

A Density-Based Greedy Algorithm for Higher Strength Covering Arrays

Renée C. Bryce
Computer Science
University of Nevada at Las Vegas
Las Vegas, NV 89154-4019, U.S.A.
reneebruce@cs.unlv.edu

Charles J. Colbourn
Computing and Informatics
Arizona State University
Tempe, AZ 85287-8809, U.S.A.
colbourn@asu.edu

March 26, 2008

Abstract

Algorithmic construction of software interaction test suites has focussed on pairwise coverage; less is known about the efficient construction of test suites for t -way interactions with $t \geq 3$. This work extends an efficient density-based algorithm for pairwise coverage to generate t -way interaction test suites, and show that it guarantees a logarithmic upper bound on the size of the test suites as a function of the number of factors. To complement this theoretical guarantee, an implementation is outlined and some practical improvements are made. Computational comparisons with other published methods are reported. Many of the results improve upon those in the literature. However, limitations on the ability of one-test-at-a-time algorithms are also identified.

Keywords: Covering arrays, greedy algorithm, mixed-level covering arrays, t -way interaction coverage, software interaction testing

1 Introduction

Software interaction testing is a means of identifying faulty interactions among factors; in this approach, all t -way interactions are tested at least once. For example, a small component-based system with four components (factors) is shown in Table 1. Each component has two possible levels from which to select. This input has 24 2-way interactions among components and 32 3-way interactions. Six tests (rows) are needed to cover all pairwise interactions, shown in Table 2. Two additional tests cover all 3-way interactions.

Interaction testing has been widely studied; see [15] for a comprehensive survey. It has been used to provide fault localization [35]. Moreover, t -way interaction coverage for “small” t reduces redundancy in block coverage while still identifying many faults [14]. Code coverage

Component	f_0	f_1	f_2	f_3
Option 1	0	0	0	0
Option 2	1	1	1	1

Table 1: A small component based system with 4 components that have 2 options each.

Test No.	f_0	f_1	f_2	f_3
1	0	1	1	1
2	1	0	0	0
3	1	1	1	0
4	0	0	0	1
5	1	1	0	1
6	0	0	1	0
7	0	1	0	0
8	1	0	1	1

Table 2: Pairwise coverage is achieved within the first 6 tests and 3-way coverage within 8.

of 93% was achieved [5] using interaction testing to test an e-mail system at Nortel, even uncovering new defects in a mature system. Interaction testing has been used for a system that identifies both real and false targets [1]. Interaction testing has also been applied to both hardware and software systems for strength $2 \leq t \leq 6$ [21, 22].

In all of these studies, interaction testing provides a systematic approach to analyze the main effects of factors and their interaction with other factors. It also provides a significant reduction in the number of tests while identifying and isolating faults [5, 14, 23, 35].

In empirical studies, Kuhn et al. [21, 22] report that testing of 2-way to 6-way interactions can be valuable in fault detection. Yet almost all algorithms to date focus on pairwise, or 2-way, interactions (see [12, 16]). However, some work to construct interaction test suites of strength at least three has been done. Combinatorial solutions are known for t -way coverage in general [25, 26, 30] and for $t \in \{3, 4\}$ in particular [6, 10, 13]. Simulated annealing, constraint programming, and tabu search have reported results for up to 3-way coverage [10, 17, 27]. AETG has published results for up to 3-way coverage [6]. Combinatorial Test Services (CTS) [16] and the newer incarnation ITCH [18] are implemented for t -way coverage, with results published for $t \leq 4$. IPO [31] has been extended from pairwise testing to higher strength in the testing tool `FireEye` [24]. Two further tools available for execution, about which no algorithmic description appears to be available, are `jenny` [19] and `TVG` [33]. Finally, Kuhn et al. [20] have devised a fast random method for generating large test suites of high strength.

None of the published techniques provides a guarantee on the size of the test suite, despite often yielding useful test suites. The work in this paper develops a practical algorithm that has an accompanying theoretical guarantee on the number of tests produced. For pairwise

testing, TConfig [34] provides such a guarantee by exploiting combinatorial techniques, while DDA [3] develops a one-test-at-a-time greedy method with a guarantee. The algorithm in this paper generalizes the latter to higher strength.

Software interaction test suites are often represented by covering arrays; see [12, 16] for surveys. A *covering array*, $CA(N; t, k, v)$, is an $N \times k$ array where N is the number of tests, t is the strength of coverage, k is the number of factors, and v is the number of levels or values of each factor. Each row of a covering array corresponds to a test, and in every $N \times t$ sub-array, each t -tuple appears at least once. For instance, consider the example input in Table 1 and the corresponding interaction test suite in Table 2. The input in Table 1 includes $k=4$ factors that can each take on $v=2$ levels. The interaction test suite in Table 2 is of strength $t=3$ and contains $N=8$ tests. Test 1 covers four 3-tuples: $(f_0=0, f_1=1, f_2=1)$, $(f_0=1, f_1=1, f_3=1)$, $(f_0=0, f_2=1, f_3=1)$, and $(f_1=1, f_2=1, f_3=1)$.

We establish some notation. Let \mathcal{F} be the set of k factors (indices of the columns), and for each $f \in \mathcal{F}$ let V_f be the set of values for factor f . A t -tuple (representing a t -way interaction) is a set of t factors, and a value for each factor in the set; there are then $\binom{k}{t} v^t$ t -way interactions, each of which is to appear in at least one test. For any s -way interaction S , let $\phi(S)$ denote the set of factors that S contains.

It has been established that, for fixed t and v , the number N of tests in a minimum size $CA(N; t, k, v)$ satisfies $N = O(\log k)$ [3]. The lower bound is trivial, since every row must be distinct. There are many proofs for the upper bound. Probabilistic methods [28, 29] establish the asymptotic growth rate but do not yield a deterministic algorithm (at least directly). Two explicit algorithms that generate one test at a time ensure logarithmic growth [7, 8]; however, as described, both necessitate the examination of an exponential number of tests. Finally, recursive methods [25, 26] could be adapted to develop algorithms to generate a covering array in which the logarithmic growth is guaranteed. However these recursions generate an entire array rather than constructing a single test at a time.

The requirement for efficient generation of covering arrays arises naturally in the software testing domain, and the requirement for logarithmic growth of the number of tests is also fundamental to the effective testing of complex systems. Additionally, tests in the test suite must be generated one test at a time. This recognizes an essential fact in the testing application. Although in principle testing is done nonadaptively and all tests are run, in practice, test budgets can limit the amount of testing to only an initial portion of the test suite, and therefore maximizing coverage within the first tests is desirable.

The rest of the paper is organized as follows. Section 2 presents a one-test-at-a-time greedy algorithm for t -way coverage and establishes that the method is efficient and yields a number of tests that grows logarithmically with the number of factors. Section 3 generalizes this method to obtain a family of more practical algorithms. Section 4 treats a number of variants of the basic method, and compares sizes of test suites with those available in the literature. Section 5 summarizes the results.

2 A Greedy Algorithm for Higher Strength

This section introduces a one-test-at-a-time greedy algorithm for constructing covering arrays of any strength t ; this extends a method for pairwise coverage [3]. The basic strategy is to start with an empty array and add one test (row) at a time. Throughout, a record is kept for each t -way interaction as to whether it has been already covered, or remains to be covered, by setting the indicator variable $\gamma_T = 1$ when the t -tuple T is uncovered, 0 otherwise.

The fundamental operation in the method is the efficient selection of a next test that covers *at least the average* number of uncovered t -tuples, where the average is taken over all possible tests. Naturally one cannot enumerate all tests in order to meet this average!

Initially in constructing a test, each factor is *free* to take on any of the admissible values. Next, determine a value $\nu \in V_f$ for each factor f in turn, making factor f *fixed* to value ν and setting $V_f = \{\nu\}$. In fixing a factor, one must avoid forcing a test to be constructed that has fewer newly covered t -tuples than the average.

So suppose that zero or more factors are fixed (for each, the only admissible value is now the one already selected). Let $R \subseteq \mathcal{F}$ be a set of $s \leq t$ factors and $S = \{(r, \nu) : r \in R, \nu \in V\}$ be an s -way interaction based on the factors in $R = \phi(S)$. Then for every set W of t factors satisfying $\phi(S) \subseteq W \subseteq \mathcal{F}$ define $E(S, W) = \{S \cup \{(w, v_w) : w \in W \setminus \phi(S), v_w \in V_w\}\}$. That is, $E(S, W)$ contains all t -way interactions that include the s -way interaction S using available levels for each of the remaining $t - s$ factors in $W \setminus \phi(S)$. The number of such t -way interactions is $\pi(S, W) = \prod_{f \in W \setminus \phi(S)} |V_f|$.

Compute the *density of S with respect to W* , $\delta(S, W) = \frac{\sum_{T \in E(S, W)} \gamma_T}{\pi(S, W)}$. Then $0 \leq \delta(S, W) \leq 1$; indeed it is the probability that, when a value for each factor of W is selected uniformly at random from the sets of admissible values that remain, the corresponding t -way interaction is uncovered. Alternatively it is the fraction of currently uncovered t -way interactions that extend the current s -way interaction S to the specified t factors of W over the total number of such extensions. Extend this definition to all t -way interactions by defining the *density of S* , $\delta(S) = \sum_{\phi(S) \subseteq W \subseteq \mathcal{F}} \delta(S, W)$. Then $0 \leq \delta(S) \leq \binom{k-s}{t-s}$; when values are selected uniformly at random from the admissible values that remain for each factor, the density is the *expected number* of sets of t factors that contain all factors of S and underlie an uncovered t -way interaction.

Now choose a factor $f \notin \phi(S)$. Write

$$\delta(S) = \sum_{\phi(S) \cup \{f\} \subseteq W \subseteq \mathcal{F}} \delta(S, W) + \sum_{\phi(S) \subseteq W \subseteq \mathcal{F} \setminus \{f\}} \delta(S, W). \quad (1)$$

The first summation on the right hand side is the *factor density* $\delta_f(S)$ for factor f and interaction S . A key observation is that

$$\sum_{\phi(S) \cup \{f\} \subseteq W \subseteq \mathcal{F}} \delta(S, W) = \delta_f(S) = \frac{1}{|V_f|} \sum_{\nu \in V_f} \delta(S \cup \{(f, \nu)\}). \quad (2)$$

Let $Z = \emptyset$ and all factors be free. Assume that some t -way interactions may be covered, but at least one is not. Then initially $\Delta_0 = \delta(Z)$ is the expected number of t -way interactions that a randomly chosen test covers. Now one factor at a time is fixed, as follows.

For $i = 0, \dots, k - 1$:

1. Choose any free factor f , computing $\delta_f(Z) = \frac{\sum_{\nu \in V_f} \delta(\{(f, \nu)\})}{|V_f|}$.
2. Choose a value $\nu \in V_f$ for which $\delta(\{(f, \nu)\}) \geq \delta_f(Z)$; that is, choose a value for f that makes at least the average contribution to the factor density.
3. Set $\Delta_{i+1} = \Delta_i + \delta(\{(f, \nu)\}) - \delta_f(Z)$.
4. Fix factor f to value ν , setting $V_f = \{\nu\}$ in the process.

By definition, $0 \leq \Delta \leq \binom{k}{t}$; indeed it is the sum, over all selections of t factors, of the fraction of t -way interactions on these factors that remain uncovered. This proves a simple lemma:

Lemma 2.1 *Suppose that the density before a test is selected is at most Δ , and that the next test is selected one factor at a time always selecting a value of at least average density. Then the addition of this test reduces the density to at most $\Delta \frac{v^t - 1}{v^t}$.*

Proof. In order to reduce the density by $\frac{\Delta}{v^t}$, it suffices to prove that each new test selected covers at least Δ previously uncovered t -way interactions. This can be proved inductively by showing that $\Delta_{i+1} \geq \Delta_i$ for $0 \leq i < k$. Indeed Δ_k is the number of t -way interactions that were uncovered but are covered by the selected test, while Δ_0 is the expected number covered by a test chosen at random.

Suppose then that the $(i+1)$ st factor to be fixed is f , and the value chosen is ν . Calculate $\Delta_{i+1} - \Delta_i = \delta(\{(f, \nu)\}) - \delta_f(Z)$, having fixed only the first i factors in this calculation. Because ν was chosen so that $\delta(\{(f, \nu)\}) \geq \delta_f(Z)$, then $\Delta_{i+1} \geq \Delta_i$. ■

Theorem 2.2 *When t and v are fixed, there is an efficient one-test-at-a-time algorithm for constructing a covering array $CA(N; t, k, v)$ in which N is $O(\log k)$.*

Proof. The bound on the size of the covering array is obtained by observing that, when no tests are present, $\Delta = \binom{k}{t}$. After j tests have been selected Lemma 2.1 establishes that $\Delta \leq \binom{k}{t} \left(\frac{v^t - 1}{v^t}\right)^j$. When $\Delta < \frac{1}{v^t}$, the covering array is completed. Minimizing j subject to $v^t \binom{k}{t} \left(\frac{v^t - 1}{v^t}\right)^j < 1$ yields the smallest number N of tests that guarantee success, and hence $N = O(\log k)$ for fixed v and t .

The issue to be treated is that of efficiency. Since the number of tests generated is $O(\log k)$, it suffices to show that each test is selected efficiently. For each of the k factors, $O(v^t \binom{k-1}{t-1})$ densities are updated when a value is selected. For v and t fixed, this is bounded by a polynomial in k . ■

Indeed the proof of Theorem 2.2 ensures that after M tests are selected, the number of uncovered t -tuples is at most $v^t \binom{k}{t} \left(\frac{v^t - 1}{v^t}\right)^M$. Hence it establishes the following:

Theorem 2.3 *In order to cover all but αv^t t -way interactions, it suffices to employ at most*

$$\frac{\log \binom{k}{t} - \log(\alpha + 1)}{\log(v^t) - \log(v^t - 1)}$$

tests.

This provides a strong guarantee that within the first tests, a large number of t -way interactions are covered. Arguably, this can be more crucial than the ultimate size of the test suite, and it appears not to be ensured by methods that construct the entire array without greedily selecting one test at a time.

3 One-Test-at-a-Time Greedy Methods

For pairwise testing, greedy algorithms have been well studied [3, 2, 4, 7, 31, 32]. A framework for one-test-at-a-time greedy algorithms generalizes instantiations of such algorithms [4]. This framework is summarized in Figure 1 and can be described at a high level. The overall goal in constructing a covering array is to create a 2-dimensional array in which all t -way interactions are covered. Using a greedy approach, this collection is built one test at a time by fixing each factor with a level (value). The order in which factors are fixed may vary, as can the scheme for choosing levels. A test may be selected from multiple candidates. When more than one candidate is permitted, several tests are constructed and the best one is chosen to add to the covering array. Once all t -tuples have been covered, the covering array is complete. When the goal is to construct the smallest covering array possible, if random selections occur, the covering array can be regenerated numerous times to select the best result.

The density measures fit into the framework and provide a family of density-based methods for generating covering arrays. This includes instantiating the four layers of the framework: *repetitions*, *multiple candidates*, *factor ordering*, and *level selection*. In addition, several tie-breaking rules are defined.

As introduced, one main limitation is that every factor has the same number of levels. A *mixed level covering array* permits factors to have different numbers of levels. The maximum number of levels of any factor is $v = \max_{f \in \mathcal{F}} |V_f|$. So at the outset simply mark as covered all t -tuples that contain an inadmissible value for one or more factors. Indeed t -tuples that need not be covered for any reason can be so marked; this provides a simple method for *seeding* an array with a specified set of tests, by just marking all t -tuples in the seed as covered at the outset. It would be possible instead to adjust the densities calculated to take into account the actual number of levels for each factor. This is not done here because in consequence it weights more heavily those t -way interactions involving factors with the fewest levels, yet these can be expected to be the easiest to cover. By treating every factor as if it had the maximum number of levels, all t -way interactions are equally weighted in the density calculation.

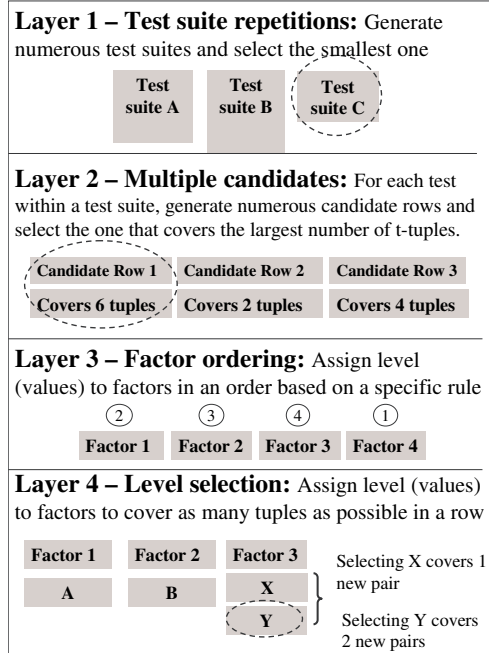


Figure 1: The framework for one-test-at-a-time greedy algorithms to generate interaction test suites.

To generate a test of a covering array, each factor is assigned an admissible level (value) by selecting a value yielding the largest increase in density. When $t > 2$, the overhead in determining this level is substantial, although still polynomially bounded. To obtain a more practical implementation, consider experiments with restricting densities as follows. In examining the factor density for the factor whose value is being selected, there are $\binom{k-1}{t-1}$ other sets of factors making a contribution. Among these, it may happen that the $t - 1$ factors are all fixed at this time, all free, or any mixture. Contributions from free factors are *expected*, while those from fixed factors are *ensured*. It appears reasonable, then, to concentrate on the contributions of the fixed factors to the density. While this negates the proof of the logarithmic guarantee, as a practical matter it is a worthwhile variant. In selecting factor f , call a choice of $t - 1$ other factors ℓ -restricted if at most ℓ of the $t - 1$ factors are currently free. Then ℓ -restricted densities are defined by summing only over ℓ -restricted sets of $t - 1$ other factors. At first this may seem artificial, but the extreme case of 0-restricted densities is precisely the commercial AETG method [7]. Indeed in that case, the selection of a level for a factor is made so as to maximize the number of newly covered t -tuples with reference only to the already fixed factors.

The following experiment uses different restrictions on density, employing three repetitions, three candidates for each test, and ordering factors by restricted density after an initial best $(t - 1)$ -tuple (as explained later) is chosen. Results are reported in Table 3. The inputs are in the first column and the results of the algorithm using different restricted densities follow. For instance, the results for input 2^5 (read as 5 factors have 2 levels each) show

that 0-restricted density produces the smallest size covering array. As the number of factors increases, the improvement obtained by using 1-restricted rather than 0-restricted densities is striking. However, the extension to 2-restricted and unrestricted density is disappointing. Although computation time is increased, the sizes obtained are in general not substantially better than those with 1-restricted density. A plausible explanation is that in the selections of values for the initial factors, the contributions involving fixed factors are overwhelmed by those involving free factors, with the consequence that too little weight is assigned to ensuring that t -tuples are covered among the fixed factors.

	0-res	1-res	2-res	unres		0-res	1-res	2-res	unres
2^5	16	17	17	17	2^6	28	26	25	25
2^7	29	30	32	31	2^8	33	32	24	31
2^9	32	37	31	36	2^{10}	40	34	34	39
3^5	101	105	105	105	3^6	135	135	137	137
3^7	160	159	160	158	3^8	180	180	180	176
3^9	204	199	204	195	3^{10}	226	211	213	213
4^5	319	329	329	329	4^6	425	427	425	425
4^7	508	502	504	505	4^8	582	570	570	569
4^9	645	632	633	633	4^{10}	712	688	689	686
5^5	790	805	805	805	5^6	1017	1024	1033	1033
5^7	1224	1219	1214	1213	5^8	1408	1383	1381	1383
5^9	1565	1535	1533	1534	5^{10}	1724	1675	1675	1671
6^5	1646	1682	1682	1682	6^6	2089	2096	2111	2111
6^7	2509	2490	2490	2494	6^8	2872	2839	2832	2843
6^9	3224	3155	3152	3153	6^{10}	3529	3450	3452	3451

Table 3: Sizes of test suites generated with strength $t = 4$ for restricted densities (3 candidates, 3 repetitions, factor ordering by best tuple plus restricted density, level selection by restricted density).

The next layer in the framework is factor ordering. This is the order in which factors are assigned levels. It is possible to implement many different factor ordering rules using each of the four restrictions on density as a level selection rule.

The use of ℓ -restricted densities for $\ell < t - 1$ raises the question of how to proceed initially when fewer than $t - \ell - 1$ factors are fixed. AETG appears to resolve this (for 0-restricted densities) by choosing initially a t -tuple that is uncovered and setting the corresponding t factors to the levels assigned in the chosen tuple. A different approach is to select the first $t - \ell - 1$ factors and their values as follows. Initially set $S = \emptyset$. While $|S| < t - \ell$, first choose a factor f among free factors that maximizes $\delta_f(S)$, and choose a value ν for factor f to maximize $\delta(S \cup \{(f, \nu)\})$; set factor f to value ν , and adjoin $\{(f, \nu)\}$ to S . Once $t - \ell - 1$ factors are fixed, ℓ -restricted densities are well defined.

Figure 2 shows an example trace of the algorithm that computes factor and level density

Input				
f_0	f_1	f_2	f_3	f_4
0	2	4	6	8
1	3	5	7	9

Assume that the 1st test is selected and that the 2nd test is under construction.

	f_0	f_1	f_2	f_3	f_4
Test 1	0	2	4	6	8
Test 2	?	?	?	?	?

Test 1 covers 10 out of the 40 possible 3-tuples for this input.

3-tuples to cover for this input						
(0,2,4)	(0,4,6)	(1,2,4)	(1,4,6)	(2,4,6)	(3,4,6)	(4,6,8)
(0,2,5)	(0,4,7)	(1,2,5)	(1,4,7)	(2,4,7)	(3,4,7)	(4,6,9)
(0,2,6)	(0,4,8)	(1,2,6)	(1,4,8)	(2,4,8)	(3,4,8)	(4,7,8)
(0,2,7)	(0,4,9)	(1,2,7)	(1,4,9)	(2,4,9)	(3,4,9)	(4,7,9)
(0,2,8)	(0,5,6)	(1,2,8)	(1,5,6)	(2,5,6)	(3,5,6)	(5,6,8)
(0,2,9)	(0,5,7)	(1,2,9)	(1,5,7)	(2,5,7)	(3,5,7)	(5,6,9)
(0,3,4)	(0,5,8)	(1,3,4)	(1,5,8)	(2,5,8)	(3,5,8)	(5,7,8)
(0,3,5)	(0,5,9)	(1,3,5)	(1,5,9)	(2,5,9)	(3,5,9)	(5,7,9)
(0,3,6)	(0,6,8)	(1,3,6)	(1,6,8)	(2,6,8)	(3,6,8)	
(0,3,7)	(0,6,9)	(1,3,7)	(1,6,9)	(2,6,9)	(3,6,9)	
(0,3,8)	(0,7,8)	(1,3,8)	(1,7,8)	(2,7,8)	(3,7,8)	
(0,3,9)	(0,7,9)	(1,3,9)	(1,7,9)	(2,7,9)	(3,7,9)	

Step 1: Select a starting (t-1)-tuple.

This example selects this (t-1)-tuple uniformly at random (referred to as the "U" method in the text.)

	f_0	f_1	f_2	f_3	f_4
Test 1	0	2	4	6	8
Test 2	1	?	?	6	?

Step 2: Choose the next factor to assign a level value to by computing factor densities. In this case, there is a 3-way tie, so we show the computation for f_1 as an example.

$f_1 = 2$		$f_1 = 3$	
(1,2,4)	½	(1,3,4)	½
(1,2,5)	½	(1,3,5)	½
(1,2,6)	1	(1,3,6)	1
(1,2,8)	½	(1,3,8)	½
(1,2,9)	½	(1,3,9)	½
(2,5,6)	½	(3,4,6)	½
(2,6,9)	½	(3,5,6)	½
Total	4	(3,6,8)	½
		(3,6,9)	½
		Total	5

Density for f_1

$$\frac{(4+5)}{2} = 4.5$$

Step 3: Choose a level for factor f_1 . As shown above, $f_1 = 3$ has a larger density than that of $f_1 = 2$.

	f_0	f_1	f_2	f_3	f_4
Test 1	0	2	4	6	8
Test 2	1	3	?	6	?

Figure 2: Walk-through of constructing a sample test.

values. The example uses 3-way interaction coverage and 1-restricted density. The left side of the figure includes an input with five factors that have two levels each, a sample first test that covers ten 3-tuples, and a table that is shaded to mark these covered 3-tuples. On the right side of the Figure 2, the second test is under construction. Step 1 is to select the first (t-1)-tuple. While this paper includes three different methods to select this starting tuple (described in the next paragraph), this example selects the starting (t-1)-tuple uniformly at random. The (t-1)-tuple of (1,6) is selected at random. Step 2 computes the factor density of all remaining free factors. These factors include: f_1 , f_2 , and f_4 . All three of these factors have the same factor density and the tie is broken at random to select factor f_1 . The figure shows an example that computes density for factor f_1 . The first column lists the 3-tuples that either will, or may potentially, be covered when f_1 is set to 2. The (1,2,6) tuple will be covered when f_1 is set to 2 and contributes 1.0 towards density. The other 3-tuples may potentially be covered but are subject to the yet unknown level that will be assigned to

another free factor associated with the tuple. Therefore, these 3-tuples contribute $\frac{1}{2}$ towards density, where the denominator of 2 is the maximum number of levels associated with any factor for this input. The 3rd and 4th columns of this example repeat these computations, but for f_1 set to level 3. To compute factor density, these level densities are added and divided by 2, to obtain a factor density of 4.5. Again, there is a tie such that f_1 , f_2 , and f_4 have the same factor densities and this example uses f_1 to demonstrate the computations. Next, assign a level value to f_1 . Using the table within the figure that computes the f_1 factor density using levels 2 and 3, it is clear that level 3 has larger density. Factor f_1 is assigned level 3. This process continues until the row is constructed.

The following experiments examine methods that order factors by specifying how the initial factors are chosen, then how the remaining ones are chosen. The *best (t - 1)-tuple method*, denoted by B, fixes the first $t - 1$ factors and their values to those of a $(t - 1)$ -tuple appearing in the most uncovered t -tuples (ties broken uniformly at random). The *uncovered t-tuple method*, denoted by U, selects an uncovered t -tuple uniformly at random and fixes the t factors to the corresponding values. The *density method*, denoted by D, fixes the initial factors by density *with the same restriction as that used for selection of levels*. The latter encounters a problem with restricted density when fewer than $t - \ell$ factors are fixed, since the ℓ -restricted densities are all zero; in this case, the first $t - \ell - 1$ factors and their values are essentially chosen at random. Once initial factors are fixed, the remaining factors are either chosen randomly (denoted R), or using restricted density (denoted by 0, 1, 2, or A for 0-, 1-, 2-, or un-restricted density). In combination, this gives 21 different level selection and factor ordering rules. Results are shown in Table 4.

Table 4 shows some general patterns. As expected, the use of restricted density without specifying an initial set of fixed factors by an alternate rule (B or U) does not fare well. Indeed in these cases it can and does happen that tests are added that fail to cover any new t -tuples at all, and only the randomness in the selection of each test overcomes this eventually. The problem is most severe for 0-restricted density, also as expected. However, even for unrestricted density, employing a separate rule for the initial factors appears to improve the results. Of the two proposed, best $(t - 1)$ -tuple and uncovered t -tuple, the latter is the more frequent winner. Whether to order the remaining factors by restricted density, or randomly, is generally in favour of ordering by density in these examples. Finally, the greater the restriction on the density used in level selection, the poorer the results become for larger numbers of factors – but for fewer factors, the restriction on density does not appear to have the same deleterious effect.

Caution must be exercised in drawing firm conclusions. These results are for few repetitions, and permit each method to examine only few candidates for each test. This is in keeping with ensuring relatively fast execution times, but since the methods all make substantial random selections, there is no reliable method to understand the potential effects of permitting more candidates to be examined, or repetitions to be performed, on the quality of the results. Table 5 examines the effect of permitting more candidates, or more repetitions, or both. Table 6 treats one of these cases in further detail, focussing only on two special cases. The first is unrestricted density for level selection, and uncovered t -tuple followed by

Type	0-restricted						1-restricted						2-restricted						unrestricted					
	B0	BR	DR	U0	UR		B1	BR	DR	U1	UR		B2	BR	DR	U2	UR		BA	BR	DA	DR	UA	UR
2 ⁵	16	16	16	16	16		17	19	19	16	16		17	19	19	16	16		17	19	17	16	16	16
2 ⁶	28	27	31	25	27		26	27	28	27	27		25	26	27	27	27		25	26	27	25	27	27
2 ⁷	29	30	36	28	30		30	30	30	29	31		32	28	31	29	30		31	28	29	29	29	30
2 ⁸	33	33	39	33	32		32	33	29	29	31		24	28	30	33	33		31	32	34	31	33	33
2 ⁹	32	36	42	35	36		37	35	40	35	36		31	36	34	36	35		35	35	35	36	31	33
2 ¹⁰	40	40	49	37	40		34	39	41	39	39		39	39	38	39	39		39	39	39	41	38	40
3 ⁵	101	100	123	100	97		105	100	108	100	97		105	100	103	100	97		105	100	104	99	100	97
3 ⁶	135	138	173	137	137		135	135	144	135	135		137	137	137	135	135		137	137	138	134	135	135
3 ⁷	160	161	205	157	163		159	161	166	159	161		160	160	161	158	160		158	159	161	163	158	160
3 ⁸	180	184	220	180	186		180	178	189	179	182		180	178	182	177	181		176	181	179	179	180	179
3 ⁹	204	208	232	206	211		199	200	213	198	202		195	200	203	195	200		199	199	200	201	196	200
3 ¹⁰	226	231	272	219	230		211	218	229	215	217		213	218	218	215	219		213	217	214	217	213	217
4 ⁵	319	326	517	319	320		329	327	359	319	320		329	327	326	319	320		329	327	327	325	319	320
4 ⁶	425	430	568	425	426		427	425	438	420	422		425	426	425	420	422		425	426	428	425	420	422
4 ⁷	508	517	718	504	507		502	507	524	500	507		504	506	502	497	501		505	505	500	502	497	501
4 ⁸	582	591	904	581	597		570	576	598	566	578		570	574	578	566	579		569	577	575	577	564	575
4 ⁹	645	668	1010	646	666		632	644	666	628	644		633	641	638	625	640		633	640	636	637	623	640
4 ¹⁰	712	739	1011	711	731		688	708	735	681	703		689	698	708	683	699		686	698	690	697	680	699
5 ⁵	790	783	2345	787	784		805	808	816	787	784		805	808	798	787	784		805	808	807	797	787	784
5 ⁶	1017	1035	1809	1014	1023		1024	1035	1107	1007	1017		1033	1023	1030	1007	1017		1033	1023	1030	1024	1007	1017
5 ⁷	1224	1248	2471	1221	1244		1219	1225	1290	1209	1213		1214	1215	1227	1201	1210		1213	1224	1217	1213	1201	1210
5 ⁸	1408	1442	3507	1407	1439		1383	1405	1470	1370	1397		1381	1395	1401	1371	1392		1383	1396	1393	1395	1372	1391
5 ⁹	1565	1621	4116	1563	1623		1535	1567	1636	1522	1560		1533	1557	1560	1519	1555		1534	1555	1542	1559	1523	1552
5 ¹⁰	1724	1800	4342	1724	1799		1675	1710	1780	1664	1705		1675	1706	1715	1658	1703		1671	1698	1689	1707	1654	1696

Table 4: Comparison of level selection and factor ordering rules for $t = 4$ (3 candidates, 3 repetitions).

Type	Can	Rep	0-restricted				1-restricted				2-restricted				unrestricted					
			B0	BR	U0	UR	B1	BR	U1	UR	B2	BR	U2	UR	BA	BR	DA	DR	UA	UR
3^5	1	100	96	93	96	91	97	97	96	91	97	97	96	91	97	97	95	100	96	91
	10	10	96	96	91	93	96	98	91	93	96	98	91	93	96	98	100	100	91	93
	100	1	104	100	96	98	90	107	96	98	90	107	96	98	90	107	100	103	96	98
3^6	1	100	133	135	134	135	134	135	133	134	133	135	133	134	133	135	131	137	133	134
	10	10	133	133	132	129	133	132	132	132	134	133	132	132	134	133	134	133	132	132
	100	1	133	137	132	133	140	136	134	133	136	139	134	133	136	139	136	136	134	133
3^7	1	100	161	166	161	165	158	157	159	160	157	161	158	161	157	161	156	162	158	161
	10	10	154	155	155	155	157	156	156	154	156	153	156	155	156	158	156	156	156	155
	100	1	155	158	157	159	160	158	156	157	155	160	156	159	159	159	159	157	156	159
3^8	1	100	186	195	186	192	179	184	180	184	177	184	180	185	177	183	178	185	180	184
	10	10	175	178	175	176	175	177	173	176	176	175	176	176	178	176	179	175	174	175
	100	1	180	177	172	175	176	179	177	179	176	176	176	172	180	181	181	174	171	177
3^9	1	100	209	218	207	218	198	206	201	206	197	201	199	202	195	203	195	206	199	203
	10	10	196	196	194	198	195	196	193	193	196	196	194	193	195	195	196	195	191	193
	100	1	194	197	190	199	197	192	192	192	197	194	193	193	196	195	201	194	191	194
3^{10}	1	100	227	241	231	241	216	226	216	225	215	223	216	223	213	222	213	224	216	223
	10	10	212	217	212	217	211	211	207	210	211	211	208	209	212	211	215	210	209	212
	100	1	210	210	209	209	215	213	210	209	212	213	207	209	212	210	214	208	209	208

Table 5: Effect of candidates and repetitions on level selection and factor ordering rules for $t = 4$.

unrestricted density for factor ordering. The second is 0-restricted density for level selection, and uncovered t -tuple followed by random for factor ordering. The second is “more greedy” in focussing on t -tuples that are guaranteed to be covered while ignoring those that may be covered later; at the same time, it permits a much wider range of factor orderings that further repetitions and candidates may be able to exploit. The first is essentially a practical implementation of unrestricted density.

In each case, increasing repetitions or the number of candidates improves the result. However, unrestricted density using density to order factors fares much better when the number of repetitions or candidates is severely constrained. None of the results show that increasing repetitions is as effective as permitting more candidates. However, the results are presented only for relatively few candidates and repetitions, in keeping with the stated goal that test suite generation be fast. Perhaps surprisingly, the methods that order factors randomly after the initial uncovered t -tuple is selected exhibit continued improvement as more candidates are examined, while selection via density appears to reach a plateau without further improvement earlier. In essence, it appears that ordering by density considers fewer orderings, but ones which *on average* are better than random orderings. Then when only few candidates are selected, density tends to obtain the better one; but when more are selected, density is constrained to select one that, while perhaps better than average, is not the best.

Consider the execution times in Table 6 (from an implementation in C on a SunBlade

	Reps	1 candidate		10 candidate		100 candidate	
		size	time(sec)	size	time(sec)	size	time(sec)
0-restricted unc+random	1	224	0.6	198	5.3	199	53.0
	10	224	6.3	198	53.9	191	519.5
	100	218	63.1	197	540.6	189	5320.4
1-restricted unc+random	1	209	0.8	198	8.1	192	78.5
	10	209	8.9	193	80.9	189	788.8
	100	206	89.6	193	808.1	189	7908.9
2-restricted unc+random	1	212	1.7	199	16.0	193	155.3
	10	207	17.1	193	158.2	191	1557.1
	100	202	171.7	193	1581.1	189	15579.3
unrestricted unc+density	1	203	2.8	195	26.9	191	263.9
	10	199	28.3	191	268.2	190	2664.5
	100	199	284.5	190	2688.3	190	26583.1

Table 6: Density – Sizes and execution time of strength 4 covering arrays of type 3^9

1000 system). More repetitions and more candidates have a substantial additional cost, which many testers may not find worthwhile for marginal improvements in test suite size. If any realistic time budget is imposed, the data suggest that, to a point, more candidates is a better investment than more repetitions, and that unrestricted density produces smaller test suites when few repetitions and few candidates are permitted.

4 Sizes of Test Suites

Density is easily implemented, and is intended primarily to give an efficient method for the generation of test suites of larger strength while providing some theoretical guarantee that the suite size is at least within a constant factor of the optimal. It is not intended to produce the smallest test suite. Combinatorial Test Services (CTS) [16] combines some easily implemented combinatorial constructions with a heuristic method for generating test suites. Table 7 compares the results from the density approach with those reported by CTS; improvements made by density are shown in bold. It appears that when CTS has a powerful combinatorial construction at its disposal, density is not competitive; however when CTS uses computational search, density obtains smaller test suites.

FireEye implements a different greedy method. In the paper by Lei et al. [24], a comparison with jenny [19], TVG [33], ITCH [18], and TConfig [34] provides convincing evidence that FireEye outperforms each. A comparison with density, limited to the parameters in [24], is given in Table 8. While FireEye often outperforms the CTS published results, the density method appears to produce smaller solutions than that of FireEye in these few examples.

Table 9 instead compares the best result that obtained from any variant of the one-test-at-a-time greedy methods with the best known results from [13, 25], which include new

$v \rightarrow$	2		3		4		5		6		7		8	
$k \downarrow$	CTS	Dens	CTS	Dens	CTS	Dens	CTS	Dens	CTS	Dens	CTS	Dens	CTS	Dens
5	24	16	135	91	256	310	625	769	2036	1604	2401	2968	4096	5052
6	28	24	153	131	564	411	625	984	2300	2017	2401	3692	4096	6222
7	38	24	207	154	688	485	1725	1171	2380	2397	2401	4371	4096	7372
8	42	24	207	172	696	549	1725	1335	2400	2728	2401	5000	4096	8431
9	50	26	285	191	696	611	1725	1483	4062	3039	4095	5569	4096	9399
10	50	26	309	207	696	664	1725	1618	6534	3326	6553	6102	6560	10325

Table 7: Comparison of unrestricted density (10 candidates, 100 repetitions for 2–4 levels and 10 repetitions for 5–8 levels, uncovered 4-tuple followed by unrestricted density) and CTS [16]

$v \rightarrow$	2		3		4		5		6		7		8	
$k \downarrow$	FE	Dens	FE	Dens	FE	Dens	FE	Dens	FE	Dens	FE	Dens	FE	Dens
10	46	26	229	207	649	664	1843	1618	3808	3326	7061	6102	11993	10325

Table 8: Comparison of unrestricted density (10 candidates, 100 repetitions for 2–4 levels and 10 repetitions for 5–8 levels, uncovered 4-tuple followed by unrestricted density) and FireEye [24]

$v \rightarrow$	2		3		4		5		6		7		8	
$k \downarrow$	Pub	Dens	Pub	Dens	Pub	Dens	Pub	Dens	Pub	Dens	Pub	Dens	Pub	Dens
5	16	16	81	91	256	310	625	767	2036	1604	2401	2968	4096	5052
6	21	22	115	129	375	411	625	980	2300	2017	2401	3692	4096	6222
7	24	24	133	153	508	484	1245	1171	2380	2397	2401	4371	4096	7372
8	24	24	153	171	508	547	1245	1335	2400	2728	2401	5000	4096	8431
9	24	24	159	189	508	608	1245	1483	3816	3039	4095	5569	4096	9399
10	24	25	159	207	508	656	1245	1618	3816	3326	4795	6102	6560	10325

Table 9: Comparison of best density result (Dens) and best published result (Pub) [13]

combinatorial constructions and constructions from metaheuristic search. Surprisingly, in the cases shown in bold, density produces the best available results. Metaheuristic search such as simulated annealing could surely improve these results, but the time to invest would be substantial; the density method constructs these arrays in minutes.

Paintball [20] reports occasional improvements on FireEye for strengths four and smaller, but its utility is primarily for strengths five and larger; see, for example, tables of the best known covering arrays [11]. Density compares well for larger strengths against Paintball; for $t = 5$, covering arrays of type 2^k for $k = 20, 30, 40$ have 152, 202, and 230 tests, respectively.

Density improves these to 146, 189, and 226. In the same way, for $t = 6$, covering arrays of type 2^k for $k = 20, 30$ have 346 and 524 tests, respectively. Density improves these to 342 and 475. Paintball does, however, scale to larger problems with lower computational investment than does density; thus it provides a valuable alternative for large problems.

Until this point, cases with strength four have been emphasized. However, density scales to larger strengths as well. Tables 10 and 11 give results for strengths 5 and 6 using 1-restricted density, 1 candidate, and 1 repetition. The 1-restricted density is chosen here because it produces the smallest covering arrays in many of the previous experiments. In addition, only one candidate and one repetition are chosen here since limiting candidates and repetitions requires less execution time. These afford improvements upon previously best known sizes [11].

t	k	v	Size	k	v	Size	k	v	Size
5	18	2	138	18	3	1200	18	4	5136
5	19	2	143	19	3	1313	19	4	5330
5	20	2	148	20	3	1301	20	4	5531
5	21	2	153	21	3	1399	21	4	5724
5	22	2	158	22	3	1389	22	4	5909
5	23	2	162	23	3	1426	23	4	6085
5	24	2	167	24	3	1467	24	4	6249
5	25	2	169	25	3	1500	25	4	6427

Table 10: Density – Sizes of strength 5 covering arrays using 1-restricted density.

t	k	v	Size
6	9	3	1774
6	10	3	2103
6	9	4	10248
6	10	4	12158
6	9	5	40777
6	10	5	48347

Table 11: Density – Sizes of strength 6 covering arrays using 1-restricted density.

Results for cases with fixed numbers of levels have been emphasized, because at this time there are few results for mixed levels with which to compare. However, in order to demonstrate feasibility for mixed levels, two results for mixed-level covering arrays of strength 4 have been calculated using 1-restricted density, 1 candidate, and 1 repetition: $4^{11}5^13^82^2$ of size 377 and $4^13^{39}2^{35}$ of size 181.

Two limitations are imposed by the one-test-at-a-time greedy methods, whether based on density or not. The first is that each test is chosen with the objective of maximizing

newly covered t -tuples. The second is that a level for each factor is chosen in turn and not subsequently replaced. In Table 9, one sees that often density-based methods are far off the mark. In the absence of further information, one could attribute this to a fundamental limit of any one-test-at-a-time method, or to the specific rules used to select the levels in the test. In order to differentiate, for very small values of t , k , and v , it is possible to enumerate all of the v^k tests and select one that truly maximizes the number of newly covered t -tuples. Call this the *best test* method. Although this is exponential in k , it is feasible for small values of k . There may be many such choices, so a list is kept to maintain all tests that realize the maximum and so that one of these tests can be selected at random as the next test. This produces a range of test suite sizes that any one-test-at-a-time greedy method selecting tests based on uncovered t -tuples could produce.

In 100 trials for 3^6 with strength four, test suite sizes range from 130 to 140. Thus the best density solution at 129 tests, obtained by an efficient algorithm, appears quite competitive. However contrasting 129 tests with the best known solution at 115 tests suggests that one-test-at-a-time methods fail in this case to produce test suites close to the smallest. The test suite with 115 tests was produced by a simulated annealing algorithm [9]. This underscores the earlier statements that such one-test-at-a-time methods are intended primarily for efficiency, and for the production of test suites covering many t -tuples early.

The limitation of one-test-at-a-time methods to produce the smallest size solutions is not restricted to this example. For 3^7 , the best test gives from 153 to 160 tests; density at 153 is good, but the best available result of 133 from simulated annealing is substantially better. For 3^8 , the best test gives from 173 to 181 tests; density is at 171, but the best available result of 153 is from simulated annealing. For 4^6 , the best test gives from 407 to 425 tests; density at 420 is within this range, but the best available result of 375 again arises from simulated annealing. For 4^7 , the best test gives from 482 to 493 tests; density at 484 is within range, but the best available previous result of 508 appears to be weak [30]. A full explanation of these results seems to be beyond reach at the moment. However the results speculate that one-test-at-a-time methods based on uncovered t -tuples, while able to provide a guarantee on test suite size, do not produce the smallest test suites. In addition, while density affords an efficient and useful heuristic for selecting the next test covering the most t -tuples, its myopia in fixing one factor at a time cannot ensure that the best next test is selected, but rather one that is better than average.

The density method can be used in conjunction with other techniques as well. To demonstrate this, form an optimal test suite for eight factors, having seven levels each, in $2401 = 7^4$ tests; this is an orthogonal array. Then select one level to remove for each factor; whenever that (factor,level) combination appears in a test, mark the corresponding entry as unspecified, and all other entries are marked as specified. Adjoin additional factors and mark their levels in each test as unspecified. Then density can first be used to select levels for all unspecified entries in each test of the seed array (for each test, factors with specified levels are fixed, those with unspecified levels are free, and free factors are ordered by unrestricted density); thereafter additional tests are added to complete the array, each selected by ordering factors by uncovered 4-tuple followed by unrestricted density and selecting levels by unrestricted

density. For 6^9 , using ten candidates and three repetitions, a covering array with 2906 tests is produced. This is both smaller (compare with 3039), and faster to produce, than applying density with no seed. For 6^{10} , 3287 tests are obtained with this seed, also comparing favourably to the 3326 tests from density with no seed. Such hybrid combinations of the density methods with combinatorial and other approaches may generally result in smaller test suites that are produced more quickly; this deserves further study.

5 Conclusions

The density method developed generates covering arrays (test suites) of arbitrary strength in time that is polynomial in k for fixed t and v . When unrestricted density is used, the method ensures that the size of the test suite is within a constant factor of the logarithmic lower bound, the constant depending only on t and v . Despite the resulting theoretical attractiveness of the method, the need to calculate all of the required density measures limits the method to “small” values of t , v , and k . However, substantial experimental results indicate that restricting density to focus primarily on fixed factors not only accelerates the computation, but also can yield smaller test suites. Results suggest that limited lookahead to factors not yet fixed is better than no lookahead, unless many candidates are examined.

References

- [1] T. Berling and P. Runeson. Efficient evaluation of multifactor dependent system performance using fractional factorial design. *IEEE Transactions on Software Engineering*, 29(9):769–781, 2003.
- [2] R. C. Bryce and C. J. Colbourn. Prioritized interaction testing for pairwise coverage with seeding and avoids. *Information and Software Technology Journal*, 48(10):960–970, 2006.
- [3] R. C. Bryce and C. J. Colbourn. The density algorithm for pairwise interaction testing. *Software Testing, Verification, and Reliability*, 17(3):159–182, 2007.
- [4] R. C. Bryce, C. J. Colbourn, and M. B. Cohen. A framework of greedy methods for constructing interaction tests. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 146–155. IEEE, Los Alamitos, CA, 2005.
- [5] K. Burr and W. Young. Combinatorial test techniques: Table-based automation, test generation, and code coverage. In *Proceedings of the International Conference on Software Testing Analysis and Review*, pages 503–513. ACM, New York, 1998.
- [6] M. Chateauneuf and D. L. Kreher. On the state of strength-three covering arrays. *J. Combin. Des.*, 10(4):217–238, 2002.

- [7] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–44, 1997.
- [8] G. D. Cohen, S. Litsyn, and G. Zémor. On greedy algorithms in coding theory. *IEEE Trans. Inform. Theory*, 42:2053–2057, 1996.
- [9] M. B. Cohen. *Designing test suites for software interaction testing*. PhD thesis, The University of Auckland, Department of Computer Science, 2004.
- [10] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling. Constructing strength three covering arrays with augmented annealing. *Discrete Mathematics*, to appear.
- [11] C. J. Colbourn. Covering array tables. <http://www.public.asu.edu/~ccolbou/src/tabby>, 10 July 2007.
- [12] C. J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche (Catania)*, 58:121–167, 2004.
- [13] C. J. Colbourn, S. S. Martirosyan, T. v. Trung, and R. A. Walker II. Roux-type constructions for covering arrays of strengths three and four. *Designs, Codes and Cryptography*, 41:33–57, 2006.
- [14] S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino. Applying design of experiments to software testing. In *Proc. Intl. Conf. on Software Engineering (ICSE '97)*, pages 205–215. IEEE, Los Alamitos, CA, 1997.
- [15] M. Grindal, J. Offutt, and S. F. Andler. Combination testing strategies – a survey. *Software Testing, Verification, and Reliability*, 5(3):167–199, 2005.
- [16] A. Hartman. Software and hardware testing using combinatorial covering suites. In M. C. Golumbic and I. B.-A. Hartman, editors, *Interdisciplinary Applications of Graph Theory, Combinatorics, and Algorithms*, pages 237–266. Springer, Norwell, MA, 2005.
- [17] B. Hnich, S. Prestwich, and E. Selensky. Constraint-based approaches to the covering test problem. *Lecture Notes in Computer Science*, 3419:172–186, 2005.
- [18] IBM. ITCH. <http://www.alphaworks.ibm.com/tech/whitch>.
- [19] R. Jenkins. jenny: A pairwise testing tool. <http://www.burtleburtle.net/bob/index.html>.
- [20] D. R. Kuhn, Y. Lei, R. Kacker, V. Okun, and J. Lawrence. Paintball: A fast algorithm for covering arrays of high strength. *Internal Tech. Report, NISTIR 7308*, 2007.
- [21] D. R. Kuhn and M. Reilly. An investigation of the applicability of design of experiments to software testing. In *Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, pages 91–95. IEEE, Los Alamitos, CA, 2002.

- [22] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Trans. Software Engineering*, 30(6):418–421, 2004.
- [23] L. J. Lazić and D. Velašević. Applying simulation and design of experiments to the embedded software testing process. *Software Testing, Verification, and Reliability*, 14(4):257–282, 2004.
- [24] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. IPOG: A general strategy for t-way software testing. In *Fourteenth Int. Conf. Engineering Computer-Based Systems*, pages 549–556, 2007.
- [25] S. S. Martirosyan and C. J. Colbourn. Recursive constructions of covering arrays. *Bayreuth. Math. Schr.*, 74:266–275, 2005.
- [26] S. S. Martirosyan and T. v. Trung. On t -covering arrays. *Des. Codes Cryptogr.*, 32(1-3):323–339, 2004.
- [27] K. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Applied Math.*, 138(9):143–152, 2004.
- [28] S. Poljak, A. Pultr, and V. Rödl. On qualitatively independent partitions and related problems. *Discrete Applied Math.*, 6:193–205, 1983.
- [29] S. Poljak and Z. Tuza. On the maximum number of qualitatively independent partitions. *Journal of Combinatorial Theory (A)*, 51:111–116, 1989.
- [30] G. B. Sherwood, S. S. Martirosyan, and C. J. Colbourn. Covering arrays of higher strength from permutation vectors. *J. Combin. Des.*, 14(3):202–213, 2006.
- [31] K. C. Tai and L. Yu. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, 2002.
- [32] Y. W. Tung and W. S. Aldiwan. Automating test case generation for the new generation mission software system. In *Proc. 30th IEEE Aerospace Conference*, pages 431–437. IEEE, Los Alamitos, CA, 2000.
- [33] TVG. TVG testing tool. <http://sourceforge.net/projects/tvg/>.
- [34] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Testing of Communicating Systems: Tools and Techniques*, pages 59–74. Kluwer, Boston Dordrecht London, 2000.
- [35] C. Yilmaz, M. B. Cohen, and A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 31:20–34, 2006.