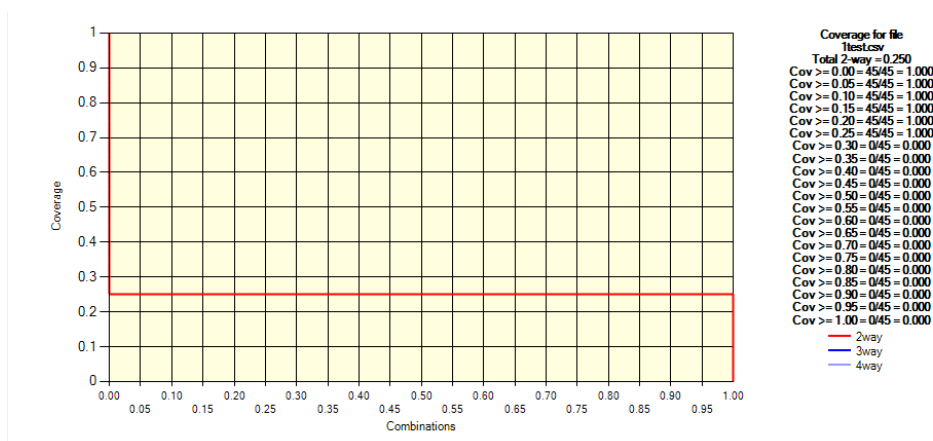

Figure 9.   Coverage for one test, 10 binary variables.

3.4 Example Application

The methods described in this report were originally developed to analyze the state space coverage of spacecraft software [11]. A very thorough set of over 7,000 tests had been developed for each of three systems. 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? If an existing test suite is relatively thorough, it may be practical to supplement it with a few additional tests to bring coverage up to the desired level.
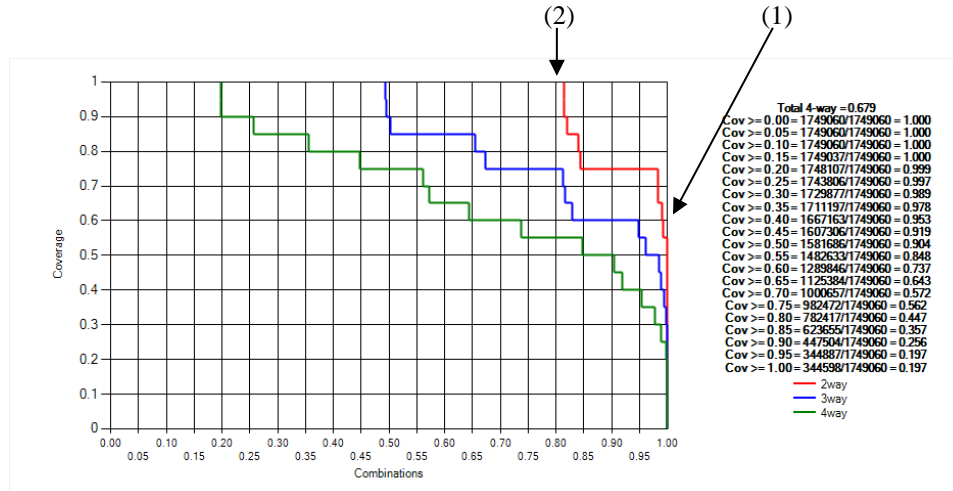

Figure 10.   Configuration coverage for $1^3 2^{75} 4^2 6^2$ inputs.

The original test suites 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, 72 binary, two 4-value, and two 6-value). Figure 10 shows combinatorial coverage for this system (red = 2-way, blue = 3-way, green = 4-way). This particular test set was not a covering array, but pairwise coverage is still relatively good, with about 99% of the 2-way combinations having at least 75% of possible variable-value configurations covered (1), and 82% have 100% of possible variable-value configurations covered (2).

3.5 Analysis of Test Strategies

These coverage metrics can be used to analyze various testing strategies by measuring the combinatorial coverage they provide. To illustrate this type of analysis, some examples are discussed in this section.

All-values: Consider the *t*-way coverage from one of the most basic test criteria, all-values, also called "each-choice" [2]. This strategy requires that every parameter value be covered at least once. If all parameters have the same number of values, *v*, then only *v* tests are needed to cover all. Test 1 has all parameters set to v_1, Test 2 to v_2, and so on. If parameters have different numbers of values, where p_1 .. p_n have v_i values each, the number of tests required is at least $Max_{i=1,n} v_i$.

Example: If there are three values, 0, 1, and 2, for five parameters, then 0,0,0,0,0; 1,1,1,1,1; and 2,2,2,2,2 will test all values once. As shown above, each test covers $1/v^t$ of the variable-value configurations, and no combination appears in more than one test, so with *v* values per parameter and thus *v* tests, we have

$$M_t \ (all\text{-}values) \ \geq \ v\frac{1}{v^t} \ = \ \frac{1}{v^{t-1}}$$

<div align="right">(3)</div>

Therefore, for the all-values criterion, where all values are covered at least once, minimum coverage $M \geq 1/v^{t-1}$. We can also reach this result by noting that each test covers $C(n, t)$ combinations, so with v values the proportion of combinations covered is $vC(n,t)/C(n,t)v^t = 1/v^{t-1}$. This relationship can be seen in Figure 11, which shows coverage for two tests with ten binary variables; 2-way minimum coverage = .5, and 3-way coverage = .25.
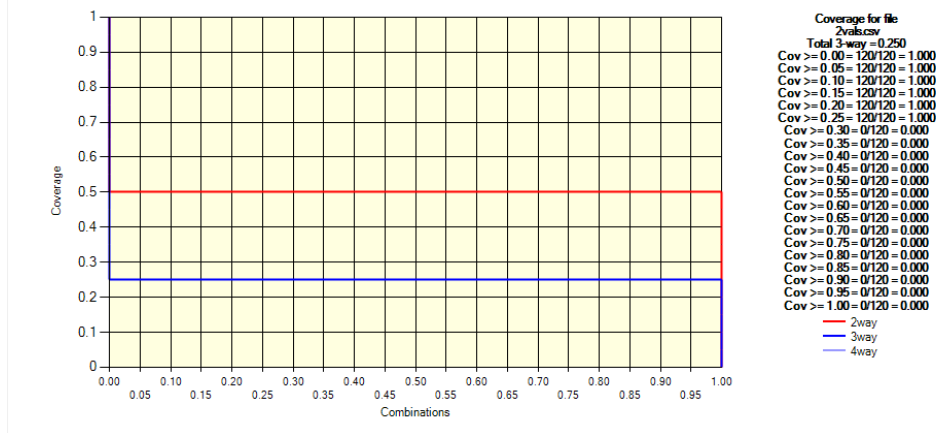

Figure 11.   t-way coverage for two tests with binary values.

Base choice: Base-choice testing [1] requires that every parameter value be covered at least once and in a test in which all the other values are held constant. Each parameter has one or more values designated as base choices. The base choices can be arbitrary, but can also be selected as "special interest" values, e.g., default values, or values that are used most often in operation. If parameters have different numbers of values, where $p_1 \ .. \ p_n$ have v_i values each, the number of tests required is at least $1 + \Sigma_{i=1,n} (v_i\,\text{-}1)$, or where all n parameters have the same number of values v, the number of tests is $1+n(v\text{-}1)$. An example is shown below in 0, with four binary parameters.

	a	b	c	d
base:	0	0	0	0
test 2	1	0	0	0
test 3	0	1	0	0
test 4	0	0	1	0
test 5	0	0	0	1

Figure 12.   Base choice test suite for a 2^4 configuration

The base choice strategy can be highly effective, despite its simplicity. In one study of five programs seeded with 128 faults [8], it was found that "although the Base Choice strategy requires fewer test cases than Orthogonal Arrays and AETG, it found as many faults [8]." In that study, AETG [5] was used to generate 2-way (pairwise) test arrays. We can use combinatorial coverage measurement to help understand this finding. For this example of analyzing base choice, we will consider n parameters with 2 values each. First, note that the base test in which each parameter takes its base choice covers $C(n, t)$ combinations, so for pairwise testing this is $C(n, 2) = n(n\text{-}1)/2$. Changing a single value of the base test to something else will cover n-1 new pairs (in our example, *ab*, *ac*, and *ad* have new values in test 2, while

bc and *bd* are unchanged). This must be done for each parameter, so we will have the original base test combinations plus $n(n-1)$ additional combinations. The total number of 2-way combinations is $C(n, 2) \times 2^2$, so for *n* binary parameters:

$$M_t \text{ (2-way binary base-choice)} = \frac{n(n-1)/2 + n(n-1)}{C(n,2)2^2}$$

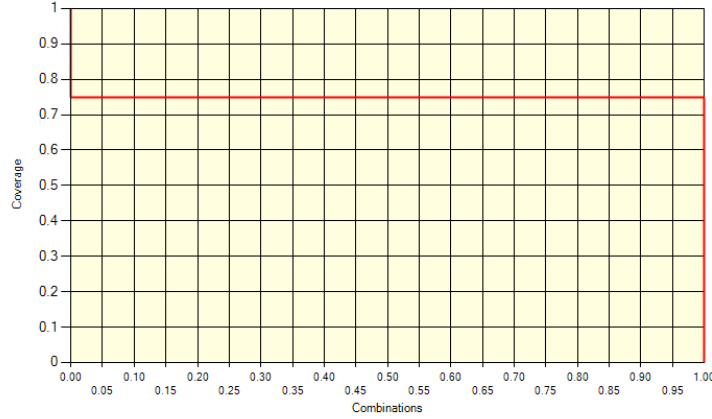$$= \frac{C(n,2) + 2C(n,2)}{C(n,2)2^2}$$

$$= 3/4.$$

Figure 13.   Graph of 2-way coverage for test set in 0.

This can be seen in the graph in Figure 13 of coverage for 0. Note that the 75% coverage level is independent of *n*. For $v > 2$, the analysis is a little more complicated. The base choice test covers $C(n, 2)$ combinations, and each new test covers *n*-1 new combinations, but we need *v*-1 additional tests beyond the base choice for *v* values. So the proportion of 2-way combinations covered by the base choice strategy in general is:

$$M_t \text{ (base-choice)} = \frac{C(n,2) + (v-1)2C(n,2)}{C(n,2)v^2}$$

$$= \frac{1 + 2(v-1)}{v^2}$$

For example, with $v = 3$ values per parameter, we still cover 55.6% of combinations, and with $v = 4$, coverage is 43.75%. This analysis helps explain why the base choice strategy can be effective even if it does not achieve full 2-way coverage.

This equation can be generalized to higher interaction strengths. The base test covers $C(n,t)$ combinations, and each new test covers $C(n-1,t-1)$ new combinations, since the parameter value being varied can be combined with *t*-1 other parameters for each *t*-way combination. Base choice testing requires $n(v-1)$ additional tests beyond the initial one, so for any *n*, *t* with $n \geq t$

$$M_t \text{ (base-choice)} = \frac{C(n,t) + n(v-1)C(n-1,t-1)}{C(n,t)v^t}$$

$$= \frac{C(n,t) + t(v-1)C(n,t)}{C(n,t)v^t}$$

$$= \frac{1+t(v-1)}{v^t}$$

$$(4)$$

3.6 Analysis of (t+1)-way Coverage

A t-way covering array by definition provides 100% coverage at strength t, but it also covers some $(t+1)$-way combinations (assuming $n \geq t+1$). Using the concepts introduced previously we can investigate minimal coverage levels for $(t+1)$-way combinations in covering arrays. Given a t-way covering array, for any set of $t+1$ parameters, we know that any combination of t parameters is fully covered in some set of tests. Joining any other parameter with any combination of t parameters in the tests will give a $(t+1)$-way combination, which has v^{t+1} possible settings. For any set of tests covering all t-way combinations, the proportion of $(t+1)$-way combinations covered is thus v^t/v^{t+1}, so if we designate $(t+1)$-way variable-value configuration coverage as S_{t+1}, then

$$S_{t+1} \geq \frac{1}{v} \text{ for any } t\text{-way covering array with } n \geq t+1. \qquad (5)$$

Note that variable-value configuration coverage is not the same as simple t-way coverage, which gives the proportion of t-way configurations that are 100% covered. Clearly no set of $(t+1)$-way combinations will be fully covered with a t-way covering array if $N < v^{t+1}$, where N = number of tests. For most levels of t and v encountered in practical testing, this condition will hold. For example, if $v=3$, then a 2-way covering array with less than $3^3=27$ tests can be computed (using IPOG-F) for any test problem with less than 60 parameters. So the proportion of combinations with full variable-value coverage, designated Φ, will be zero for 3-way coverage for this example. And in general, designating $(t+1)$-way full variable-value configuration coverage as Φ_{t+1}, if $N < v^{t+1}$, then $\Phi_{t+1} = 0$, for any t-way covering array with $n \geq t+1$.

As an additional illustration, we show that expression (5) can also be reached by noting that with a t-way covering array, unique $(t+1)$-way combinations can be identified as follows: traversing the covering array, for the first appearance of each t-way combination, append each of the $n-t$ parameters not contained in the t-way combination to create a $(t+1)$-way combination. This procedure counts each $(t+1)$-way combination $C(t+1, t) = t+1$ times. The covering array contains $C(n,t)v^t$ t-way combinations, so the proportion of $(t+1)$-way combinations, S_{t+1}, in the t-way array is at least

$$\frac{C(n,t)v^t(n-t)/(t+1)}{C(n,t+1)v^{t+1}} = \frac{1}{v}$$

In most practical cases, the $(t+1)$-way coverage of a t-way array will be higher than $1/v$.

3.7 Summary

1. Many coverage measures have been devised for code coverage, including statement, branch or decision, condition, and modified condition decision coverage. These measures are based on aspects of source code and are not suitable for combinatorial coverage measurement.

2. Combinatorial coverage measurements may be useful in understanding test set thoroughness and for analyzing test strategies. Measures include simple t-way coverage, tuple density, variable-value configuration coverage, t-way minimum coverage, and (p,t)-completeness. Test strategies differ in the level of coverage they provide as measured by these concepts.

3. The following properties hold:

$$M_t \geq \frac{1}{v^t} > 0$$

$$S_t \geq \Phi_t + M_t - \Phi_t M_t$$

$$S_{t+1} \geq \frac{1}{v} \text{ for any } t\text{-way covering array with } n \geq t+1$$

if $N < v^{t+1}$, then $\Phi_{t+1} = 0$, for any t-way covering array with $n \geq t+1$

where
  M_t = minimum proportion of coverage for all t-way variable combinations;
  Φ_t = proportion of t-way combinations with 100% variable-value coverage;
  S_t = proportion of total t-way variable-value coverage;
  N = number of tests.

References

1. Ammann, P. E. & Offutt, A. J. (1994). Using formal methods to derive test frames in category-partition testing, *Proceedings of the Ninth Annual Conference on Computer Assurance* (COMPASS'94),Gaithersburg MD, IEEE Computer Society Press, pp. 69-80.
2. P. Ammann, J. Offutt, *Introduction to Software Testing*, Cambridge University Press, New York, 2008.
3. B. Chen, J. Zhang, Tuple Density: A New Metric for Combinatorial Test Suites, *Proceedings of the 33rd International Conference on Software Engineering*, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011. ACM 2011, ISBN 978-1-4503-0445-0
4. J. J. Chilenski, An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion, *Report DOT/FAA/AR-01/18*, April 2001, 214 pp.
5. D.M. Cohen, S.R. Dalal, M.L. Friedman, and G.C. Patton, *"The AETG System: An Approach to Testing Based on Combinatorial Design"*, *IEEE Transactions on Software Engineering*, Vol. 23, No. 7, pp. 437–444, (1997).
6. C.J. Colbourn, Combinatorial aspects of covering arrays, *Matematiche* (Catania) 58 (2004) 121–167.
7. C.J. Colbourn, J.H. Dinitz (Eds.), *The CRC Handbook of Combinatorial Designs*, CRC Press, Boca Raton, 1996.
8. M. Grindal, J. Offutt, S.F. Andler, Combination Testing Strategies: a Survey, *Software Testing, Verification, and Reliability*, v. 15, 2005, pp. 167-199.
9. R. Kuhn, R. Kacker, Y. Lei, J. Hunter, "Combinatorial Software Testing", *IEEE Computer*, vol. 42, no. 8 (August 2009).
10. D.R. Kuhn, D.R. Wallace, and A. Gallo, "Software Fault Interactions and Implications for Software Testing," *IEEE Transactions on Software Engineering*, 30(6): 418-421, 2004
11. 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.