Efficient Algorithms for Generation of Combinatorial Covering Suites

Adrian Dumitrescu

Department of Computer Science,
University of Wisconsin–Milwaukee,
ad@cs.uwm.edu

Abstract. In this note we describe efficient algorithms for generating tests that cover a prescribed set of combinations of a software system's input parameters. Our methods for obtaining uniform t-wise coverage are based on repeatedly coloring the vertices of a graph such that the vertices in each t-subset have different colors in at least one of the colorings. The resulting algorithm is compared to other known algorithms for uniform coverage, a greedy algorithm and a randomized algorithm, in particular. The size of its output test suite is then related to a new lower bound that we obtain on the minimal size of a test suite.

Keywords: Combinatorial covering suite, Software testing, Automatic test generation.

1 Introduction

Development and production of high-quality software at a reasonable price is a critical issue for today's society, when the safety and quality of life is increasingly dependent on software systems. Software testing is an important but expensive part of the software development process, as it often consumes between 1/3 and 1/2 of this cost [7].

In this note, we address black-box testing, which ensures that a program meets its specification from a functional perspective. The number of possible black-box test cases is equal to the number of all possible valid settings of the input parameters of the system and thus in general is extremely large. Consequently testers have resorted to restricting the number of input combinations fed into the system under test, by only requiring that all pairwise (and more generally t-wise) combinations of input parameters are covered. For example, covering all pairwise combinations means that for any two parameters P_1 and P_2, and any valid values v_1 for P_1 and v_2 for P_2, there is a test in which P_1 is set to v_1 and P_2 is set to v_2.

The efficient construction of combinatorial covering suites has important applications in the process of testing software and hardware systems. A short example (cf. [17]) is as follows. Before shipping new machines to customers, running some final tests is desired. There are 16 switches on the back of each machine, that have to be set, each with two positions. Testing all possible $2^{16} = 65,536$

T. Ibaraki, N. Katoh, and H. Ono (Eds.): ISAAC 2003, LNCS 2906, pp. 300–308, 2003.

possible combinations is infeasible, but testing all 2^3 combinations for each subset of three switches is possible. Not more than 17 test vectors are necessary: 0000000000000000 and all the cyclic shifts of 0000101101110111 are sufficient for this task [17]. The same principle (and test sequence) can be applied for testing a software system with 16 inputs if every output of the software system depends on at most three input parameters (or one wants to test the interactions of all triples of parameters). A higher degree of testing of interactions between input parameters can be achieved using t-wise coverage (for some $t \geq 2$): the test sequence will discover all errors if each output of the system depends on at most t input parameters.

A technical challenge that remains in applying this technique in software testing is the issue of efficiently constructing the covering test suite. This issue is important, because the number of test cases, as well as the time necessary to obtain them directly influences the amount of resources needed to test a software system.

1.1 Covering Suites

Throughout this note, let $[m]$ denote the set $\{1, 2, \ldots, m\}$, and $\log x$ stand for $\log_2 x$. We follow some of the terminology in [10]. Consider a software system with k parameters whose domains are the finite sets $D_1, \ldots, D_k$. We write $l_i = |D_i|,\ i = 1, \ldots, k$. The actual domains are in fact not important, but only their sizes; we will therefore assume that $D_j = \{0, 1, \ldots, l_j - 1\}$. A test suite with N test vectors is a matrix $A = (a_{ij} : 1 \leq i \leq N, 1 \leq j \leq k)$, where $a_{ij} \in D_j$ for all i and j. The rows of the matrix are called test vectors, or simply tests. A test suite A can be also viewed as a sequence $\mathcal{T}$ of N test vectors.

We say that a test suite A is a *t-wise covering suite* with parameters $l_1, \ldots, l_k$, $(t \leq k)$, if for any subset $\{j_1, \ldots, j_t\}$ of t columns of A, and for any t-tuple of values $T \in D_{j_1} \times \ldots \times D_{j_t}$, there exists at least one row r in A, such that $(a_{r,j_1}, \ldots, a_{r,j_t}) = T$. We speak in this case of *uniform coverage*. The *covering suite number* $h_t(l_1, \ldots, l_k)$ is then defined as the minimum integer N such that there exists a t-wise covering suite with N tests for k domains of sizes $l_1, \ldots, l_k$. Other names used in the literature for t-wise covering suites are covering arrays, (k, t)-universal sets, and t-surjective arrays. If all k domains are of the same size l — which we refer to as the *uniform range* case — we write $h_t(l^k)$ instead of $h(l, \ldots, l)$. Similarly, we write $h_t(l^2, m^3)$ instead of $h(l, l, m, m, m)$ for example. We call the matrix of a t-wise covering suite with k domains of size l a (k, t, l)-*universal test matrix*. If the domains are ordered by their size, say $l_1 \geq \ldots \geq l_k$, then clearly $h_t(l_1, \ldots, l_k) \geq l_1 \ldots l_t$ [10].

The uniform case $l = 2$ (all the domains are binary) and $t = 2$ (pairwise coverage), was solved by Rényi [15], Katona [11], Kleitman and Spencer [12]: for all $k \geq 2$, $h_2(2^k) = N$, where N is the smallest integer such that

$$\binom{N-1}{\lceil N/2 \rceil} \geq k.$$

A test suite of this size may be constructed as follows: the first test is the 0 vector in all coordinates. The columns of the remaining $N-1$ rows each contain exactly $\lceil N/2 \rceil$ ones, and each column is constructed by choosing a different $\lceil N/2 \rceil$-subset of the rows [10]. Asymptotically $h_2(2^k)$ satisfies

$$h_2(2^k) = \log k + \frac{1}{2} \log \log k + O(1). \tag{1}$$

For the uniform case $l = 2$ and $t = 3$ (3-wise coverage), the best known bounds are

$$3.21256 \ldots \log k(1 + o(1)) \leq h_3(2^k) \leq 7.56444 \ldots \log k(1 + o(1)),$$

where the lower bound is due to Kleitman and Spencer [12], while the upper bound is due to Sloane [17].

An extensive literature is devoted to constructing covering suites using orthogonal arrays, intersecting codes, or algebraic techniques — see [10,17] and the references therein. However these methods are complicated and in many cases do not work for an arbitrary given number of parameters, or of domain-size.

Seroussi and Bshouty [16] have shown that a generalized version of the problem of finding a minimal t-covering suite — where only a prescribed collection of t-sets of parameters needs to be covered — is NP-complete, using a reduction from graph 3-coloring. We refer to this case as non-uniform coverage. Thus finding polynomial time algorithms for generating optimal covering suites in the general case remains unlikely. However, to efficiently compute covering suites whose sizes are close to optimal remains a topic of continued interest.

In our note we present algorithms for efficient generation of combinatorial covering suites, and whose modular design allows for the resulting covering suites to be stored in a compact form. In Section 2 we address pairwise coverage, we continue with t-wise coverage in Section 3, and we conclude with a short discussion of non-uniform ranges and post-processing in Section 4.

The non-uniform coverage demand comes also from practical considerations, since outputs of the software system seldom depend on the same number of input parameters. A technique which in many cases reduces a given testing problem to one with a smaller number of parameters (and thus easier to solve) appears in [2].

2 Pairwise Coverage

The reported effectiveness of test sets with a low degree of coverage (such as pairwise or triple) is a major motivation for the combinatorial design approach [5]. In this section we outline a simple algorithm for generating covering suites in the uniform case (with k domains of size l) and $t = 2$ (pairwise coverage), and then implicitly get an upper bound on $h_2(l^k)$. We however note that better constructions are known for this case and that we feature our algorithm only for its simplicity and to illustrate the simplest case (for $t = 2$ colors) of our

coloring approach detailed in Section 3. For example, Kobayashi et. al. proposed an algebraic method for pairwise coverage which yields a good upper bound on $h_2(l^k)$ and which works well even in the non-uniform range case.

We next outline the algorithm. The following fact is probably well known.

Lemma 1. *The edge set of the complete graph on the k vertices $\{0, \ldots, k-1\}$ can be expressed as a union of the sets of edges of $m = \lceil \log k \rceil$ complete bipartite graphs $(A_i, B_i),\ i = 1, \ldots, m$ on the same set of vertices. More precisely for each edge $(j_1, j_2),\ 0 \le j_1 < j_2 \le k-1$, there exists i, such that $j_1 \in A_i$, $j_2 \in B_i$ and $(j_1, j_2) \in E(A_i, B_i)$. Moreover, each such bipartition can be generated in linear time.*

Proof. Put $m = \lceil \log k \rceil$; m represents the number of bits necessary to represent in binary all integers in the range $\{0, \ldots, k-1\}$. For any such integer j, let j_i be the i-th bit in the binary representation of j. We assume that the vertex set of the complete graph is $\{0, \ldots, k-1\}$. For $i = 1, \ldots, m$, let $A_i = \{j \in \{0, \ldots, k-1\} \mid j_i = 0\}$ and $B_i = \{j \in \{0, \ldots, k-1\} \mid j_i = 1\}$ specify the bipartitions. It is easy to see that each edge $(j_1, j_2),\ 0 \le j_1 < j_2 \le k-1$ is covered as required. All edges of these bipartite graphs are also present in the complete graph, which concludes the proof. □

Set $m = \lceil \log k \rceil$. The algorithm — which we refer to as $\mathcal{A}_1$ — generates the following tests. For each pair of parameters values $(u, v),\ u, v \in [l],\ u \ne v$, output m vectors, each corresponding to a bipartition of the vertex set (of k parameters) in Lemma 1: if (A_i, B_i) is the i-th bipartition, the j-th component $(1 \le j \le k)$ of the i-th output vector w is set as follows. If $j \in A_i$, set $w[j] \leftarrow u$, and if $j \in B_i$, set $w[j] \leftarrow v$. For each pair of parameters values $(u, u),\ u \in [l]$, output a vector w all of whose components are equal to u.

The resulting sequence of vectors is a pairwise covering suite, as we show next. Take any pair of parameters $(j_1, j_2),\ j_1 < j_2$, and any pair of values $(u, v) \in [l]^2$. The case $u = v$ is taken care in the second part of the algorithm. In the case $u \ne v$, there exists a bipartition (A_i, B_i), such that $j_1 \in A_i$ and $j_2 \in B_i$, thus the parameter j_1 is set to u and the parameter j_2 is set to v in the vector corresponding to this bipartition, which is generated when the value pair (u, v) is considered in the first part of the algorithm.

For instance, the covering suite generated for $l = 3,\ k = 7$ is shown in Table 1, and corresponds to the bipartite graphs $A_1 = \{0, 1, 2, 3\},\ B_1 = \{4, 5, 6\}$, $A_2 = \{0, 1, 4, 5\},\ B_2 = \{2, 3, 6\}$, and $A_3 = \{0, 2, 4, 6\},\ B_3 = \{1, 3, 5\}$.

The size of the output covering suite is

$$N_1 = l(l-1)\lceil \log k \rceil + l,$$

which also provides an upper bound on $h_2(l^k)$. Incidentally, the upper bound N_g (see below) on the number of tests derived in [3] using the greedy algorithm is at least twice ours, although their algorithm may output shorter covering suites.

$$N_g = l^2 \times \lceil \log \binom{k}{2} + \log l^2 \rceil.$$

Table 1. Pairwise coverage for seven parameters with three values each

P_0	P_1	P_2	P_3	P_4	P_5	P_6
0	0	0	0	1	1	1
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	0	0	0	2	2	2
0	0	2	2	0	0	2
0	2	0	2	0	2	0
1	1	1	1	0	0	0
1	1	0	0	1	1	0
1	0	1	0	1	0	1
1	1	1	1	2	2	2
1	1	2	2	1	1	2
1	2	1	2	1	2	1
2	2	2	2	0	0	0
2	2	0	0	2	2	0
2	0	2	0	2	0	2
2	2	2	2	1	1	1
2	2	1	1	2	2	1
2	1	2	1	2	1	2
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2

We note that for $l = 2$, the size of the covering suite is roughly twice the size of the optimal in (1). We also note that Gargano, Körner and Vaccaro have shown that for very large values of k, the minimal number of test cases is $\sim \frac{l}{2} \log k$ [8], although their methods are non-constructive and the linear dependence on l for moderate k is unlikely [3,6].

3 *t*-Wise Coverage

In this section we show how the approach in Section 2 can be extended to obtain t-wise coverage in the uniform case (with k domains of size l). This approach can be used for any $t \geq 2$, so we can obtain an alternative method for pairwise coverage ($t = 2$).

The algorithm — which we refer to as $\mathcal{A}_2$ — has two phases. Fix a palette of t colors, say $[t] = \{1, \ldots, t\}$. In the first phase, repeatedly color uniformly and randomly the vertices of the complete graph on k vertices, until each unordered t-tuple is *multicolored* in at least one such coloring, i.e, each element of the t-tuple is colored by a different color in that coloring. Say n colorings have been used to achieve this property. In the second phase, for each ordered t-tuple of parameter values $(v_1, \ldots, v_t) \in [l]^t$, output n vectors, each corresponding to one of the above colorings. For a given coloring c, the j-th component ($1 \leq j \leq k$)

of the output vector w is set according to its color: if $c(j) = i$, set $w[j] \leftarrow v_i$, $i = 1, \ldots, t$.

The next lemma shows that the resulting sequence of vectors is a t-wise covering suite.

Lemma 2. *For each t-tuple of values $(v_1, \ldots, v_t) \in [l]^t$, and each t-tuple of parameters $i_1 < \ldots < i_t$, there exists a test vector w such that $w[i_j] = v_j$, $j = 1, \ldots, t$.*

Proof. Take a coloring c (which gives a test in the output suite) for which all parameters $i_1, \ldots, i_t$ have different colors, i.e., $\{c(i_1), \ldots, c(i_t)\} = [t]$. Let $\tau = (\tau_1, \ldots, \tau_t)$ be a permutation of $[t]$, such that $c(i_j) = \tau_j$, $j = 1, \ldots, t$. Let $\sigma = (\sigma_1, \ldots, \sigma_t) = \tau^{-1}$ be the inverse of τ. Consider the t-tuple of values $(v_{\sigma_1}, \ldots, v_{\sigma_t})$. In the output vector w corresponding to coloring c and the above t-tuple of values,

$$w[i_j] = v_{\sigma_{\tau_j}} = v_j,$$

since by definition of σ, $\sigma_{\tau_j} = j$. □

Next we estimate n so that the resulting sequence is a t-wise covering suite. Fix a t-tuple U of parameters. The probability that U is not multicolored is

$$P_t = Prob[U \text{ is not multicolored}] = \frac{t^t - t!}{t^t}.$$

The probability that U is not multicolored in any of the n colorings is P_t^n, so the expected number of such t-tuples is $\binom{k}{t} P_t^n$. Requiring this to be < 1, and since $\binom{k}{t} < k^t$, for $t \geq 2$, it is enough to take

$$n = \left\lceil \frac{t \log k}{\log \frac{1}{P_t}} \right\rceil.$$

The size of the test sequence output by our algorithm is not more than $N_2 = \left\lceil \frac{t \log k}{\log \frac{1}{P_t}} \right\rceil l^t$. Thus

$$h_t(l^k) \leq \left\lceil \frac{t \log k}{\log \frac{1}{P_t}} \right\rceil l^t.$$

For fixed t (which holds for most practical applications), this becomes

$$N_2 = O((\log k) \cdot l^t). \tag{2}$$

A different randomized algorithm for t-wise coverage — which we refer to as $\mathcal{A}'$ — is presented in [14,9]. That algorithm selects instead a collection of N' random vectors $\in [l]^k$: for each vector, each component is selected uniformly and randomly from $[l]$. Via the probability 'union bound', if

$$N' = \left\lceil \frac{t \log k + t \log l}{\log \frac{l^t}{l^t - 1}} \right\rceil,$$

t-wise coverage is obtained [14]. For fixed t, this becomes $N' = O((\log k + \log l) \cdot l^t)$. The size of the test sequence given by our algorithm $\mathcal{A}_2$ is (under the same assumption) a slight improvement for large l over the above bound, since the logarithmic dependence on l is removed. Besides that, the space and time requirements of $\mathcal{A}_2$ are much smaller that those of $\mathcal{A}'$: the former only needs to verify that all $\binom{k}{t}$ t-tuples of parameters have been covered (i.e., multicolored), while the latter needs to check that all $\binom{k}{t}l^t$ value t-tuples have been covered. Notice that $\binom{k}{t}l^t \gg \binom{k}{t}$!

In addition, if one wants to store enough information for obtaining covering suites for a large family of (k, t, l) instances, this becomes very easy to do using $\mathcal{A}_2$, and it uses very small space. Indeed, once a set of colorings is obtained for a given pair (k, t), it can be used as a seed to obtain covering suites for all (k, t, l) instances (i.e., for any value of l). Moreover, the space needed to store a set of colorings for a given (k, t) instance is much smaller than that needed for storing a complete covering suite for a (k, t, l) instance for a single l: the number of colorings is much smaller than the number of test vectors in a covering suite, and the number of bits per entry is usually smaller ($\lceil \log t \rceil$ versus $\lceil \log l \rceil$).

Next, we give a (straightforward) extension of the lower bound in [16] on the size of minimal t-wise covering suites for the binary case $l = 2$ to arbitrary l. We note that for fixed t and l and large k, the covering suite output by algorithm $\mathcal{A}_2$ is within a multiplicative constant factor of the optimal (cf. (2) above).

Theorem 1. *For all $k \geq t \geq 2$ and $l \geq 2$,*

$$h_t(l^k) \geq \max(l^t, l^{t-2} \cdot \lceil \log (k - t + 2) \rceil).$$

Proof. The first term in the maximum is justified by the trivial lower bound $h_t(l^k) \geq l^t$. To justify the second, let M be a (k, t, l)-universal test matrix of dimensions $m \times k$. For any $x \in [l]^{t-2}$, denote by M_x the matrix of rows of M whose last $t - 2$ coordinates agree with x. If m_x is the number of rows of M_x, then

$$m = \sum_{x \in [l]^{t-2}} m_x.$$

The minimum m_x, corresponding to some $\xi = \xi_1 \xi_2 \dots \xi_{t-2} \in [l]^{t-2}$, satisfies

$$m_\xi \leq \frac{m}{l^{t-2}}. \tag{3}$$

Let $k' = k - t + 2$, and let M' be the $m_\xi \times k'$ matrix consisting of the first k' columns of M_ξ. We claim that M' is $(k', 2, l)$-universal. Indeed, assuming that for two column indices $j_1, j_2 \leq k'$, a value pair (v_1, v_2) is missing, then the t-tuple $v_1 v_2 \xi_1 \xi_2 \dots \xi_{t-2}$ is missing from M at indices $j_1, j_2, k' + 1, \dots, k$, contradicting the assumption that M is (k, t, l)-universal. Hence M' is $(k', 2, l)$-universal, thus also $(k', 2, 2)$-universal. By [1] (and also implied by [12], see also (1)), $m_\xi \geq \lceil \log k' \rceil = \lceil \log (k - t + 2) \rceil$. Combining with (3), we obtain

$$m \geq l^{t-2} m_\xi \geq l^{t-2} \cdot \lceil \log (k - t + 2) \rceil,$$

which completes the proof. □

Corollary 1. *If* $k - t = k^{\Omega(1)}$, *then*

$$h_t(l^k) = \Omega(l^{t-2} \log k).$$

4 Other Aspects and Concluding Remarks

4.1 Non-uniform Ranges

Applications having large variations in the domain-size of the input parameters are a rule rather than an exception in software [4]. Two algorithms $\mathcal{A}_3$ and $\mathcal{A}_4$ are obtained by adapting algorithms $\mathcal{A}_2$ and $\mathcal{A}'$ respectively.

Algorithm $\mathcal{A}_3$ reduces the non-uniform case to the uniform one. It first sets $l = \max(l_1, \ldots, l_k)$, and then uses algorithm $\mathcal{A}_2$ (or $\mathcal{A}'$) to generate a t-wise covering suite for k domains of size l. Then for each test vector in this suite, and for each $i = 1, \ldots, k$, arbitrarily assigns its i-th entry to a value in its valid range if it lies outside its range. Clearly the result is a t-wise covering suite of the same size.

An algorithm similar to $\mathcal{A}'$ is algorithm $\mathcal{A}_4$: random vectors $\in D_1 \times \ldots \times D_k$ are chosen until coverage is obtained: i.e., for each vector, its i-th component is selected uniformly and randomly from D_i.

4.2 Post-processing: The Greedy Algorithm

Cohen et. al. have presented a greedy algorithm for test generation which for a fixed t, shows that the size of the test suite which provides t-wise coverage grows logarithmically in the number of parameters k [4,3]. As remarked in [3], the proof of the logarithmic growth for the greedy algorithm assumes that at each step, it is possible to find a test vector that covers the maximum number of uncovered t-tuples. Since there are an exponential number of test vectors (l^k), this may be computationally unfeasible. Therefore they outlined a heuristic random greedy variant of the algorithm to get around this problem, by using a random order in setting parameter values in their algorithm.

A different approach is to make use of the greedy algorithm in a post-processing optimization step. Once a covering suite $\mathcal{T}$ is obtained, it is fed as input to the greedy algorithm. Since the size of $\mathcal{T}$ is "small", the greedy algorithm can efficiently find a test vector that covers the maximum number of uncovered t-tuples, and thus in the end, it may produce as output only a subset $\mathcal{T}'$ of the covering sequence $\mathcal{T}$. This optimization step can be applied to any of the suites output by our algorithms $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{A}_4$ presented earlier, or obtained by any other method.

4.3 Concluding Remarks

In this note we have outlined alternative algorithms for generating test suites which achieve uniform coverage of the software's input parameters. Our algorithms are efficient and extremely simple to implement. In addition, their design

does not use sophisticated mathematical techniques, and the number of tests generated favorably compares with a general lower bound we have given. Moreover, their modular design allows for the resulting covering suites to be stored in a compact form. In combination with the use of the greedy algorithm as a post-processing optimization step, they provide a practical tool in the software testing process.

References

1. A. K. Chandra, L. T. Kou, G. Markowsky and S. Zaks, On sets of boolean n-vectors with all k-projections surjective, *Acta Informatica*, **20** (1983), 103–111.
2. C. Cheng, A. Dumitrescu and P. Schroeder, Generating small test suites for non-uniform instances, *Third International Conference on Quality Software* (QSIC 2003), Dallas, November 2003, accepted.
3. 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) (1997), 437–444.
4. D. M. Cohen, S. R. Dalal, A. Kajla and G. C. Patton, The automatic efficient test generator (AETG) system, *Proceedings of the 5-th International Symposium on Software Reliability Engineering*, IEEE, 1994, 303–309.
5. D. M. Cohen, S. R. Dalal, J. Parelius and G. C. Patton, The combinatorial design approach to automatic test generation, *IEEE Software*, **13** (1996), 83–89.
6. D. M. Cohen and M. L. Fredman, New techniques for designing qualitatively independent systems, *Journal of Combinatorial Designs*, **6**(6) (1998), 411–416.
7. S. R. Dalal and C. M. Mallows, Factor-covering designs for testing software, *Technometrics*, **40**(3) (1998), 234–243.
8. L. Gargano, J. Körner and U. Vaccaro, Sperner capacities, *Graphs and Combinatorics*, **9** (1993), 31–46.
9. A. P. Godbole, D. E. Skipper and R. A. Sunley, t-Covering arrays: upper bounds and Poisson approximations, *Combinatorics, Probability and Computing*, **5** (1996), 105–117.
10. A. Hartman, Software and hardware testing using combinatorial covering suites, to appear in *Interdisciplinary Applications of Graph Theory, Combinatorics and Algorithms* (ed. M. Golumbic), manuscript, July 2002.
11. G. O. H. Katona, Two applications (for search theory and truth functions) of Sperner type theorems, *Periodica Mathematica Hungarica*, **3** (1973), 19–26.
12. D. J. Kleitman and J. Spencer, Families of k-independent sets, *Discrete Mathematics*, **6** (1973), 255–262.
13. N. Kobayashi, T. Tsuchiya and T. Kikuno, A new method for constructing pairwise covering designs for software testing, *Information Processing Letters* **81**(2) (2002), 85–91.
14. M. Naor, L. J. Schulman and A. Srinivasan, Splitters and near-optimal derandomization, *Proceedings of the 36-th Annual Symposium on Foundations of Computer Science* (FOCS), 1995, 182–191.
15. A. Rényi, *Foundations of Probability*, Wiley, New york, 1971.
16. G. Seroussi and N. H. Bshouty, Vector sets for exhaustive testing of digital circuits, *IEEE Transactions on Information Theory*, **34**(3) (1988), 513–522.
17. N. J. A. Sloane, Covering arrays and intersecting codes, *Journal of Combinatorial Designs*, **1** (1993), 51–63.