REVISED FOR SUBMISSION 19/12/03

Problems and Algorithms for Covering Arrays

Alan Hartman and Leonid Raskin
IBM Haifa Research Laboratory

Dedicated to Curt Lindner on his 65th birthday.

Abstract: Covering arrays are combinatorial structures which extend the notion of orthogonal arrays and have applications in the realm of software testing. In this paper we raise several new problems motivated by these applications and discuss algorithms for their solution.

1. INTRODUCTION

A covering array $CA(t,k,g)$ of size b and strength t, is a $k \times b$ array $A = (a_{i,j})$ over $Z_g = \{0,1,2,...,g-1\}$ with the property that for any t distinct rows $1 \le r_1 < r_2 < ... < r_t \le k$, and any member $(x_1, x_2, ..., x_t)$ of Z_g^t there is at least one column c such that $x_i = a_{r_i,c}$ for all $1 \le i \le t$. (This is what Curt Lindner would call the t-finger rule: "Just run t fingers along any t rows and you will eventually meet all ordered t-tuples over the alphabet Z_g.") The covering array number $CAN(t,k,g)$ is the smallest b for which a $CA(t,k,g)$ of size b exists.

There is a vast array of literature on covering arrays, and the problem of determining the minimum size of covering arrays has been studied under many guises over the past 30-40 years. A recent survey [8] with a comprehensive bibliography is available on the internet. For undefined terms we refer the reader to Colbourn and Dinitz [7].

In this paper we discuss some generalizations of the problem of creating small covering arrays. Our motivation is from the area of software testing. When testing a software component with k parameters, each of which must be tested with g values, the total number of possible test vectors is g^k. If it suffices to test the interactions of any subset of t parameters, then the number of test vectors may be as small as g^t. The test vectors are the columns of a covering array $CA(t,k,g)$.

In this paper we consider five natural generalizations of this problem:

- Covering arrays over heterogeneous alphabets

- Covering arrays with a maximum number of columns
- Covering array extension
- Covering array reduction
- Covering arrays with forbidden configurations

We also describe the software used to solve these problems and discuss the algorithms implemented.

2. THE PROBLEMS

2.1 Heterogeneous alphabets

In practice the number of values to be tested varies from parameter to parameter, and thus we need to study the problem of covering arrays with heterogeneous alphabet sizes. Some theoretical work [11] has been done recently on this problem, and there are some commercial covering array generators which generate arrays of this type (see e.g. [6]).

The precise formulation of the problem is as follows:

Let $\vec{g} = (g_1, g_2, ...g_k)$ be a vector of positive integers, and define the domain sets $D_i = \{0,1,2,..., g_i - 1\}$ for $1 \le i \le k$. A *testing array* of *type* $\vec{g}$ and *size* b is a $k \times b$ array $A = (a_{i,j})$ whose i-th row contains only members of the domain set $D_i = \{0,1,2,..., g_i - 1\}$ for each i. A testing array is said to be a *covering array* $CA(t, k, g_1, g_2, ..., g_k)$ if for any t distinct rows, the submatrix induced by the selected rows contains every member of the Cartesian product of the t domain sets as a column vector.

The classical covering array problem is to minimize the number of columns b for given fixed values of $t, k,$ and $g_1, g_2, ..., g_k$. We will use the notation $CAN(t, k, g_1, g_2, ..., g_k)$ to denote the size of a smallest covering array. We will abbreviate this to $CAN(t, k, g)$ when $g_1 = g_2 = ... = g_k = g$.

Moura et. al. [11] have created a theory for heterogeneous covering arrays with $t = 2$. They have almost completely solved the problem of constructing minimal arrays when $k \le 4$, and also solved large regions of the problem space when $k = 5$.

2.2 Covering arrays with budget constraints

Another practical limitation in the realm of testing is the budget. In most software development environments, the time, human, and computing

resources needed to perform the testing of a component is strictly limited. To model this situation, we consider the problem of creating the best possible test suite (covering the maximum number of *t*-tuples) within a fixed number of test cases (fixed number of columns of the array).

We define the *coverage measure* $\mu_t(A)$ of a testing array by the ratio between the number of distinct t-tuples contained in the column vectors of A and the total number of t-tuples $T_t(g_1, g_2, ...g_k)$ given by

$$T_t(g_1, g_2, ..., g_k) = \sum_X \prod_{i \in X} g_i$$

where the summation is over all subsets $X \subseteq \{1,2,...,k\}$ of cardinality t.

The **testing budget problem** is to construct a testing array A of size at most b having largest possible coverage measure, given fixed values of $t, k, b,$ and $\vec{g}$.

2.3 Embedding covering arrays

A third problem that arises in the real world is that of extending a given set of test vectors. Often the application has been tested in the past, and a set of regression tests has been used to guarantee compatibility and consistency with prior releases of the software. In this case we are interested in the minimal extension of a given array to achieve coverage of the *t*-tuples. (Curt Lindner is very fond of embedding problems – so this should be right up his alley.)

The **embedding problem** for a given positive integer t is to construct a covering array B of smallest size whose initial columns are precisely the columns of a given testing array A.

2.4 Minimizing covering arrays

A complementary problem is the construction of efficient **regression test** sets. Here one is given a t-covering array B, and one is required to find a smallest subset of the columns which retains the property of covering all the *t*-tuples.

The regression test problem is a special case of the classical set covering problem, which is known to be NP-complete. On the other hand, the standard covering array problem is a special case of the regression suite problem, where the input array A is just the universal array U of size $\prod_i g_i$ containing all the distinct column vectors.

2.5 Forbidden configurations

Another formulation of the regression suite problem comes from the so-called **forbidden configuration** problem.

The set of values used for testing a software component often contains one or two inadmissible data values. For example if a parameter was specified as taking positive integer values only, one might still want to test the software by inputting 0 or a negative number. One might also want the test vectors to contain not more than one inadmissible data value – so as to test the error handling of each parameter separately.

In general there may be an arbitrary set of column vectors which the tester regards as unacceptable members of a testing array. The forbidden configuration problem is specified by giving a testing array F of forbidden configurations, and asking for the construction of a covering array A which contains none of the columns of F. The problem has no solution if the complement, $\overline{F}$, of F with respect to U is not a covering array.

The forbidden configuration problem for F is equivalent to the regression suite problem for $\overline{F}$, and vice versa.

We have implemented heuristic solutions to these problems and made the software available on the internet for academic use. In the remainder of this paper we describe the software and the algorithms used to heuristically solve these problems. In the next section we summarize the results from the literature that we used.

3. BACKGROUND RESULTS

Elementary counting arguments show the following result:

Lemma 3.1: $CAN(t,k,g_1,g_2,...,g_k) \geq g_1 g_2 ... g_t$, and hence $g^k \geq CAN(t,k,g) \geq g^t$

The following monotonicity results are used heavily in the heuristics for constructing covering arrays.

Lemma 3.2: For all positive integer parameters, we have:

a) if $k < r$ then $CAN(t,k,g) \leq CAN(t,r,g)$

b) if $g_i \leq h_i$ for all *i* then $CAN(t,k,g_1,g_2,...g_k) \leq CAN(t,k,h_1,h_2,...h_k)$

c) if $g < h$ then $CAN(t,k,g) < CAN(t,k,h)$

The construction of orthogonal arrays of strength *t* for alphabets of size *g* a prime power, shows that the lower bound of Lemma 3.1 can be met provided that the number of rows is not larger than $g+1$.

Theorem 3.3 (Bush[3]): Let $g = p^{\alpha}$ be a prime power with $g > t$. Then $CAN(t,k,g) = g^t$ for all $k \le g+1$. Moreover, if $g \ge 4$ is a power of 2, then $CAN(t,k,g) = g^t$ for all $k \le g+2$.

Bush's generalization of Macneish's theorem for mutually orthogonal Latin squares also holds for orthogonal arrays of strength greater than 2.

Theorem 3.4 (Bush[2]): If $g = \prod q_j$ where the q_j are powers of distinct primes, then $CAN(t,k,g) = g^t$, for any $k \le 1 + \max(t, \min q_j)$.

Stevens, Ling, and Mendelsohn [14] give a construction for near optimal covering arrays using affine geometries over finite fields. It is one of the few constructions in the literature for covering arrays with heterogeneous alphabets.

Theorem 3.5(Stevens, Ling, and Mendelsohn [14]): Let $g = p^{\alpha}$ be a prime power then $CAN(2, g+2, g+1, g-1, g-1, \ldots, g-1) \le g^2 - 1$.

The size of the strength 2 covering array numbers are known precisely when all the domains are of size 2. This result was proved by Renyi using methods from extremal set theory. This result is actually couched in terms of the budget problem, in that it starts from a fixed number of columns and determines the maximum number of rows in a covering array of strength 2 over the binary alphabet.

Theorem 3.6 (Renyi [12]: For all $k > 1$ we have $CAN(2,k,2) = N$ where N is the smallest integer such that

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

The following recursive constructions are due to Williams. The first one appears to have been known also to Cohen, Dalal, Fredman, and Patton [6].

Theorem 3.7 (Williams [18]): If q is prime power, then

a) $CAN(2, kq+1, q) \le CAN(2,k,q) + q^2 - q$ and

b) $CAN(2, k(q+1), q) \le CAN(2,k,q) + q^2 - 1$

The following result (whose proof appears in [8]) is a common generalization of the results of Tang and Chen [15], Chateaneuf,

Colbourn, and Kreher [4], and Boroday [1]. It gives a method of squaring the number k of rows in a covering array of strength t while multiplying the number of columns by a factor dependent only on t and g, but independent of k. This factor is related to the Turan numbers $T(t,g)$ (see [17]) that are defined to be the number of edges in the Turan graph. The Turan graph is the complete *g*-partite graph with t vertices, having b parts of size $a+1$, and $g-b$ parts of size $a=\lfloor t/g \rfloor$ where $b=t-ga$. Turan's theorem (1941) states that among all *t*-vertex graphs with no $g+1$ cliques, the Turan graph is the one with the most edges.

Note that when $g \geq t$, $T(t,g)=t(t-1)/2$, and that when $g=2$, we have $T(t,2)=\lfloor t^2/4 \rfloor$.

Theorem 3.8: If $CAN(t,k,g)=N$ and there exist $T(t,g)-1$ mutually orthogonal Latin squares of side k (or equivalently $CAN(2,k,T(t,g)+1)=k^2$) then $CAN(t,k^2,g) \leq (T(t,g)+1)N$.

The following result is a doubling construction for covering arrays of strength 3 and 4. The result for strength 3 is due to Chateauneuf, Colbourn, and Kreher [4], while the result for strength 4 can be found in [8].

Theorem 3.9: For all positive integers g and k,

a) $CAN(3,2k,g) \leq CAN(3,k,g)+(g-1)CAN(2,k,g)$

b)
$CAN(4,2k,g) \leq CAN(4,k,g)+(g-1)CAN(3,k,g)+CAN(2,k,g^2)$

Finally we quote a result due to Moura, Stardom, Stevens and Williams for increasing the size of the largest domain.

Theorem 3.10 [11]: Let $e \geq 0$, and $g_1 \geq g_2 \geq ... \geq g_k$, then
$CAN(2,k,g_1+e,g_2,g_3,...,g_k) \leq CAN(2,k,g_1,g_2,...,g_k)+eg_2$

4. THE ALGORITHMS

In this section, we discuss how the results presented in section 3 can be used and extended by heuristic methods to solve practical problems in the generation of covering arrays and the solution of the other problems described in section 2. All of these algorithms are available in the Combinatorial Test Services (CTS) package [9] which we have created.

We first discuss the construction problem for covering arrays.

The main methods used to construct covering arrays are the direct and recursive constructions given in Theorems 3.3 to 3.10, and these are realized by straightforward implementations of the construction and recursion algorithms. However, Lemma 3.2, the monotonicity lemma, often produces smaller arrays when the alphabet size is not a prime power or one less than a prime power (when Theorems 3.3, 3.5, and 3.7 are applicable).

We begin by analyzing the parameter values t, k, and $\vec{g}$. We sort the vector $\vec{g}$, remove all those coordinates with $g_i = 1$, and replace it either by a constant vector with value g_1 (the maximum valued coordinate), or a vector whose first coordinate is g_1 and all remaining coordinates are equal to g_2. We now apply either Theorem 3.5 or 3.10 in the second case, or any of the other theorems in the first case. We also consider alphabets of sizes larger than g_1 in the event that neither g_1 nor $g_1 + 1$ are prime powers.

The resulting covering array is over a larger alphabet than required, and Lemma 3.2 can be used to reduce the alphabet sizes. When applying Lemma 3.2, we are also careful to remove any columns which contain fewer than t elements of the original alphabets, thus reducing the size of the array.

Another difficulty in programming the use of the results of Section 3 is in deciding which result to apply for any given parameter set, and which recursions to use. The CTS package tries several alternatives and chooses the smallest array that is constructed. Since all the algorithms are deterministic, the time involved in trying several alternatives is negligible.

In Section 5, Tables 1-3 we give the bounds on $CAN(t, k, g)$ for $t \leq 4$ produced by the CTS package. These results are almost always as good as the best theoretical results, the only exceptions being where special constructions have been used to give particular special case improvements. The other major difference between our approach and the theoretical approach is that not only does the CTS package provide the bounds, it also provides the array explicitly in several different formats. Other similar tables of results may be found in [4], [15], and [18].

We deal with the testing budget problem using a similar approach. The CTS package first constructs a covering array, and then orders its columns in such a way as to maximize the incremental coverage achieved by each new column. It does not explore all possible orderings of the columns, but rather takes a greedy approach, selecting the next column myopically using an algorithm whose complexity is quadratic in the number of columns. For larger arrays we use a linear algorithm which achieves similar results.

The same heuristic is also used for the selection of a regression suite.

The CTS heuristic for the embedding of a testing array is different from that given by Cohen, Dalal, Fredman, and Patton in [6] and also from that of Lei and Tai in [10]. Assume that we are given an array, and we are required to add a new column. We first find the set of t rows with the largest number of missing *t*-tuples, and select one of the missing tuples as the values in those rows. We then rank all the remaining (row, value) pairs by computing *t* values, ($p_0, p_1, \ldots p_{t-1}$) - which we call the *potential vector*. The first of these values p_0 is the amount by which the inclusion of the value *v* in the row *r* in the partial column would increase the coverage measure. In other words, p_0 counts the number of *t*-tuples containing *v* in row *r* and *t*-1 other values that have already been fixed in the partial column under construction. In general, p_i counts the total number of missing *t*-tuples containing *v* in row *r* as well as $t-1-i$ values that have already been fixed, and i undecided values in the other rows. We then choose the (row, value) pair with the lexicographically maximum potential vector. If several pairs achieve the same maximum potential vector, we break the tie by a random choice among those pairs that achieve the maximum.

Despite the fact that the forbidden configuration problem is theoretically equivalent to the regression testing problem, in practice it is not. (I once asked Curt Lindner "What is the difference between theory and practice?" He answered: "In theory, there is no difference...") It all depends on the representation used for covering arrays and forbidden configurations.

In our software implementation, a forbidden pair of values is represented by a single vector of length k with two entries from the domain sets, and the remaining $k-2$ entries set to a wildcard value. This representation forbids a set of g^{k-2} different columns, using only a single column vector.

The CTS package does not implement a construction algorithm which takes into account forbidden configurations, but rather it provides services for deleting and perturbing columns which contain a forbidden configuration.

There are many open theoretical and algorithmic problems which remain to be tackled. We hope that this paper will inspire others to improve on our results.

The CTS package which implements our algorithms will probably be made available in object code form for academic use. The package contains complete documentation and header files for the use of our combinatorial testing services.

5. TABLES OF COVERING ARRAY NUMBERS

k/g	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
3	4	9	16	25	36	49	64	81	100	
4	5	9	16	25	48	49	64	81	120	
5	6	15	16	25	48	49	64	81	120	
6	6	15	24	25	48	49	64	81	120	
7	6	15	28	45	48	49	64	81	120	
8	6	15	28	45	48	49	64	81	120	
9	6	15	28	45	62	63	64	81	120	
10	6	15	28	45	78	79	80	81	120	
11	7	15	28	45	78	91	120	119	120	
12	7	15	28	45	78	91	120	119	120	
13	7	15	28	45	78	91	120	153	166	
14	7	17	28	45	78	91	120	153	166	
15	7	17	28	45	78	91	120	153	210	
16	8	17	28	45	88	91	120	153	210	
17	8	21	28	45	88	91	120	153	210	
18	8	21	28	45	88	91	120	153	210	
19	8	21	28	45	88	91	120	153	210	
20	8	21	28	45	88	91	120	153	210	
21	8	21	28	45	88	91	120	153	210	
22	8	21	31	45	88	91	120	153	210	
23	8	21	31	45	90	91	120	153	210	
24	8	21	31	45	90	91	120	153	228	
25	8	21	31	45	90	91	120	153	228	
26	8	21	40	45	90	91	120	153	228	
27	8	21	40	45	90	91	120	153	228	
28	8	21	40	45	90	91	120	153	228	
29	8	21	40	45	90	91	120	153	228	
30	8	21	40	45	90	91	120	153	228	
k/g	11	12	13	14	15	16	17	18	19	20
3	121	144	169	196	225	256	289	324	361	400
4	121	144	169	252	225	256	289	360	361	400
5	121	168	169	254	255	256	289	360	361	400
6	121	168	169	254	255	256	289	360	361	526
7	121	168	169	254	255	256	289	360	361	526
8	121	168	169	254	255	256	289	360	361	526
9	121	168	169	254	255	256	289	360	361	526
10	121	168	169	254	255	256	289	360	361	526
11	121	168	169	254	255	256	289	360	361	526
12	121	168	169	254	255	256	289	360	361	526
13	167	168	169	254	255	256	289	360	361	526
14	167	168	169	254	255	256	289	360	361	526
15	231	252	253	254	255	256	289	360	361	526
16	231	252	253	254	255	256	289	360	361	526
17	231	252	253	254	255	256	289	360	361	526
18	231	284	285	286	287	288	289	360	361	526

19	231	300	325	356	357	358	359	360	361	526
20	231	300	325	356	357	358	359	360	361	526
21	231	300	325	436	465	496	523	524	525	526
22	231	300	325	436	465	496	523	524	525	526
23	231	300	325	436	465	496	523	524	525	526
24	231	300	325	436	465	496	523	524	525	526
25	231	300	325	436	465	496	561	618	619	620
26	231	300	325	436	465	496	561	618	619	620
27	231	300	325	436	465	496	561	666	703	722
28	231	322	325	436	465	496	561	666	703	722
29	231	322	325	436	465	496	561	666	703	832
30	231	322	325	436	465	496	561	666	703	832

Table 1: A list of the upper bounds on $CAN(2,k,g)$ provided by our software.

k/g	2	3	4	5	6	7	8	9	10	11	12	13	14
4	8	27	64	125	216	343	512	729	1000	1331	1728	2197	2744
5	12	45	64	125	336	343	512	729	1324	1331	2190	2197	4052
6	12	45	64	125	342	343	512	729	1330	1331	2196	2197	4076
7	13	45	112	225	342	343	512	729	1330	1331	2196	2197	4092
8	13	45	112	225	342	343	512	729	1330	1331	2196	2197	4092
9	18	75	112	225	510	511	512	729	1330	1331	2196	2197	4094
10	18	75	112	225	510	511	512	729	1330	1331	2196	2197	4094
11	18	75	136	225	634	637	960	1329	1330	1331	2196	2197	4094
12	18	75	136	225	634	637	960	1329	1330	1331	2196	2197	4094
13	19	75	196	405	634	637	960	1377	2194	2195	2196	2197	4094
14	19	75	196	405	634	637	960	1377	2194	2195	2196	2197	4094
15	19	75	196	405	634	637	960	1377	2538	2541	4092	4093	4094
16	19	75	196	405	634	637	960	1377	2538	2541	4092	4093	4094
17	24	105	196	405	886	889	960	1377	2538	2541	4092	4093	4094
18	24	105	196	405	886	889	960	1377	2538	2541	4092	4093	4094
19	24	105	196	405	982	985	1072	1377	2538	2541	4222	4225	6854
20	24	105	196	405	982	985	1072	1377	2538	2541	4222	4225	6854
21	25	105	220	405	1156	1183	1800	2281	2538	2541	4222	4225	7926
22	25	105	220	405	1156	1800	1800	2281	2538	2541	4222	4225	7926
23	25	105	220	405	1156	1800	1800	2281	2538	2541	4222	4225	7926
24	25	105	220	405	1156	1800	1800	2281	2538	2541	4222	4225	7926
25	26	105	280	585	1156	1800	1800	2601	3862	3865	4222	4225	7926
26	26	105	280	585	1156	1800	1800	2601	3862	3865	4222	4225	7926
27	26	117	280	585	1156	1800	1800	2601	3862	3865	4222	4225	7926
28	26	117	280	585	1156	1800	1800	2601	3862	3865	4222	4225	7926
29	26	117	280	585	1156	1800	1800	2601	4808	4851	7126	7129	7926
30	26	117	280	585	1156	1800	1800	2601	4808	4851	7126	7129	7926

Table 2: A list of the upper bounds on $CAN(3,k,g)$ provided by our software.

g/k	2	3	4	5	6	7	8
5	24	135	256	625	2036	2401	4096
6	28	153	564	625	2300	2401	4096
7	38	207	688	1725	2380	2401	4096
8	42	207	696	1725	2400	2401	4096
9	50	285	696	1725	4062	4095	4096
10	50	309	696	1725	6534	6553	6560

Table 3: A list of the upper bounds on $CAN(4,k,g)$ provided by our software.

Parameter sizes	IPO	TConfig	AETG	CTS
3^4	10	9	9	9
3^{13}	20	15	15	15
$4^{15}3^{17}2^{29}$	34	40	41	39
$4^{1}3^{39}2^{35}$	27	30	28	29
2^{100}	15	14		10
10^{20}	219	231	180	210
4^{10}	31	28		28
4^{20}	34	28		28
4^{30}	41	40		40
4^{40}	42	40		40
4^{50}	47	40		40
4^{60}	47	40		40
4^{70}	49	40		40
4^{80}	49	40		40
4^{90}	52	43		43
4^{100}	52	43		43

Table 4: A comparison of the bounds obtained for covering arrays of strength 2 by four different algorithmic approaches. We use the exponential notation for the vector of alphabet sizes. A similar table was originally published in [18]. We have extended the table to include the results of [6] and those of this paper. The IPO algorithm is reported in [10], the TConfig algorithm is reported in [18], the AETG algorithm is

reported in [6], and the last column, CTS, represents the results reported in this paper.

NOTE ADDED IN PROOF

A recently published paper by N. Kobayashi, T. Tsuchiya, and T. Kikuno, A new method for constructing pair-wise covering designs for software testing, Information Processing Letters 81 (2002) 85-91, includes some new algorithms and heuristics for covering arrays with heterogeneous alphabets. This paper claims new world records for the 3rd and 4th lines in Table 4 with their bounds being $CAN(2,4^{15}3^{17}2^{29}) \le 31$ and $CAN(2,4^{1}3^{39}2^{35}) \le 24$.

REFERENCES

1. Boroday S. Y.*, **Determining essential arguments of Boolean functions***. (in Russian), in Proceedings of the Conference on Industrial Mathematics, Taganrog 1998, 59-61. The translation is a personal communication from the author.

2. Bush K. A., ***A generalization of the theorem due to MacNeish***. Ann. Math. Stat. 23 (1952) 293-295.

3. Bush K. A., ***Orthogonal arrays of index unity.*** Annals of Mathematical Statistics 23 (1952) 426-434.

4. Chateauneuf M. A. and Kreher D. L., ***On the state of strength 3 covering arrays.*** J. Combinatorial Designs, 10 (2002) 217-238.

5. Chateauneuf M. A., Colbourn C. J., and Kreher D. L., ***Covering arrays of strength 3***. Designs, Codes, and Cryptography, 16 (1999) 235-242.

6. Cohen D. M., Dalal S. R., Fredman M. L., and Patton G. C., ***The AETG System: An approach to Testing Based on Combinatorial Design.*** IEEE Transactions on Software Engineering, 23 (1997), 437-444.

7. Colbourn C. J. and Dinitz J. H., ***The CRC Handbook of Combinatorial Designs***. CRC Press 1996.

8. Hartman A. ***Software and hardware testing using combinatorial covering suites.*** to appear in Interdisciplinary Applications of Graph Theory, Combinatorics and Algorithms (ed. M. C. Golumbic).

9. Hartman A. and Raskin L. Combinatorial Test Services – a software package. Available at http://www.alphaworks.ibm.com/.

10. Lei Y. and Tai K. C., ***In-parameter order: A test generation strategy for pairwise testing***. in Proc. 3rd IEEE High Assurance Systems Engineering Symposium, (1998) 254-161.

11. Moura L., Stardom J., Stevens B., and Williams A., *Covering arrays with heterogeneous alphabet sizes*, preprint June 2002.

12. Rényi A., ***Foundations of probability***. Wiley, New York, 1971.

13. Roux G., ***k-propriétés dans les tableaux de n colonnes; cas particulier de la k-surjectivité et de la k-permutativité***. Ph. D. Dissertation, University of Paris 6, 1987.

14. Stevens B., Ling A., and Mendelsohn E., *A direct construction of transversal covers using group divisible designs*. Ars Combin. 63 (2002) 145-159.

15. Stevens B. and Mendelsohn E. *New recursive methods for transversal covers*, Journal of Combinatorial Designs, 7, 185-203, 1999.

16. Tang D. T. and Chen C. L.***, Iterative exhaustive pattern generation for logic testing***. IBM J. Res. Develop. 28 (1984), 212-219.

17. West D. B.***, Introduction to Graph Theory***. Prentice Hall NJ, 1996.

18. Williams A. W.***, Determination of Test Configurations for Pair-Wise Interaction Coverage***. in Proceedings of the 13th International Conference on the Testing of Communicating Systems (TestCom 2000), Ottawa Canada, 2000, 59-74.