Combinatorial and Random Testing Effectiveness for a Grid Computer Simulator

D. Richard Kuhn[1], Raghu Kacker[1], Yu Lei[2]
[1]*National Institute of Standards and Technology*, [2]*University of Texas, Arlington*,
kuhn@nist.gov, raghu.kacker@nist.gov, ylei@uta.edu

***Abstract*:** This paper compares the effectiveness of random and *t*-way combinatorial testing, where $t = 2, 3, 4$, for a grid computer network simulator. Previous investigations of random vs. combinatorial testing have reached conflicting results, with some showing more effective fault detection for combinatorial testing and others finding no significant difference between the two methods. In this paper, these two methods are compared for deadlock detection on a simulator with tests covering 2-way to 4-way combinations of configuration values, paired with an equal number of randomly generated tests. Random testing provided better results than pairwise (2-way) testing and there was no statistically significant difference between the methods for 3-way testing, but 4-way combinatorial tests detected more deadlocks than the same number of random tests. The paper reviews explanations for these results and implications for testing.

1 Background

Pairwise testing is a well-established practice in software assurance, and *t*-way/ combinatorial testing – using 3-way, 4-way, or higher strength combinations – is attracting increasing attention. In pairwise, or all-pairs, testing, every possible pair of input parameters is assigned every pairwise combination of input values at least once. Generalizing this approach to *t*-way combinations for $t > 2$ is referred to as combinatorial or *t*-way testing. As with any test methodology, it is important to measure the effectiveness of combinatorial testing, and to compare it with other methods so that test engineers can make informed decisions.

Some studies have compared the effectiveness of combinatorial and random approaches to testing, but have reached conflicting results. Schroeder et al. (2004) and Bach, Schroeder (2004) found no significant difference between these two testing approaches, and Bryce et al. (2006) found that combinatorial testing provided only a small improvement over random testing in structural coverage, and no improvement in coverage for black box tests derived from requirements. However Kobayashi et al. (2001), Bell, Vouk (2005), Pretschner et al. (2008) and Ellims, Ince, Petre (2008) found *t*-way testing to be more effective for fault detection. In this paper we compare the effectiveness of these two approaches in finding configurations that lead to deadlock in a grid computer network simulation.

Evidence for the effectiveness of combinatorial testing includes extensive investigations of pairwise testing (e.g., Burr and Young, 1998; Burroughs et al., 1994; Dohen et al., 1996; Dunietz et al., 1997; Williams and Probert, 1996), some studies using interaction strengths above pairwise (Schroeder et al., 2004; Kuhn and Okun, 2006), and empirical data on the number of faults at different interaction strengths (Wallace and Kuhn, 2001; Kuhn et al., 2004; Bell, 2006). These previous studies covered a variety of application domains. The effectiveness of pairwise and other combinatorial test methods rests on the observation that a significant number of events in software are triggered only by the interaction of two or more variable values. By including tests for all 2-way, 3-way, etc., interactions, the test set should be able to detect faults that occur only with complex interactions.

The key enabler in combinatorial testing is a *covering array* that covers all *t*-way combinations of parameter values, for the desired strength *t*. Covering arrays are combinatorial

objects that represent interaction test suites. A covering array, $CA(N;t,k,v)$, is an N x k array, where k is the number of variables, and v is the number of possible values for each variable such that in every N x t subarray, each t-tuple occurs at least once, then t is the *strength* of the coverage of interactions, Each row of the covering array represents a test, with one column for each parameter that is varied in testing. Collectively, the rows of the array include every t-way combination of parameter values at least once. For example, Figure 1 shows a covering array that includes all 3-way combinations of binary values for 10 parameters. Each row corresponds to one test, and each column gives the values for a particular parameter. It can be seen that any three columns in any order contain all eight possible combinations (000, 001, 010, 011, 100, 101, 110, 111) of the parameter values. Collectively, this set of tests will exercise all 3-way combinations of input values in only 13 tests, as compared with 1,024 for exhaustive coverage.

The primary goal in simulation is to study the behavior of the system with different input configurations. For example, a production line simulation may study the effects of changing line speed, interconnection between workstations, and buffer size on the number of items that can be produced per hour. A network simulation may investigate the effect of configurations on packet rate, delay, or potential for deadlock in the network.

	Parameters									
Tests	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	0	1	0	0	0	0	1
4	1	0	1	1	0	1	0	1	0	0
5	1	0	0	0	1	1	1	0	0	0
6	0	1	1	0	0	1	0	0	1	0
7	0	0	1	0	1	0	1	0	1	0
8	1	1	0	1	0	0	1	0	1	0
9	0	0	0	1	1	1	0	0	1	1
10	0	0	1	1	0	0	1	0	0	1
11	0	1	0	1	1	0	0	1	0	0
12	1	0	0	0	0	0	0	1	1	1
13	0	1	0	0	0	1	1	1	0	1

Figure 1. 3-way covering array for 10 parameters with 2 values each.

In this study we compare random and combinatorial testing of a network simulator, to determine if these two test approaches produce significantly different deadlock detection in the simulation. Using deadlocks as events of interest makes evaluating program responses straightforward and unambiguous. Numerical results such as packet rates or delays are not considered, but could be the subject of a future investigation. The two test modes – random or combinatorial – are compared using a standard two-tailed t-test for statistical significance.

2 Experimental Evaluation

This work investigates the hypothesis that combinatorial test suites will detect significantly more deadlocks than random test suites of the same size, for interaction strengths of t = 2, 3, 4.

Independent and Dependent Variables: The independent variable in this study is the type of testing used, either *t*-way combinatorial or random. The dependent variable is the number of deadlocks detected.

Subject Application and Test Suites: Software under test for the experiment was Simured (Pardo, 2005), a multicomputer network simulator developed at the University of Valencia. The software is available in C++ and Java versions, for both Linux and Windows. The core command line code (not including user interface or graphical display) consists of 2,131 lines of C++. Simured provides a simulation of the switching and routing layers for a multicomputer, allowing the user to study grid computer configurations to investigate the effect of topologies and configurable parameters on routing, timing, and other variables of interest. We used the C++ command line version of this software, compiled with gcc and run on 64-bit processors under Red Hat Enterprise Linux V4. No seeded faults or other modifications were made to the Simured software.

Simured provides a set of 14 parameters that can be set to a variety of values in a configuration file that is read by the simulator. Parameters and values used are shown in Table 1. Larger values are possible for a number of parameters, but would require extensive run time on a large system.

	Parameter	Values
1	DIMENSIONS	1,2,4,6,8
2	NODOSDIM	2,4,6
3	NUMVIRT	1,2,3,8
4	NUMVIRTINJ	1,2,3,8
5	NUMVIRTEJE	1,2,3,8
6	LONBUFFER	1,2,4,6
7	NUMDIR	1,2
8	FORWARDING	0,1
9	PHYSICAL	true, false
10	ROUTING	0,1,2,3
11	DELFIFO	1,2,4,6
12	DELCROSS	1,2,4,6
13	DELCHANNEL	1,2,4,6
14	DELSWITCH	1,2,4,6

Table 1. Simured configuration parameters and test values used.

Evaluation Metrics: Test suites were evaluated according to the number of deadlocks detected. We also compare the percentage of *t*-way combinations covered for the random test suites of equal size, and determine the number of random tests needed to provide 100% coverage of the respective *t*-way combinations. (By definition, a covering array provides 100% coverage of *t*-way combinations.)

Threats to Validity: Clearly there is a limitation on the extent to which these results can be generalized to other applications. While previous comparisons of combinatorial and random testing focused on fault detection, this study evaluates these methods with respect to deadlock detection in a simulation. Some implications of this difference are discussed in the analysis of results, in Section 4.2. A second difference is the nature of the software under test. Simured is a small but complex program that is not assumed to have characteristics similar to other application domains. Network simulation requires extensive calculations for statistics such as packet transmission rates and delays, and is not directly comparable to other types of software.

While the issues raised above should be considered in evaluating results, we believe that the experiment has identified a number of factors that can be usefully considered when deciding whether to use random or combinatorial testing for a particular problem.

3 Testing Procedure

Covering arrays that include all *t*-way combinations for $t = 2, 3$, and 4 were generated using the IPOG algorithm (Lei et al., 2007), which produces compact test suites. Test suites for the configuration shown in Table 1 included 28, 161, and 752 tests for $t = 2, 3$, and 4 respectively. Random test suites matching the sizes of the 2, 3, and 4-way combinatorial test suites were produced using the standard C library *rand()* function, producing one test at a time with a call to *rand()* for each variable value. In generating random test sets, the *rand()* function was initialized with a call to *srand()* to seed the pseudo-random number generator from the system clock. From these tests, configuration files were generated for Simured and the command line version of Simured invoked with each configuration file.

Each test set was executed for 500, 1000, 2000, 4000, and 8000-packet simulation runs. For combinatorial testing, one test suite run was conducted for each of the five packet counts and three interaction levels (28, 161, and 752 tests, for a total of 4,705 simulations). Random generation produces a different test set with each test generation run. For random testing, eight runs at each combination of packet count and interaction level were conducted (37,640 simulations), and the average deadlock detection calculated.

4 Results and Analysis

4.1 Test Results

Results for the two test modes were compared with a standard t-test for paired samples. Table 2 shows the number of deadlocks detected using tests produced from IPOG versus the average number of deadlocks detected with an equal number of randomly generated tests. Values for random test detection represent the average of eight runs with randomly generated tests at each combination of interaction level and packet count. Table 3 gives the two-tailed probability of a difference between the number of deadlocks detected by combinatorial and that by random testing.

For pairwise testing ($t = 2$), combinatorial testing detected slightly fewer deadlocks than an equal number of random tests, and the difference is statistically significant. At interaction strength $t = 3$ the difference between the two test methods is not statistically significant. At $t = 4$, however, the covering arrays produced by IPOG detected significantly more deadlocks than an equal number of random tests (see Table 3). In the next section we consider some possible reasons for the variation in effectiveness of these two test methods.

Deadlocks Detected – IPOG						
t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0	0	0	0	0
3	161	2	3	2	3	3
4	752	14	14	14	14	14
Average Deadlocks Detected – random						
t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0.63	0.25	0.75	0. 50	0. 75
3	161	3.00	3.00	3.00	3.00	3.00
4	752	10.13	11.75	10.38	13.00	13.25

Table 2. Deadlock detection, IPOG vs. random

Interaction strength	2-tailed probability
2	.0035
3	.1778
4	.0235

Table 3. t-test results for difference between random and IPOG generated tests

4.2 Analysis of Results

In considering explanations for the results, we first note that there can be a number of differences between the simulations conducted in this work and software testing in other application domains. In many applications, such as databases or web applications, different parameter values may result in different execution paths within the application, but the amount and complexity of processing is often similar for many different inputs. Network simulation, by contrast, may exhibit wide variations in processing depending on whether the input configuration is a small network of simple topology, or a large, complex one. This difference was observed in widely varying run times (not reported in this paper), and may also contribute to the distribution of deadlocks detected at the three interaction levels. Previous work (see Section 1) has found that increasing values of *t* detect progressively fewer faults, even in cases where combinatorial testing performed no better than random tests. Pairwise testing (*t*=2) often detected 70% to more than 90% of faults, while 3-way tests found roughly 10% to 20% of more faults, and 4-way to 6-way tests typically detected less than 5% more faults. This distribution is essentially reversed for the Simured testing (see Table 2), with 0%, 18%, and 82% of deadlocks detected at *t*=2, 3, and 4 respectively. This result is not unexpected. Faults can be triggered by combinations of any of the variables in a program. Even though a large set of variables may be directly or indirectly involved in triggering deadlocks, the set can be expected to be much smaller than the total number of variables in a program. With deadlocks occurring in roughly 2% of simulation runs, larger test sets would be expected to locate more deadlocks. Another significant difference between the simulation results and conventional testing is that a program can in most cases be expected to have a finite number of bugs. In conventional program testing, it is not surprising that increasing the number of tests applied against a fixed program with a fixed number of bugs can result in diminishing returns. Each new test is targeting an ever-decreasing number of undiscovered faults. With Simured however, configuration parameters make it possible to generate a nearly unlimited number of network configurations, so the ways in which deadlock can occur are similarly almost unlimited. Each new test has a high probability of generating a

previously untested configuration, so it is possible for the number of deadlocks to increase with larger test sets.

In addition to the "reverse" relationship between deadlock detection and interaction strength, another interesting finding was that pairwise tests detected slightly fewer deadlocks than the same number of random tests. Careful analysis shows that there is in fact a combinatorial explanation for this result, which we discuss in the remainder of this section.

Because a significant percentage of events can only be triggered by the interaction of two or more variables, one consideration in comparing random and combinatorial testing is the degree to which random testing covers particular *t*-way combinations. Any test set will also cover a certain proportion of possible *t*+1, *t*+2, etc. combinations as well. Table 4 gives the average percentage of *t*-way combinations covered by 100 randomly generated test sets of the same size as a *t*-way covering array generated by IPOG, for various combinations of *k = number of variables* and *v = number of values per variable*. Figures 2 through 6 summarize the coverage for arrays with variables of 2 to 10 values. As seen in the figures, the coverage provided by a covering array versus a random test suite of the same size varies considerably with different configurations. An important practical consideration in comparing combinatorial with random testing is the effectiveness of the covering array generator. Algorithms have a wide range in the size of covering arrays they produce, but all are designed to produce the smallest array possible that covers all *t*-way combinations. It is not uncommon for the better algorithms to produce arrays that are more than 50% smaller than other algorithms. We have found in comparisons with other tools that there is no uniformly "best" algorithm (Lei et al., 2007b). Algorithms vary greatly in the size of combinatorial test suites they produce, so the comparable random test suites will also vary in the number of tests. Random testing may produce results similar to combinatorial tests produced by an algorithm that generates a larger, sub-optimal covering array, because the correspondingly larger random test set has a greater probability of covering the *t*-way combinations.

A covering array algorithm that produces a compact array, i.e., with few tests, for *t*-way combinations may also include fewer (*t*+1)-way combinations because there are fewer tests. Tables 7 and 8 illustrate this phenomenon for the Simured experiment. Table 7 shows the percentage of *t*+1 up to *t*+3 combination coverage provided by the IPOG tests and in Table 8 the average coverage of an equivalent number of random tests. Although IPOG pairwise tests provide better 3-way coverage than the random tests, at higher values of *t*, the random tests are roughly the same or better in combination coverage than IPOG. Recall from Section 4.1 that pairwise combinatorial tests detected slightly fewer deadlocks than the equivalent number of random tests. One possible explanation may be that the superior 4-way and 5-way coverage (Table 7) of the random tests allowed detection of more deadlocks. Almost paradoxically, an algorithm that produces a larger, sub-optimal covering array may provide better fault detection because the larger array is statistically more likely to include *t*+1, *t*+2, and higher degree interaction tests as a byproduct of the test generation. This result demonstrates that the smallest possible array is not necessarily best for testing purposes if higher strength interactions are not also tested. It also suggests that covering array generation algorithms that fill "don't care" values (those for which all combinations have already been covered) with random values may provide better test results by covering a larger number of *t*+1, *t*+2, and higher degree combinations.

Now consider the size of a random test set required to provide 100% combination coverage. For most covering array algorithms, the difficulty of finding tests with high coverage increases as tests are generated. Thus even if a randomly generated test set provides better than 99% of the coverage of an equal sized covering array, it should not be concluded that only a few more tests are needed for the random set to provide 100% coverage. Table 5 gives the sizes of randomly generated test sets required for 100% combinatorial coverage at various configurations, and the ratio of these sizes to covering arrays computed with IPOG. Although there is considerable variation among configurations, note that the ratio of random to combinatorial test set size for 100% coverage exceeds 3 in most cases, with average ratios of 3.9, 3.8, and 3.2 at t = 2, 3, and 4 respectively. Thus combinatorial testing offers a significant advantage over random testing if the goal is 100% combination coverage.

Vars	Values/ Variable	IPOG 2-way tests	Random 2-way coverage	IPOG 3-way tests	Random 3-way coverage	IPOG 4-way tests	Random 4-way coverage
10	2	10	94.1	20	94.3	42	93.2
10	4	30	84.6	151	90.6	657	92.3
10	6	66	85.6	532	91.6	3843	94.8
10	8	117	83.8	1214	90.6	12010	94.7
10	10	172	82.1	2367	90.6	29231	94.6
15	2	10	93.9	24	96.2	58	97.5
15	4	33	88.1	179	94.1	940	97.5
15	6	77	88.6	663	95.4	5243	98.2
15	8	125	86.1	1551	95.2	16554	98.2
15	10	199	86.4	3000	95.0	40233	98.2
20	2	12	96.5	27	97.3	66	98.6
20	4	37	90.9	209	96.2	1126	98.8
20	6	86	91.3	757	97.0	6291	99.2
20	8	142	91.3	1785	96.9	19882	99.2
20	10	215	88.4	3463	96.9	48374	99.2
25	2	12	95.9	30	98.5	74	99.2
25	4	39	92.1	233	97.5	1320	99.4
25	6	89	91.8	839	97.9	7126	99.6
25	8	148	90.3	1971	97.9	22529	99.6
25	10	229	90.0	3823	97.8	54856	99.6

Table 4. Average percent of t-way combinations covered by equal number of random tests

Vars	Values	2-way Tests			3-way Tests			4-way Tests		
		IPOG Tests	Random Tests	Ratio	IPOG Tests	Random Tests	Ratio	IPOG Tests	Random Tests	Ratio
10	2	10	18	1.80	20	61	3.05	42	150	3.57
10	4	30	145	4.83	151	914	6.05	657	2256	3.43
10	6	66	383	5.80	532	1984	3.73	3843	13356	3.48
10	8	117	499	4.26	1214	5419	4.46	12010	52744	4.39
10	10	172	808	4.70	2367	11690	4.94	29231	137590	4.71
15	2	10	20	2.00	24	52	2.17	58	130	2.24
15	4	33	121	3.67	179	672	3.75	940	2568	2.73
15	6	77	294	3.82	663	2515	3.79	5243	17070	3.26
15	8	125	551	4.41	1551	6770	4.36	16554	60568	3.66
15	10	199	940	4.72	3000	15234	5.08	40233	159870	3.97
20	2	12	23	1.92	27	70	2.59	66	140	2.12
20	4	37	140	3.78	209	623	2.98	1126	3768	3.35
20	6	86	288	3.35	757	2563	3.39	6291	18798	2.99
20	8	142	630	4.44	1785	8450	4.73	19882	59592	3.00
20	10	215	1028	4.78	3463	14001	4.04	48374	157390	3.25
25	2	12	34	2.83	30	70	2.33	74	174	2.35
25	4	39	120	3.08	233	790	3.39	1320	3520	2.67
25	6	89	327	3.67	839	2890	3.44	7126	19632	2.75
25	8	148	845	5.71	1971	7402	3.76	22529	61184	2.72
25	10	229	1031	4.50	3823	16512	4.32	54856	191910	3.50
Ratio Average:				3.90			3.82			3.21

Table 5. Size of random test set required for 100% *t*-way combination coverage.

Values per variable	Ratio, 2-way	Ratio, 3-way	Ratio, 4-way
2	2.14	2.54	2.57
4	3.84	4.04	3.04
6	4.16	3.59	3.12
8	4.70	4.33	3.44
10	4.68	4.59	3.86

Table 6. Average ratio of random/IPOG for covering arrays by values per variable, variables = 10, 15, 20, 25

The comparisons between random and combinatorial testing detailed in Tables 4 – 6 suggest a number of tentative conclusions:

- *For binary variables (v=2), random tests compare reasonably well with covering arrays* (94% to 99% coverage) for all three values of *t* (Table 4, Figure 2). Thus random testing for a system under test (SUT) with all or mostly binary variables may compare favorably with combinatorial testing. This factor may explain results in previous studies in which combinatorial testing performed no better than random testing. All variables of the application used in (Bryce et al., 2006) were binary, and comparably sized random test suites covered a high number of combinations. For one of the two applications studied in (Schroeder, 2004), 16 of 18 variables were binary and 18 of 19 variables for the other were either binary or 3-valued. The percentages of combinations covered by random test suites for

binary variables in Table 4 are similar to the coverage reported in (Schroeder, 2004), but are somewhat higher than the coverage in (Bryce et al., 2006), probably as a result of different covering array algorithms. Kobayashi et al. (2001) compared combinatorial and random testing for logic testing (thus 2-valued variables) and found combinatorial testing superior on average for 2-way, 3-way, and 4-way tests. However, for 2-way combinations nearly half of the random test suites performed as well or better than the corresponding 2-way combinatorial tests.

- *Combination coverage provided by random generation of the equivalent number of pairwise tests at (t = 2) decreases as the number of values per variable increases*, and the coverage provided by pairwise testing is significantly less than 100% (Table 4, Figures 2 - 6). The effectiveness of random testing relative to pairwise testing should be expected to decline as the average number of values per variable increases.

- *For 4-way interactions, coverage provided by random test generation increases with the number of variables* (Table 4). Thus combinatorial testing should be significantly more effective at fault detection for a module with approximately 10 variables than random testing, while the difference between the two test methods should be less for modules with 20 or more variables.

- *The combination coverage advantage of combinatorial testing relative to random testing decreases at higher interaction levels* (Table 4, Figures 2 – 6). For example, with 15 variables of 6 values each, random tests provide roughly 88% coverage of 2-way combinations, increasing to 98% coverage for 4-way combinations. Note however that this does not mean the random test set will be almost as effective as the combinatorial set. A random test set must still be approximately 4 times the size of the corresponding combinatorial set to provide 100% combination coverage (Table 5).

- *For 100% combination coverage, the advantage of combinatorial testing varies directly with the number of values per variable and inversely with the interaction strength t* (Table 6). Figure 7 illustrates how these factors (interaction strength t and values per variable v) combine: the ratio of random/combinatorial coverage is highest for 10 variables with $t = 2$, but declines for other pairings of t and v. Random testing cannot assure any pre-set level of desired coverage while covering arrays by definition achieve 100% combination coverage. Random testing is significantly less efficient than combinatorial testing, requiring 2 to nearly 5 times as many tests as a covering array to obtain complete coverage (Tables 6). Thus if 100% combination coverage is desired, combinatorial testing should be significantly less expensive than random test generation.

Note also that the number of faults in the SUT can affect the degree to which random testing approaches combinatorial testing effectiveness. For example, suppose the random test set covers 99% of combinations for 4-way interactions, and the SUT contains only one 4-way interaction fault. Then there is a 99% probability that the random tests will contain the 4-way interaction that triggers this fault. However, if the SUT contains m independent faults, then the probability that combinations for all m faults are included in the random test set is $.99^m$. Hence with multiple faults, random testing may be significantly less effective, as its

probability of missing at least one the *m* combinations that detect these faults will be $1 - c^m$, for *c* = percent coverage and *m* = number of independent faults.

t-way	2-way coverage	3-way coverage	4-way coverage	5-way coverage	Average coverage
2	1.00	.758	.429	.217	0.601
3	1.00	1.00	.924	.709	0.908
4	1.00	1.00	1.00	.974	0.994

Table 7. Combination coverage of IPOG t-way tests

Same size as t-way	2-way coverage	3-way coverage	4-way coverage	5-way coverage	Average coverage
2	.940	.735	.499	.306	0.620
3	1.00	.942	.917	.767	0.906
4	1.00	1.00	.965	.974	0.985

Table 8. Combination coverage of random tests

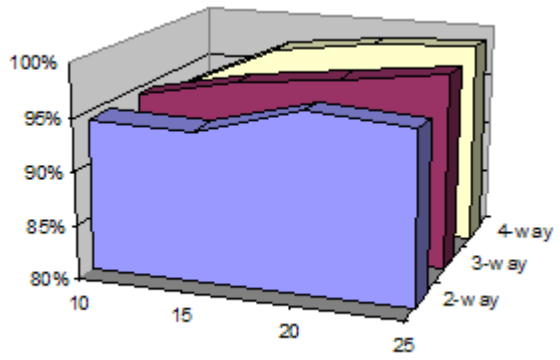

Figure 2. Random coverage of *t*-way combinations for v=2.

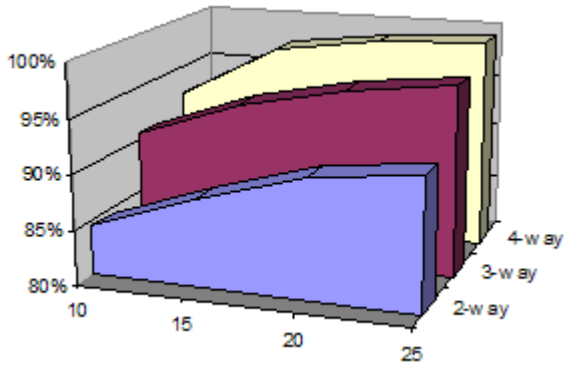

Figure 3. Random coverage of *t*-way combinations for v=4.

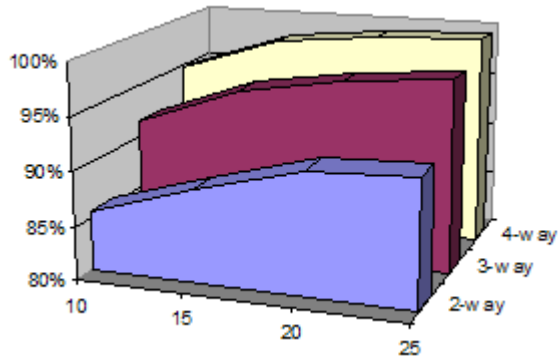

Figure 4. Random coverage of *t*-way combinations for v=6.

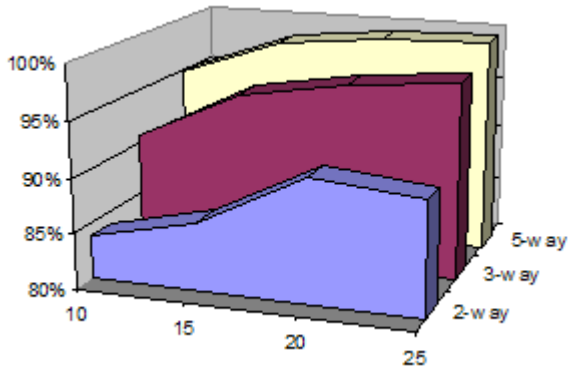

Figure 5. Random coverage of *t*-way combinations for v=8.

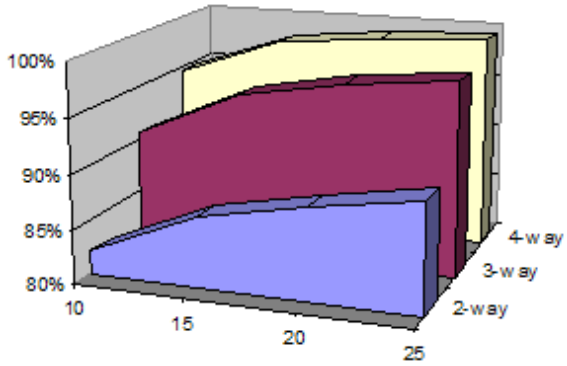

Figure 6. Random coverage of *t*-way combinations for v=10

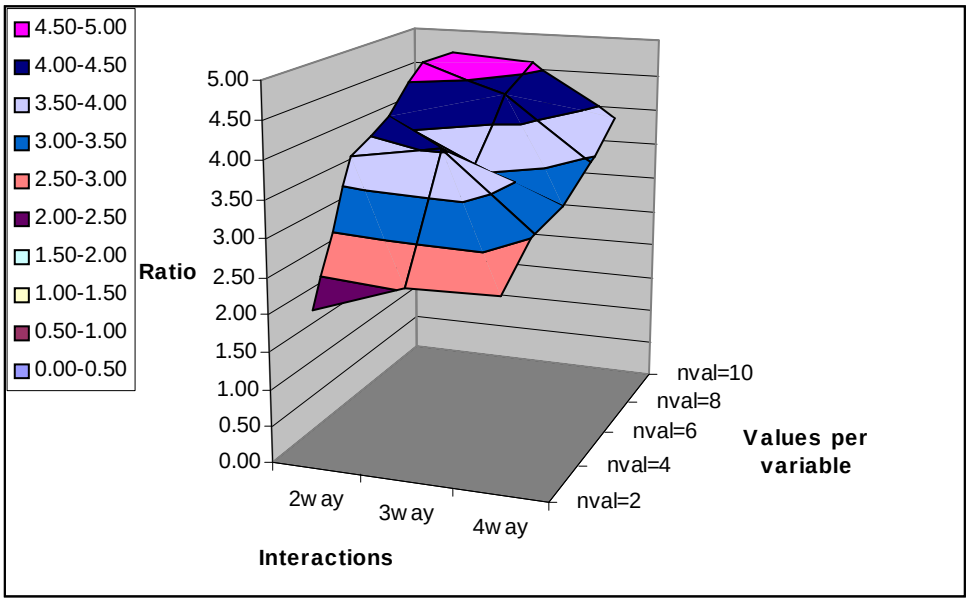

Figure 7. Average ratio of random/IPOG for covering arrays by values per variable

5 Conclusions

For the simulation program tested in this study, pairwise tests detected slightly fewer deadlocks than an equal number of random tests, but 4-way combinatorial testing produced better results than an equal number of random tests. Analyzing the random test sets suggests a number of reasons for these results. Although pairwise tests covered all 2-way combinations and an equal number of random tests covered fewer, the random tests covered more 4-way and 5-way combinations, and thus had a greater probability of triggering deadlocks that depended on 4-way or 5-way interactions. However, the 4-way combinatorial tests covered significantly more 4-way combinations (100% vs. 96%) and also provided equal 5-way coverage compared with the corresponding random test set, and found more deadlocks as well.

This result demonstrated that the smallest possible array is not necessarily best for testing purposes if higher strength interactions are not also tested. When using *t*-way combinatorial testing, it can be helpful to evaluate the test set for coverage of *t*+1 and higher interaction strengths. Methods of combining combinatorial and random tests may also be effective, as proposed in Bell (2006) and Bell, Vouk (2006). These results also suggest that covering array algorithms may provide better test results by filling "don't care" values with random (rather than constant, sequential, or other non-random) values.

Disclaimer: Reference to commercial products or trademarks does not imply endorsement by NIST or any other agency of the US Government, nor that such products are necessarily best suited to any purpose.

References

J. Bach, P. Shroeder, Pairwise Testing - A Best Practice That Isn't. Proceedings of 22nd Pacific Northwest Software Quality Conference, 2004, pp. 180-196

Kera Z. Bell and Mladen A. Vouk. On effectiveness of pairwise methodology for testing network-centric software. Proceedings of the ITI Third IEEE International Conference on Information & Communications Technology, pages 221–235, Cairo, Egypt, December 2005.

K.Z. Bell, Optimizing Effectiveness and Efficiency of Software Testing: a Hybrid Approach, PhD Dissertation, North Carolina State University, 2006.

R. Bryce, A. Rajan, M.P.E. Heimdahl, Interaction Testing in Model Based Development: Effect on Model Coverage, IEEE, 13th Asia Pacific Software Engineering Conference (APSEC'06) pp. 259-268.

K. Burr and W. Young, Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Test Coverage, International Conference on Software Testing, Analysis, and Review (STAR), San Diego, CA, October, 1998

K. Burroughs, A. Jain, and R. L. Erickson. Improved quality of protocol testing through techniques of experimental design. In Proceedings of the IEEE International Conference on Communications (Supercomm/ICC'94), May 1-5, New Orleans, Louisiana, USA. IEEE, May 1994, pp. 745-752

D. M. Cohen, S. R. Dalal, J. Parelius, G. C. Patton The Combinatorial Design Approach to Automatic Test Generation IEEE Software, Vol. 13, No. 5, pp. 83-87, September 1996

M. Ellims, D. Ince, M. Petre, The Effectiveness of T-Way Test Data Generation, SAFECOMP 2008, pp. 16-29, Springer Verlag.

I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, A. Iannino. Applying design of experiments to software testing Proceedings of the Intl. Conf. on Software Engineering, (ICSE '97), 1997, pp. 205-215, New York

Kuhn, D. R., D. Wallace, and A. Gallo, "Software Fault Interactions and Implications for Software Testing," IEEE Transactions on Software Engineering, 30(6):418-421, 2004.

Kuhn, D. R. and V. Okun, "Pseudo-exhaustive Testing for Software," Proceedings of 30th NASA/IEEE Software Engineering Workshop, pp. 153-158, 2006.

D.R. Kuhn, M.J. Reilly, An Investigation of the Applicability of Design of Experiments to Software Testing, 27th NASA/IEEE Software Engineering Workshop, NASA Goddard Space Flight Center, 4-6 December, 2002 .

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun, J. Lawrence, "IPOG/IPOG-D: Efficient Test Generation for Multi-Way Combinatorial Testing", Software Testing, Verification, and Reliability. (Published Online: Nov 29 2007, DOI: 10.1002/stvr.381)

Y.Lei, R. Kacker, D.R. Kuhn, V. Okun, J. Lawrence, "IPOG - a General Strategy for t-way Testing", IEEE Engineering of Computer Based Systems conference, 2007.

Kobayashi, N., T. Tsuchiya, T. Kikuno, "Applicability of Non-Specification Based Approaches to Logic Testing for Software", *Proceedings of the 2001 International Conference on Dependable Systems and Networks*, IEEE, pp. 337 – 346.

F. Pardo, JSimured - Simulador de Redes de Multicomputadores Paralelo, University of Valencia, May, 2005. http://simured.uv.es/doc/memoria.pdf

Alexander Pretschner, Tejeddine Mouelhi, Yves Le Traon. Model Based Tests for Access Control Policies, *2008 International Conference on Software Testing, Verification, and Validation* pp. 338-347

Patrick J. Schroeder, Pankaj Bolaki, and Vijayram Gopu. Comparing the fault detection effectiveness of n-way and random test suites. In Proceedings of the IEEE International Symposium on Empirical Software Engineering, pages 49–59, 2004.

Wallace, D. R. and D. R. Kuhn, "Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data," International Journal of Reliability, Quality and Safety Engineering, 8(4):351-371, 2001.

A.W. Williams, R.L. Probert. A practical strategy for testing pair-wise coverage of network interfaces The Seventh International Symposium on Software Reliability Engineering (ISSRE '96) p. 246