2017 IEEE International Conference on Software Quality, Reliability and Security (Companion Volume)

Combinatorial and MC/DC Coverage Levels of Random Testing

Sergiy Vilkomir, Aparna Alluri
Department of Computer Science
East Carolina University
Greenville, NC 27858, USA
vilkomirs@ecu.edu

D. Richard Kuhn, Raghu N. Kacker
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
{kuhn, raghu.kacker}@nist.gov

***Abstract*—Software testing criteria differ in their effectiveness, the numbers of test cases required, and the processes of test generation. Specific criteria often are compared to random testing, and in some cases, random testing shows a surprisingly high level of effectiveness. One reason that this is the case is that any random test set has a specific level of coverage according to any coverage criterion. Numerical evaluation of coverage levels of random testing according to various coverage criteria is an interesting research task and is important in understanding the relationship between different testing approaches. In this paper, we performed an experimental evaluation of the coverage levels of random testing for two criteria: MC/DC and combinatorial *t*-way testing. Our experiments showed that, when the number of random test cases increased, a high level of coverage was reached rapidly, both for MC/DC and *t*-way. However, many more random tests are required to reach 100% coverage. An unexpected result was that there were significant differences in the measurement of partial MC/DC coverage by various tools. The results may be used to select optimal methods for practical testing and develop new testing methods based on the integration of existing approaches.**

Keywords—random testing; combinatorial testing; MC/DC; pairwise; coverage

I. INTRODUCTION

Random testing is a simple and practical approach in software testing. Its different modifications are based on random selection of test cases from the input domain. Another popular approach that is used widely is testing based on coverage criteria. According to ISO/IEC/IEEE 29119-1:2013 Standard [1], test coverage is "a degree, expressed as a percentage, to which specified test coverage items have been exercised by a test case or test cases." Simple examples include statement, path, and branch coverage.

A more sophisticated example is *t*-way combinatorial coverage, which measures the proportion of *t*-way combinations of values included in a test set. The *t*-way criterion requires that all possible combinations of values of *t* parameters be included in some test case of the test suite, i.e., 100% *t*-way testing using coverage arrays [2, 3]. A covering array, CA(t, n, v) is an array of k rows, such that every *t*-way combination of *n* variables with *v* values is covered at least once. Efficient means exist to generate covering arrays up to a few hundred variables [3]. In the most basic form for *t*=2, this criterion is known as pairwise testing, and requires that all possible pairs of values be covered [4].

Another example of the coverage criterion is one of the strongest structural (code) criteria for testing logical predicates—Modified Condition/Decision Coverage (MC/DC)—[5], which requires that every condition in a decision is covered and that "each condition has been shown to independently affect the decision's outcome" [6]. MC/DC coverage subsumes branch coverage and, in turn, statement coverage of the code. Because MC/DC is such a strong criterion, the U.S. Federal Aviation Administration (FAA) has for many years required MC/DC testing for Level A (life critical) software aboard commercial aircraft [5, 7, 8]. However, it is used rarely outside the aerospace industry. One of the most significant barriers to wider use of MC/DC is its expense. While the cost of testing consumer-grade software is roughly the same as that of producing the code, costs in the aviation industry may be seven times higher for verification than for development [9]. To encourage the use of MC/DC beyond the aerospace industry, the cost of its application needs to be reduced.

Test criteria differ in the numbers of test cases required, processes of test generation, and effectiveness, i.e., the ability to detect faults during testing. Empirical evaluation and experimental comparison of testing criteria began in the 1980s [10, 11], but continues to be an important research direction [12, 13, 14, 38, 39]. Specific criteria are often compared to random testing [15, 16, 17, 18, 40].

In some cases, random testing shows a surprisingly high level of effectiveness. For example, in testing logical expressions, random testing provides practically the same results as does *t*-way testing [18, 19]. One reason that this is the case is that a random (or any other) test set has a specific level of coverage according to any coverage criterion. Thus, random testing has a certain level of pairwise coverage, 3-way coverage, MC/DC coverage, etc. This level is never 100%, except for extremely large random test sets, but it can be quite high.

For many practical problems, a covering array is the most efficient method to cover all *t*-way combinations, but some problems are too large for current covering array algorithms. For example, software product lines may contain many

This work was performed under the following financial assistance award 70NANB15H217 from the U.S. Department of Commerce, National Institute of Standards and Technology (NIST).

978-1-5386-2072-4/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS-C.2017.19

hundreds or even thousands of variables. An alternative to generating covering arrays for such testing problems is to generate tests randomly, in sufficient numbers to cover a large proportion of *t*-way combinations for an appropriate value of *t*. Numerical evaluation of coverage levels of random testing according to various coverage criteria is an interesting and important research task.

In this paper, we performed an experimental evaluation of the coverage levels of random testing for two criteria: MC/DC and *t*-way testing. The paper is structured as follows: Section II discusses the purpose and practical value of the investigation and formulates the two research questions. The methods and scope of the investigation are described in Section III. Section IV provides the experimental results on the MC/DC coverage level of random testing, and Section V provides similar results for combinatorial coverage levels. Threats to the validity of our work are addressed in Section VI, and conclusions and directions for future work are presented in Section VII.

II. Purpose and Research Questions

The goal of our study was to investigate how effective random testing is in providing coverage for different criteria, and how the level of coverage changes depending on the number of test cases in a random test set. We considered two specific research questions:

- RQ1: What is the level of MC/DC coverage of random test sets of different sizes?
- RQ2: What is the level of *t*-way coverage of random test sets of different sizes?

This investigation addressed both research and applied issues.

With respect to research, our goal was to obtain original results that provide new numerical evaluations of coverage levels and help us understand the relationships between random testing and other approaches. Even when two criteria are completely independent and use different principles, such relationships exist, because a test set that satisfies the first criterion has some level of coverage according to the second criterion and the converse. For example, a 100% MC/DC test set has some pairwise coverage and a test set that provides 100% pairwise coverage also provides a certain percentage of MC/DC coverage. This fact is well-known, and some initial results have been obtained on such relationships, i.e., between pairwise and MC/DC testing [14]. However, this area still requires further empirical evaluations, including those of random testing. We considered our research efforts to be a useful step in this direction.

From an applied perspective, our results can be helpful in solving two tasks. The first is test generation when 100% coverage is not required, but it is desirable to achieve a high (although not 100%) level of coverage. In this situation, testers can apply random testing but still care about coverage. Our results allow the evaluation of the levels of coverage that can be expected and the number of random tests that are necessary to achieve desirable levels of coverage.

The second practical task is to develop new approaches to achieve MC/DC coverage using random testing. One popular idea is to combine different testing approaches and exploit the advantages of each. The purpose of such a combination is to minimize the number of test cases, maximize test effectiveness, and simplify test generation. Research in this direction includes combining functional and structural testing [20, 21], model-based and combinatorial testing [22, 23], model-based and search-based testing [24], and more. Because of its simplicity, random testing is a good candidate for combination with some other technique to achieve MC/DC coverage. Development of such a method is out of the scope of this paper. However, we consider it an important direction for future work and our current results can act as a foundation for such efforts.

III. Methods and Scope of Investigation

A. Used tools

To measure MC/DC coverage, we used CodeCover and Testwell CTC++ tools. CodeCover is an open-source, white-box testing tool developed at the University of Stuttgart, Germany in 2007 [25, 26]. CodeCover measures several types of coverage, including term coverage (subsumes MC/DC), and supports several programming languages, including Java and C. Testwell CTC++ is a code coverage and dynamic analysis tool for C and C++ code but also supports Java and C#. The tool was developed by Testwell Ltd. (Finland) [27], now owned (since 2013) by Verifysoft Technology GmbH [28]. As well as CodeCover, CTC++ provides measurements of several types of coverage, including MC/DC. A sample report produced by Testwell CTC++ is presented in Fig. 1.

```
HITS/TRUE      FALSE    LINE DESCRIPTION
=============================================================================

         2                  6 FUNCTION main()
         1          1      10   if (myfile . is_open ( ))
         8          1      12     while (getline ( myfile , line ))
        24          8      19       while (iss >> val)
                           22       }+
         3          5      26       if (( a && b ) || ( a && c ))
         2                 26         1: (T && T) || (_ && _)
         1                 26         2: (T && F) || (T && T)
         0                 26         3: (F && _) || (T && T)
                    1      26         4: (T && F) || (T && F)
                    0      26         5: (T && F) || (F && _)
                    0      26         6: (F && _) || (T && F)
                    4      26         7: (F && _) || (F && _)
                           26         MC/DC (cond 1): 1 + 7, 1 - 6
                           26         MC/DC (cond 2): 1 + 4, 1 - 5
                    -      26         MC/DC (cond 3): 2 - 5, 3 - 7
                           26         MC/DC (cond 4): 2 + 4, 3 - 6
                           26       }+
                           27       else
                           27       }+
                           28     }+
                           30   }+
                           31   else
                           31   }+
         2                 32   return 0
                           33 }

***TER  93 % ( 13/ 14) of FUNCTION main()
      100 % ( 21/ 21) statement
```

Fig. 1. Testwell CTC++ sample report.

We used two tools to evaluate MC/DC levels because the results of such an evaluation for the same software and the same test sets frequently differ for different tools. The main reason for this is that there is no commonly accepted definition of an incomplete (<100%) MC/DC coverage. Different principles are used in different tools. Some tools evaluate coverage separately for each logical condition and then calculate an average value. Other tools create complete MC/DC test sets, compare them with used test sets, and evaluate the percentage of coverage based on this.

In this paper, it is not our task to judge the different approaches to MC/DC evaluation. There are some justifications for all of them. However, we provide the results of MC/DC evaluation from two tools and compare them in Section 4.

To measure *t*-way coverage, we used the Combinatorial Coverage Measurement (CCM) tool developed by the National Institute of Standards and Technology (NIST) and the Centro Nacional de Metrologia of Mexico [29, 30]. CCM can analyze existing levels of *t*-way (*t*=2…6) coverage for any test set. Fig. 2 shows the CCM tool user interface. The tool displays a graph of the coverage for the given tests. Additional tests can be added to increase the coverage. The percentage of coverage can be viewed in the "Results" tab.

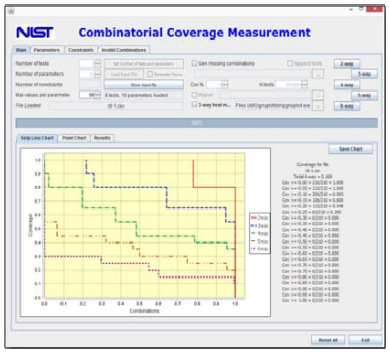

Fig. 2. CCM tool user interface.

B. *Primary steps and scope of the investigation*

The general organization of the experimental evaluation is illustrated in Fig. 3.

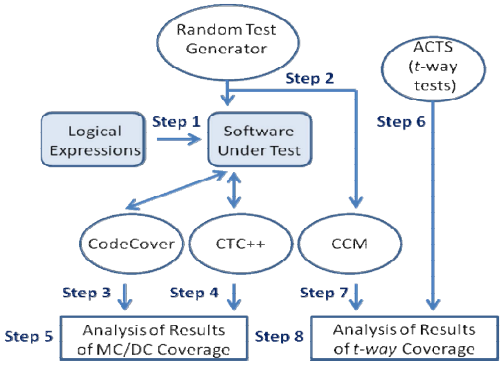

Fig. 3. Organization of experimental evaluation.

The investigation included the following main steps:

Step 1. Generation of logical expressions of different sizes (i.e., different numbers of logical variables in expressions). The sets of expressions were entered into software programs that have no real functionality, but are used for testing purposes to evaluate coverage levels. Thus, the programs contain "if-then" statements with multiple-conditional predicates, but do nothing when the predicate equals *true*. We omitted functionality because it does not affect coverage.

A total of 100 logical expressions were generated for testing: 50 expressions were generated from 10 fixed input variables and another 50 from 20 fixed input variables. Both sets contained 25 simple (no repeated variables) and 25 complex (repeated variables) expressions of different sizes: 25 expressions of size 3, 15 of size 4, 5 of size 5, 2 of size 6, and 1 each of sizes 7, 8, and 9. Some examples of the expressions generated from 10 variables are presented in Table I. The sizes of the expressions were chosen to reflect situations in real software, i.e., more expressions of small size and fewer of large size. The proportion of different sizes used corresponded approximately to data on expression sizes reported in [31, 32].

TABLE I. EXAMPLES OF LOGICAL EXPRESSIONS

	Simple expressions	**Complex expressions**
Size 3	$d + f + c$	$(h + \bar{d})(\bar{h} + \bar{e})$
Size 4	$(c + f)(i + g)$	$(f(a + h)) + (\bar{h}e + \bar{f})$
Size 5	$(ce) + (\bar{a}f + d)$	$(\bar{h} + \bar{f})(\bar{c} + h + ea)(f + \bar{e})$

Step 2. Generation of random test sets of different sizes, where the size of a random test set is the number of test cases in it. To generate test cases and check that all test cases in a set are different, we created a simple software program that used Java's random number generator: 42 random test sets of different sizes were generated for 10 variables and 42 similar sets for 20 variables. The sizes of the random test sets generated were 1, 2, 3…25, 30, 40…100, 200…900, 1024. Three test sets were generated for each size, so the coverage for any random test size was the average of the three coverage values.

Step 3. Evaluation of MC/DC coverage for random test sets using the CodeCover tool.

Step 4. Evaluation of MC/DC coverage for random test sets using the Testwell CTC++ tool.

Step 5. Analysis of experimental data on MC/DC coverage.

Step 6. Generation of *t*-way tests using ACTS tool [33] developed at the National Institute of Standards and Technology. The main purpose of this step was to evaluate the sizes of combinatorial test sets to allow comparison of random and combinatorial test sets of the same sizes (see Table IV in Section V).

Step 7. Evaluation of *t*-way coverage for random test sets using the CCM tool.

Step 8. Analysis of experimental data on *t*-way coverage.

The total number of logical expressions and test sets are summarized in Table II.

TABLE II. SCOPE OF EXPERIMENTAL TESTING

Number of		
Logical Variables	**Logical Expressions of mixed sizes**	**Random Test Sets**
10	50	126
20	50	126
Total	100	252

IV. MC/DC LEVEL OF RANDOM TESTING

All test inputs in generated random test sets have values T (True) or F (False). The example of the test set of size 5 as one of 42 generated random test sets is presented in Table III.

TABLE III. EXAMPLE OF THE RANDOM TEST SET OF SIZE 5

	a	b	c	d	e	f	g	h	i	j
1	T	T	F	F	T	F	F	F	T	F
2	T	F	T	F	F	T	T	F	F	F
3	T	F	T	F	T	T	T	T	T	F
4	T	T	F	F	F	T	F	F	F	F
5	F	T	F	F	F	T	T	T	T	F

The results show that MC/DC coverage demonstrated a fast-growing trend when the number of random test cases increased. This trend was shown both by CodeCover (Fig. 4) and CTC++ (Fig. 5) tools for the simple and complex expressions. For the simple expressions, MC/DC coverage reached 99% for 55 random test cases and complete 100% MC/DC coverage was achieved starting from the 100 tests. The results by CodeCover and CTC++ for simple expressions were similar and close to each other (Fig. 6). Note that for all plots the scale on the *x* axis is not linear.

For complex expressions, more test cases were required to reach maximum MC/DC coverage. The coverage was very close to the maximum from 200 test cases, and the maximum MC/DC coverage required approximately 400 random test cases. However, it is necessary to mention two distinct features of the estimation of MC/DC coverage for complex expressions:

- The MC/DC levels reported by the two tools were significantly different (Fig. 7).
- The maximum of MC/DC coverage did not reach 100%. Even after exhaustive testing (all 1024 possible test cases for 10 variables), the maximum level of MC/DC coverage was 93% according to CodeCover and only 77% according to CTC++.

This situation holds not only for CodeCover and CTC++ but also for other tools providing MC/DC coverage; for example, Kalimetrix Logiscope [34] or TESSY [35]. The reason, as mentioned in Section 3.A, might be that different tools use different principles for coverage evaluation. They can also evaluate different types of MC/DC; for example, Unique-Cause MC/DC vs. Masking MC/DC. The other factors affecting the evaluation of the coverage are how the tools treat short-circuit Boolean expressions and multiple occurrences of logical variables in expressions. Thus, in our case, CTC++ considers multiple occurrences of the same variable as different variables and requires test cases with different values of the first and second occurrences of the variable, which is impossible.

Under this definition, complex expressions can never have 100% MC/DC coverage. In contrast to CTC++, CodeCover considers multiple occurrences as the same variable, so 100% MC/DC coverage for complex expressions is possible. However, when the short-circuit operator is used for evaluating expression values, CodeCover considers some variables as uncovered even if test cases provide necessary coverage.

To investigate how the total number of logical variables in the software affects MC/DC coverage of random testing, we repeated the testing with 20 variables instead of 10. The sizes of the logical expressions remained the same (from 3 to 9), but variables for each expression were selected from the set of 20 variables. Each random test case had 20 input values, though not all were used for each expression. Data for this testing are presented in Fig. 8 and Fig. 9.

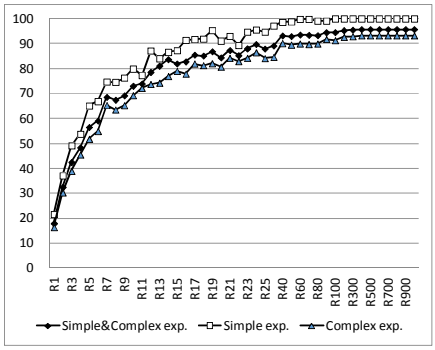

Fig. 4. MC/DC coverage by CodeCover for random tests (10 variables).

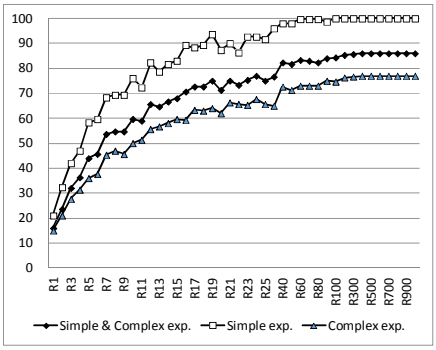

Fig. 5. MC/DC coverage by Testwell CTC++ for random tests (10 variables.).

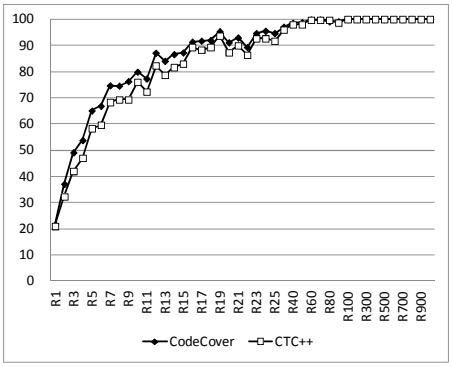

Fig. 6. Comparison of results for simple expressions (10 variables).

Some conclusions were similar for 10 and 20 variables:

- The levels of MC/DC coverage for simple expressions by CodeCover and CTC++ were close to each other.
- The levels of MC/DC coverage for complex expressions by CodeCover and CTC++ were significantly different, and CodeCover reported higher levels of coverage.

- Maximal level of MC/DC coverage for complex expressions did not reach 100%.

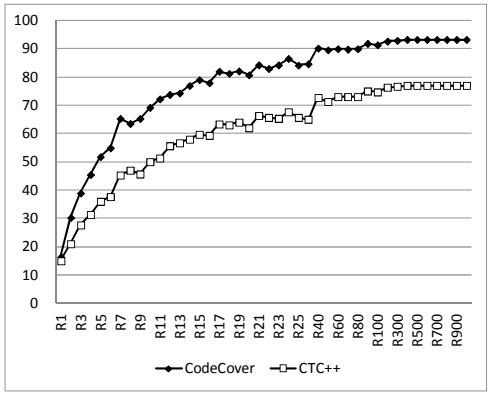

Fig. 7. Comparison of results for complex expressions (10 variables).

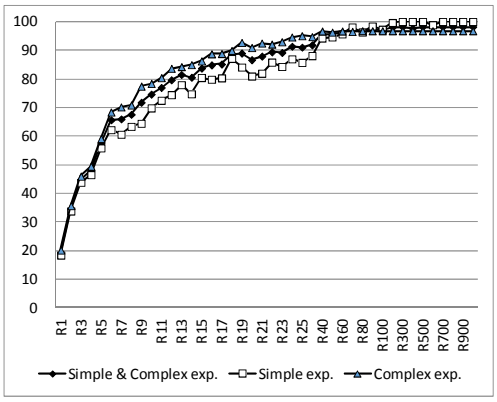

Fig. 8. MC/DC coverage by CodeCover for random tests (20 variables).

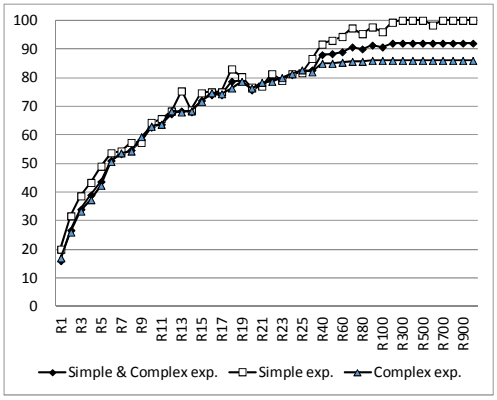

Fig. 9. MC/DC coverage by Testwell CTC++ for random tests (20 variables).

However, numerical data were slightly different for 20 variables vs. 10 variables:

- For the simple expressions, MC/DC coverage reached 99% level for 200 random test cases and complete 100% MC/DC coverage was achieved starting at 400 tests.
- For the complex expressions, maximal possible levels of MC/DC coverage were 96.8% by CodeCover and 86% by CTC++.
- For the complex expressions, maximal levels of MC/DC coverage were reached after 100 random test cases.

In general, random test cases rapidly achieved a high level of MC/DC coverage as the number of tests increased. The precise data are given above but, very approximately, around 100 random tests provided high MC/DC coverage in all cases. Of course, this number is much higher than the number of MC/DC test cases for one expression, which is $n+1$ for expressions size n, i.e., maximum 10 tests for expressions size 9. However, the number of different MC/DC tests necessary for all expressions together can be close to these 100 tests. At the same time, the process of MC/DC test generation is much harder compared with random testing and should be done separately for each expression.

V. Combinatorial Coverage Level of Random Testing

In contrast to MC/DC, the combinatorial coverage level does not depend on logical expressions in the software, but only on the input variables. In this section, we evaluated t-way coverage of random test cases for t=2…6. As in Section IV, we considered this coverage for 10 and 20 input variables, using the same random test cases we used in Section IV with sizes that ranged from 2 to 1024.

To understand the combinatorial coverage level of random test cases, it is necessary to consider random sets of the same sizes as the t-way covering arrays. The sizes of combinatorial test sets for t=2…6 are presented in Table IV. Although they are not optimal (minimal), they approximate the best-known values [36] and reflect sizes of combinatorial test sets generated by the ACTS tool. There is no known method to compute the minimum size for a covering array.

TABLE IV. Sizes of Combinatorial Test Sets

Number of logical variables	2-way	3-way	4-way	5-way	6-way
10	10	20	44	93	178
20	12	27	66	165	375

Tables V and VI show that the level of combinatorial coverage of random test cases is quite high. Thus, in all situations for any n and t, the level of coverage for random test sets of the same size as combinatorial test sets is approximately 90%–97%.

We can easily compute an estimator for the level of combinatorial coverage of a given number of random tests. For t-way coverage of variables with v values each, we must cover all v^t settings of combinations of these variables taken t at a time. Thus, the number of combinations to be covered is:

$$v^t \times C(n,t), \text{ where } C(n,t) = n!/t!(n-t)! \quad (1)$$

TABLE V. COMBINATORIAL COVERAGE FOR RANDOM TESTS (10 VAR.)

Random Test Set Size	2-way	3-way	4-way	5-way	6-way
2	46.85	24.51	12.44	6.25	3.12
3	60.55	34.27	18.08	9.25	4.67
4	68.33	41.24	22.63	11.86	6.08
5	76.85	49.51	28.12	14.96	7.70
6	83.52	55.49	32.11	17.31	9.01
7	85.74	59.23	35.48	19.58	10.33
8	85.92	63.96	39.61	22.20	11.78
9	92.78	70.48	44.58	25.28	13.47
10	96.85	76.70	49.29	27.94	14.86
11	94.44	76.60	51.57	30.10	16.20
12	98.52	81.39	54.71	32.17	17.52
13	97.78	83.40	57.68	34.31	18.76
14	97.78	83.16	58.00	35.03	19.48
15	99.26	87.95	63.13	38.37	21.18
16	98.15	86.84	63.43	39.53	22.30
17	99.44	91.42	69.30	43.66	24.43
18	99.44	90.73	68.26	43.27	24.63
19	99.81	94.10	73.06	46.64	26.40
20	98.89	91.18	70.29	45.43	26.21
21	99.81	94.69	75.81	50.24	29.09
22	99.81	95.69	77.41	51.41	29.87
23	100.00	95.49	77.80	52.35	30.76
24	100.00	96.84	79.70	54.14	32.02
25	100.00	97.51	81.58	56.31	33.43
30	100.00	98.23	86.07	61.76	37.85
40	100.00	99.51	92.74	72.51	47.37
50	100.00	99.96	97.12	81.48	55.84
60	100.00	100.00	96.32	84.80	60.76
70	100.00	100.00	99.01	90.00	68.60
80	100.00	100.00	99.53	93.01	73.16
90	100.00	100.00	99.65	94.43	76.68
100	100.00	100.00	99.87	96.60	81.34
200	100.00	100.00	100.00	99.96	97.42
300	100.00	100.00	100.00	100.00	99.61
400	100.00	100.00	100.00	100.00	99.98
500	100.00	100.00	100.00	100.00	100.00
600	100.00	100.00	100.00	100.00	100.00
700	100.00	100.00	100.00	100.00	100.00
800	100.00	100.00	100.00	100.00	100.00
900	100.00	100.00	100.00	100.00	100.00
1024	100.00	100.00	100.00	100.00	100.00

Each set of n inputs includes C(n,t) combinations at each level of t, for n variables, so each test or input set can cover $1/v^t$ of the total number of settings. The proportion of combinations not covered by each test will thus be $1 - 1/v^t$. For Boolean expressions, v=2, so the proportion of non-covered combinations for each test is $1–2^{-t}$. Thus, for random generation, k tests will produce t-way coverage of approximately:

$$\text{coverage} \approx 1 - (1 - 2^{-t})^k \qquad (2)$$

Levels of coverage calculated thus are very close to the empirical values shown in Tables V and VI. For example, expression (2) estimated the level of coverage of 3-way combinations for 30 tests to be 98.18%, by comparison to 98.23% for 10 and 98.69% for 20 variables. Notice that this calculation is independent from the number of variables, and the values in Table V are very close to the comparable values in Table VI, typically within one percentage point. More on the production of t-way tests using random generation can be found in [37].

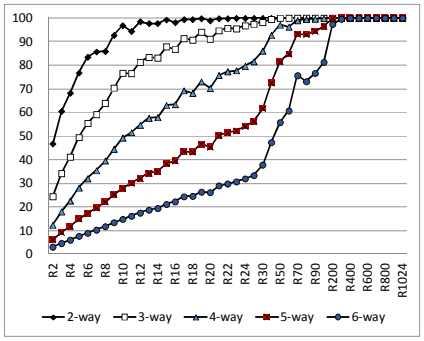

Fig. 10. Combinatorial coverage for random tests (10 variables).

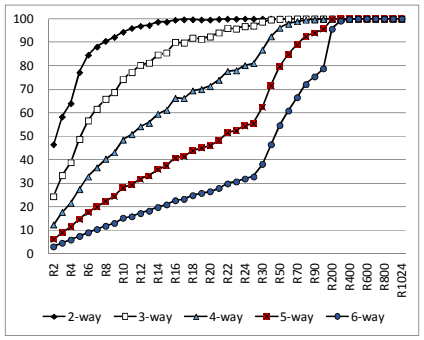

Fig. 11. Combinatorial coverage for random tests (20 variables).

The level of combinatorial coverage grows rapidly as the number of random tests increases (Fig. 10 and 11). However, after 90%, the increase slows significantly. To reach 100% combinatorial coverage, considerably more random tests are required compared with combinatorial test sets based on covering arrays. Thus, 23 (for n=10) and 24 (for n=20) random test cases are necessary to archive 100% pairwise coverage, compared to 10 and 12 test cases using covering arrays. Similar situations prevail for t-way coverage, as shown in Tables V and VI. To produce 100% t-way coverage, we need k large enough to reduce the expected number of uncovered combinations from the initial value of $v^t \times C(n,t)$ to less than one remaining uncovered combination at the end of random test generation, i.e.,: $v^t \times C(n,t) \times (1 - 2^{-t})^k < 1$. Consequently, the number of random tests needed for 100% t-way coverage increases with n.

VI. THREATS TO VALIDITY

There were several threats to the validity of our investigation. The primary threat is that we analyzed specially created programs with inserted logical expressions, but not real software. Therefore, the results of our experiments may not reflect the characteristics of real programs completely. To reduce this problem, we attempted to consider this when we created the sets of logical expressions. As mentioned in Section III-B, the proportion of sizes of expressions reflects situations in real software. Most of the expressions created had small sizes, and only a few expressions contained six or more

variables. We believe it is important to study real-world software, and although it is beyond the scope of this paper, we plan to use real software as the next step in our future work. On the other hand, using separate logical expressions (as in this paper) is also important and allows direct investigation of the effect on coverage of sizes and complexities of predicates.

TABLE VI. COMBINATORIAL COVERAGE FOR RANDOM TESTS (20 VAR.)

Random Test Set Size	2-way	3-way	4-way	5-way	6-way
2	46.57	24.43	12.41	6.23	3.12
3	58.34	33.41	17.80	9.16	4.64
4	64.02	38.91	21.67	11.54	6.01
5	77.19	48.80	27.50	14.62	7.54
6	84.61	56.70	32.82	17.60	9.10
7	88.03	61.58	36.68	20.02	10.46
8	90.48	65.84	40.40	22.46	11.86
9	92.19	68.71	43.16	24.45	13.07
10	94.38	74.22	48.49	28.14	15.29
11	95.97	77.20	50.84	29.45	15.89
12	96.88	80.29	54.19	31.79	17.26
13	97.24	81.22	55.66	33.22	18.27
14	98.64	84.71	59.52	35.94	19.83
15	98.64	85.55	61.20	37.54	20.94
16	99.47	90.09	66.24	40.86	22.71
17	99.65	89.75	66.19	41.38	23.33
18	99.69	91.83	69.41	43.92	24.87
19	99.56	91.38	69.98	45.04	25.82
20	99.56	92.38	71.31	46.10	26.54
21	99.83	94.10	73.94	48.33	27.96
22	99.96	95.95	77.69	51.54	29.88
23	99.91	95.73	78.03	52.40	30.72
24	100.00	96.79	80.17	54.36	31.96
25	99.96	96.99	81.05	55.43	32.81
30	100.00	98.69	86.76	62.43	38.18
40	100.00	99.60	92.40	71.66	46.51
50	100.00	99.89	96.10	79.74	54.68
60	100.00	99.94	97.82	84.91	60.86
70	100.00	99.99	98.86	88.99	66.54
80	100.00	100.00	99.52	92.51	72.08
90	100.00	100.00	99.64	93.97	75.40
100	100.00	100.00	99.84	95.59	78.80
200	100.00	100.00	100.00	99.83	95.64
300	100.00	100.00	100.00	99.99	99.10
400	100.00	100.00	100.00	100.00	99.79
500	100.00	100.00	100.00	100.00	99.96
600	100.00	100.00	100.00	100.00	99.99
700	100.00	100.00	100.00	100.00	100.00
800	100.00	100.00	100.00	100.00	100.00
900	100.00	100.00	100.00	100.00	100.00
1024	100.00	100.00	100.00	100.00	100.00

The structure and complexity of expressions also may have affected the experimental results. Some of our expressions may appear to be unrealistic for certain types of software (e.g., business applications), but they are typical for safety-critical software, e.g., avionics or nuclear power plant systems. To reduce this threat, we divided all logical expressions into two groups (simple and complex) and analyzed results for each group separately, as well as for all expressions together.

Real software usually contains dependencies between different logical statements and between separate conditions in the same expression. In our study, separate conditions within an expression were independent that may have threatened validity. However, dependencies between different expressions were considered because expressions contained common Boolean variables as elementary conditions. In addition, real programs contain not only Boolean inputs, but also input variables of other types that play a role in logical expressions. However, this difference has a greater effect on the process of test generation than on the numerical evaluations of coverage.

The final threat to validity is that the experimental results depended on the correctness and accuracy of the used tools. Different tools can indeed produce slightly different evaluations of the MC/DC coverage of the same software. To reduce this threat, we used two different tools to evaluate MC/DC and provided the results from both for further analysis.

VII. CONCLUSIONS AND FUTURE WORK

This paper evaluated the ability of random testing to provide coverage according to MC/DC and *t*-way testing criteria. We generated one hundred logical expressions of different sizes and also generated 252 random test sets with 2 to 1024 test cases in a set. These sets were used to test a software with logical expressions, and the levels of coverage were evaluated using the structural coverage CodeCover and Testwell CTC++ tools, and also CCM tool, which measures the coverage of input value combinations in a test suite.

Our experiments showed that, when the number of random test cases increased, a high level of coverage was reached rapidly, both for MC/DC and *t*-way. The high level of MC/DC coverage was achieved after approximately 100 random tests. The paper provides detailed data for different types of expressions and testing tools.

An unexpected result of the study was that structural coverage tools differed in their definition of partial MC/DC coverage, resulting in significant variation in coverage calculations. The primary standard for MC/DC coverage, RTCA DO-178B, requires full coverage, so partial coverage numbers generally are not used. In cases in which knowledge of less than complete MC/DC coverage is of interest, a consistent definition of partial coverage must be specified.

Random test sets of the same size as the *t*-way sets provided a 90%–97% level of combinatorial coverage. However, many more random tests are required to reach 100% coverage. This paper provided detailed data for different numbers of input variables, different types of expressions, and *t*-way coverage for t=2…6.

The results obtained can help integrate random testing with other approaches and improve the effectiveness of testing software with a complex logical structure. Thus, combining random testing with other techniques to achieve 100% MC/DC coverage is one direction for future work. It is also important to investigate further the coverage levels of random tests for real, large-scale software programs.

DISCLAIMER

REFERENCES

[1] International Standard ISO/IEC/IEEE 29119-1:2013 "Software and systems engineering — Software testing — Part 1: Concepts and definitions," 2013.

[2] M. Grindal, J. Offutt, S. Andler, "Combination Testing Strategies: a Survey," Software Testing, Verification and Reliability, Vol. 15, No. 3, pp. 167-199, 2005.

[3] D. R. Kuhn, R. Kacker, and Y. Lei, Introduction to Combinatorial Testing, Chapman and Hall/CRC, 2013, 341 pages.

[4] D. R. Kuhn, R. Kacker, Y. Lei, and J. Hunter, "Combinatorial software testing, " IEEE Computer, Volume 42, no. 8, August 2009.

[5] RTCA/DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," RTCA, Washington D.C., USA, 1992.

[6] J. Chilenski and S. Miller, "Applicability of Modified Condition/Decision Coverage to software testing," Software Engineering Journal, September 1994, pp. 193-200.

[7] RTCA/DO-178C "Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, 2012.

[8] DOT/FAA/AR-01/18, "An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion," 2001.

[9] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," IEEE Software, May/June 2013, Volume 30, Issue 3, pp. 50-57.

[10] M. R. Girgis and M. R. Woodward, "An experimental comparison of the error exposing ability of program testing criteria," Proceedings of the Workshop on Software Testing, pp. 64-73. IEEE Computer Society Press, July 1986.

[11] V. Basili and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, Volume 13, Issue 12, December 1987, pp. 1278–1296.

[12] C.-A. Sun, Y. Zai, and H. Liu, "Evaluating and comparing fault-based testing strategies for general boolean specifcations: A series of experiments," The Computer Journal, 2015, Volume 58, Issue 5, pp. 1199-1213.

[13] P.S. Kochhar, F. Thung, and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," Proceedings of the IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER), Montreal, Canada, 2-6 March 2015, pp. 560-564.

[14] S. Vilkomir and D. Anderson, "Relationship between pair-wise and MC/DC testing: Initial experimental results," Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2015), 13-17 April 2015, Graz, Austria.

[15] P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet, "An experimental study on software structural testing: deterministic versus random input generation," Proceedings of the 21st International Symposium on Fault-Tolerant Computing (FTCS 91), IEEE Press, Jun. 1991, pp. 410-417.

[16] M. A. Vouk, K-C. Tai, and A. Paradkar, "Empirical studies of predicate-based software testing," Proceedings of the 5th International Symposium on Software Reliability Engineering, 6-9 November 1994, pp. 55-64.

[17] T. Y. Chen, F. C. Kuo, H. Liu, and W. E. Wong, "Code coverage of adaptive random testing," IEEE Transactions on Reliability, Volume 62, Issue 1, March 2013, pp. 226-237.

[18] S. Vilkomir, O. Starov, and R. Bhambroo, "Evaluation of t-way Approach for Testing Logical Expressions in Software," Proceedings of the IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2013), 18–20 March 2013, Luxembourg, pp. 249-256.

[19] W. Ballance, S. Vilkomir, and W. Jenkins, "Effectiveness of Pair-wise Testing for Software with Boolean Inputs," Proceedings of the Fifth International Conference on Software Testing, Verification and Validation (ICST 2012), April 17-21, 2012, Workshop on Combinatorial Testing (CT-2012), Montreal, Canada, pp. 580-585.

[20] S. Liu and Y. Chen, "A relation-based method combining functional and structural testing for test case generation," Journal of Systems and Software, Volume 81, Issue 2, February 2008, pp. 234-248.

[21] C. Pfaller and M. Pister, "Combining Structural and Functional Test Case Generation," Proceedings of the Software Engineering Conference (SE08) , Munich, February 2008, pp. 229-241.

[22] C. D. Nguyen, A. Marchetto, and P. Tonella, "Combining model-based and combinatorial testing for effective test case generation," Proceedings of the 2012 International Symposium on Software Testing and Analysis, pp. 100-110, ACM, 2012.

[23] R. Bartholomew, "An industry proof-of-concept demonstration of automated combinatorial test," Proceedings of the 8th International Workshop on Automation of Software Test (AST'13), May 18-19, 2013, San Francisco, CA, USA, pp. 118-124.

[24] E. P. Enoiu, K. Doganay, M. Bohlin, D. Sundmark, and P. Pettersson, "MOS: an integrated model-based and search-based testing tool for function block diagrams," Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering, pp. 55-60. IEEE Press, 2013.

[25] "CodeCover," http://codecover.org , accessed at December 2016.

[26] R. Schmidberger, "Well-Defined Coverage Metrics for the Glass Box Test," In Testing Software and Systems, pp. 113-128. Springer, 2014.

[27] Testwell, "Testwell CTC++. Test Coverage Analyzer for C/C++," http://www.testwell.fi/ctcdesc.html , accessed at December 2016.

[28] Verifysoft, "Testwell CTC++ Test Coverage Analyser," http://www.verifysoft.com/en_ctcpp.html, accessed at December 2016.

[29] National Institute of Standards and Technology (NIST), "Combinatorial Coverage Measurement Tool, User Guide, January 30, 2011," http://csrc.nist.gov/groups/SNS/acts/documents/ComCoverage110130.pdf, accessed at December 2016.

[30] I. Dominguez Mendoza, D. R. Kuhn, R. N. Kacker, and Y. Lei, "CCM: A Tool for Measuring Combinatorial Coverage of System State Space," Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2013), 10-11 Oct. 2013, Baltimore, Maryland, USA, p. 291.

[31] J. Chilenski, "An investigation of three forms of the modified condition decision coverage (MCDC) criterion," Tech. Report DOT/FAA/AR-01/18, FAA, 2001.

[32] V. Durelli, et al., "What to expect of predicates: An empirical analysis of predicates in real world programs," Journal of Systems and Software, Volume 113, March 2016, pp. 324-336.

[33] National Institute of Standards and Technology (NIST), "Automated Combinatorial Testing for Software (ACTS)," http://csrc.nist.gov/groups/SNS/acts/, accessed at December 2016.

[34] Kalimetrix, "Logiscope TestChecker," http://www.kalimetrix.com/logiscope/testchecker, accessed at Dec 2016.

[35] Razorcat, "Automated testing of embedded software," http://www.razorcat.eu/tessy.html, accessed at December 2016.

[36] C. Colbourn, "Covering array tables for t=2, 3, 4, 5, 6," http://www.public.asu.edu/~ccolbou/src/tabby/catable.html, accessed at December 2016.

[37] A. Arcuri, L. Briand, "Formal analysis of the probability of interaction fault detection using random testing," IEEE Transactions on Software Engineering, Volume 38, Issue 5, 2012, pp. 1088-1099.

[38] A. Javeed and C. Yilmaz. "Combinatorial interaction testing of tangled configuration options," Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2015), 13-17 April 2015, Graz, Austria.

[39] R. C. Bryce, A. Rajan, and M. P. Heimdahl, "Interaction testing in model-based development: Effect on model-coverage," Proceedings of the IEEE 13th Asia Pacific Software Engineering Conference (APSEC'06), 6-8 December 2006, Bangalore, India, pp. 259-268.

[40] J. Bach, and P. J. Schroeder, "Pairwise testing: A best practice that isn't," Proceedings of 22nd Annual Pacific Northwest Software Quality Conference, 12-13 October 2004, Portland, Oregon, pp. 175-192.