LE MATEMATICHE
Vol. LIX (2004) – Fasc. I–II, pp. 125–172

COMBINATORIAL ASPECTS OF COVERING ARRAYS

CHARLES J. COLBOURN

Covering arrays generalize orthogonal arrays by requiring that t-tuples be covered, but not requiring that the appearance of t-tuples be balanced. Their uses in screening experiments has found application in software testing, hardware testing, and a variety of fields in which interactions among factors are to be identified. Here a combinatorial view of covering arrays is adopted, encompassing basic bounds, direct constructions, recursive constructions, algorithmic methods, and applications.

1. Mathematical Preliminaries.

An *orthogonal array* $OA_\lambda(N; t, k, v)$ is an $N \times k$ array. In every $N \times t$ subarray, each t-tuple occurs exactly λ times, where $\lambda = \frac{N}{v^t}$. The parameter t is the *strength*; k is the number of *factors*; and v is the number of levels associated with each factor, the *order*. The requirement that every t-tuple arise exactly λ times can be too restrictive in applications that require only that every t-tuple be covered at least once. We therefore relax the definition to introduce the covering array and mixed-level covering array.

A *covering array* $\mathrm{CA}_\lambda(N; t, k, v)$ is an $N \times k$ array. In every $N \times t$ subarray, each t-tuple occurs at least λ times. Then t is the *strength* of the coverage of interactions, k is the number of components (*degree*), and v is the number of

Keywords: Covering array, qualitative independence, transversal cover, orthogonal array, transversal design, perfect hash family, interaction testing, group-divisible designs, heuristic search, greedy algorithm.

symbols for each component (*order*). We treat only the case when $\lambda = 1$, i.e. every t-tuple must be covered at least once. The size of a covering array is the *covering array number* $\mathrm{CAN}(t, k, v)$. The covering array is *optimal* if it contains the minimum possible number of rows. Various authors transpose the array in the definition, and of course this is a matter of personal preference. In our discussions here, we employ the $N \times k$ format, but occasionally construct the tranposed covering array.

Here is an example of a covering array of strength three with ten factors having two levels each. It has $N = 13$ rows.

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

This combinatorial object is fundamental in developing interaction tests when all factors have an equal number of levels. However, systems are typically not composed of components *(factors)* that each have exactly the same number of parameters *(levels)*. To remove this limitation of covering arrays, the mixed-level covering array can be used.

A *mixed level covering array* $\mathrm{MCA}_\lambda(N; t, k, (v_1, v_2, \ldots, v_k))$ is an $N \times k$ array. Let $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$, and consider the subarray of size $N \times t$ obtained by selecting columns $i_1, \ldots, i_t$ of the MCA. There are $\prod_{i=1}^{t} v_i$ distinct t-tuples that could appear as rows, and an MCA requires that each appear at least once. We use the notation $\mathrm{CAN}(t, k, (v_1, v_2, \ldots, v_k))$ to denote the smallest N for which such a mixed covering array exists.

An early investigation of covering arrays appears implicitly in Marczewski [87]. Rényi [107] determined sizes of covering arrays for the case $t = v = 2$ when N is even. Kleitman and Spencer [76] and Katona [74] independently determined covering array numbers for all N when $t = v = 2$. They showed that N grows as follows:

$$k = \binom{N-1}{\lceil \frac{N}{2} \rceil}$$

For large k, N grows logarithmically. The construction is straightforward. Form a matrix in which the columns consist of all distinct binary N-tuples of weight $\lceil \frac{N}{2} \rceil$ that have a 0 in the first position. In 1990 Gargano, Körner and Vaccaro [58] gave a probabilistic bound when $t = 2$ and $v > 2$:

$$N = \frac{v}{2} \log k(1 + o(1))$$

Now we explore a dual formulation. Let C be an $N \times k$ covering array. Suppose that rows are indexed by a set R of size N. Then each column can be viewed as a partition of R into exactly v classes $(M_1, \dots, M_v)$; the class M_i of a symbol $r \in R$ is determined by the value i appearing in row r in the chosen column. In this manner, such an array gives a collection $\mathcal{P} = \{R_1, R_2, \dots, R_k\}$ of partitions of R. A family of partitions is *t-qualitatively independent* when for every t of the partitions $R_{i_1}, \dots R_{i_t}$, and for every choice of classes $M_{i_j} \in R_{i_j}$, for $1 \leq j \leq t$, we find that $\bigcap_{j=1}^{t} M_{i_j} \neq \emptyset$. It follows that covering arrays of strength t having N rows are the same as t-qualitatively independent partitions of a set of size N. Many of the early results were established using this vernacular, but we for the most part translate to the language of covering arrays. Poljak, Pultr, and Rödl [105] and Körner and Lucertini [79] discuss combinatorial problems related to qualitative independence.

Covering arrays appear in other mathematical disguises as well. A (k, t)-*universal set* is a subset of $\{0, 1\}^k$ such that the projection on every t coordinates contains all 2^t combinations. Hence it is a CA$(t, k, 2)$. Naor and Naor [94] establish that (k, t)-universal sets arise as probability spaces with limited independence; indeed these have been extensively studied as ϵ-biased arrays [2], [80], [83], [94], [95]. Bierbrauer and Schellwatt [5] extend this framework to more than two values per symbol.

The nomenclature *t-surjective array* for a covering array of strength t is also used, to indicate that on each t columns, every possible outcome arises. See [1], [17], [28], [29], [56], [111], for example.

2. Bounds and Asymptotics.

An obvious lower bound is:

$$v^t \leq \text{CAN}(t, k, v). \tag{1}$$

More generally for a $\mathrm{MCA}_\lambda(N; t, k, (v_1, v_2, \ldots, v_k))$, $\prod_{i=1}^{t} v_i$ is a lower bound on the size of the mixed covering array. Although the $\{v_i\}$ are listed in a specified order, reordering the $\{v_i\}$ in the definition results in a simple column reordering of the MCA. For this reason, we present these sizes in any convenient order. In this way, the stated bound can be treated as the product of the t largest values among the $\{v_i\}$.

Evidently if any of the $\{v_i\}$ is 1, it can be omitted provided that the number of factors is at least two. With this in mind, a simple inequality establishes that

$$\mathrm{CAN}(t, k, (v_1, v_2, \ldots, v_k) \leq \mathrm{CAN}(t, k, (v_1 + 1, v_2, \ldots, v_k) \tag{2}$$

and consequently

$$\mathrm{CAN}(t, k-1, (v_2, \ldots, v_k) \leq \mathrm{CAN}(t, k(v_1, v_2, \ldots, v_k) \tag{3}$$

Suppose A is a covering array $\mathrm{CA}(N; t, k, v)$ and let i be any row and x any symbol. Then the $(k-1) \times N'$ subarray obtained by deleting row i from A and keeping only those columns of A that had symbol x in row i is a $\mathrm{CA}(N'; t-1, k-1, v)$, where N' is the number of occurrences of x in row i. Hence

$$\mathrm{CAN}(t-1, k-1, v) \leq \frac{1}{v}\,\mathrm{CAN}(t, k, v). \tag{4}$$

By the same token,

$$\mathrm{CAN}(t, k, (v_1, v_2, \ldots, v_k)) \geq v_1 \cdot \mathrm{CAN}(t-1, k-1, (v_2, \ldots, v_k)) \tag{5}$$

More sophisticated bounds have been the subject of much research, as we shall see. In this section, we outline some asymptotic and probabilistic results.

Godbole, Skipper, and Sunley [60] examine the random process of choosing, in each entry of an $N \times k$ array, each of v possible values equiprobably. They establish that when N is large enough with respect to t, k, and v, such a random array has nonzero probability of being a $\mathrm{CA}(N; t, k, v)$. It follows from their results that

$$\mathrm{CAN}(t, k, v) \leq \frac{(t-1)\log k}{\log \frac{v^t}{v^t - 1}}(1 + \mathrm{o}(1)).$$

In [4], [94], [96], it is established that

$$\mathrm{CAN}(t, k, 2) \leq 2^t t^{\mathrm{O}(\log t)} \log k.$$

Gargano, Körner, and Vaccaro [57] show that the ratio of CAN$(2, k, v)$ to $\log k$ is asymptotic to $\frac{1}{2}v$. Indeed a stronger version in [58] establishes the same result when only a small fraction of the pairs need to be covered. For higher strength, considering the largest k for which a CA$(N; t, k, v)$ exists, it has been established that k is at least $\frac{et}{v}e^{\frac{N}{tv^t}}$ and at most $\frac{\kappa_{v,N}}{\sqrt{N}}4^{\frac{N}{v^{t-1}}}$, where $\kappa_{v,N}$ is a constant depending only upon v and N; see [105], [106].

Lower bounds are in general not well explored, but see [71], [122], for example.

3. Algebraic and Combinatorial Methods.

Evidently an orthogonal array OA$(N; t, k, v)$ is a covering array, and when $N = v^t$ it is optimal. The theory of orthogonal arrays is well developed; indeed a recent book treats many facets of their construction and use [66]. Although orthogonal arrays both provide many of the best constructions, and motivate many of the methods to be described, we do not delve into them here but instead refer the reader to [64] for an overview in the context of covering arrays, to [37], [38] for OA$(2, k, v)$s, and to [66] for higher strength.

We mention one important direction here. A *group divisible design* $(V, \mathcal{G}, \mathcal{B})$ $(k - GDD)$ is a set V of v elements, a partition $\mathcal{G}$ of V into *groups*, and a collection of k-subsets of V (*blocks*) with the property that every block intersects every group in at most one element, and every pair of elements from different groups appears as a subset of exactly one block. When there are n groups, and their sizes are $g_1, \dots, g_n$, the *type* of the k-GDD is $g_1 g_2 \cdots g_n$. Exponential notation is used, so that g^ℓ indicates ℓ occurrences of the group size g. A k-GDD of type v^k is a *transversal design* TD(k, v). Such a design is equivalent to a certain orthogonal array (see [38], [66]), and hence gives a covering array. Moreover the generalization of transversal designs to allow pairs of elements in different groups to be covered *at least* once leads to another formulation of covering arrays, *transversal covers*. Stevens, Ling, and Mendelsohn [118] observe that when a k-GDD of type v^n exists and $k < n$, one can simply extend every block to add points from each group that the block does not intersect. The result is a transversal cover; thus when a k-GDD of type v^n exists, we can deduce that CAN$(2, n, v) \leq \frac{v^2 n(n-1)}{k(k-1)}$. This idea can be carried further, by adjoining further groups to the k-GDD. See [118] for some constructions in this vein; of interest is the special case that CAN$(2, v+3, v) \leq v^2 + 2v$ when $v + 1$ is a prime or prime power.

3.1. Some Direct Constructions for Strength Two

Allowing a finite group to act on the symbols of the covering array enables one to develop arrays more compactly by choosing suitable orbit representatives. Using cyclic groups leads to the notion of *difference covering arrays* (for strength two) treated in [138], [139]. In general, let Γ be a group of order v, with $\odot$ as its binary operation. A *difference covering array* $D = (d_{ij})$ over Γ, denoted by $\mathrm{DCA}(N, \Gamma; 2, k, v)$, is an $N \times k$ array with entries from Γ having the property that for any two distinct columns j and ℓ, $\{d_{ij} \odot d_{i\ell}^{-1} : 1 \leq i \leq N\}$ contains every *non-identity* element of Γ at least once. When Γ is abelian, additive notation is used, explaining the "difference" terminology. We often employ the case when $\Gamma = \mathbb{Z}_v$, and omit it from the notation. We denote by $\mathrm{DCAN}(2, k, v)$ the minimum N for which a $DCA(N, \mathbb{Z}_v; 2, k, v)$ exists.

In a similar manner, in [39], [89], arrays developed with one fixed point and a $(v-1)$-cycle are produced. We outline some work in the latter direction here. Choose two parameters, ℓ and g. We form a vector $(v_0, \dots, v_{\ell-1})$ with entries from $\mathbb{Z}_{g-1} \cup \{\infty\}$. The set $D_s = \{(v_j - v_i) \bmod g - 1 : j - i \equiv s \bmod \ell, v_i \neq \infty, v_j \neq \infty\}$ consists of the s-apart differences. Consider vectors in which $v_0 = \infty$ and $v_i \in \mathbb{Z}_{g-1}$ for $1 \leq i < \ell$. When $D_s = \mathbb{Z}_{g-1}$ for $1 \leq s < \ell$, such a vector is a (g, ℓ)-*cover starter*. When $\mathbb{Z}_{g-1} \setminus \{0\} \subseteq D_s$ for each $1 \leq s < \ell$, and $\{v_1, \dots, v_{\ell-1}\} = \mathbb{Z}_{g-1}$, such a vector is a (g, ℓ)-*distinct cover starter*.

When a (g, ℓ)-cover starter exists, Meagher and Stevens [89] note that a $\mathrm{CA}(\ell(g-1)+1; 2, \ell, g)$ exists, as follows. Form all ℓ cyclic shifts of the cover starter. For each, form $g-1$ vectors by developing each modulo $g-1$ (keeping ∞ fixed). The resulting $\ell(g-1)$ vectors form the rows of an array, so that for any two columns all pairs are covered except for (∞, ∞). Adding one constant row consisting only of ∞ completes the covering array. Meagher and Stevens [89] give numerous examples of cover starters, proving

Lemma 3.1. *A (g, ℓ)-cover starter exists when*

1. $g = 3$ *and* $\ell \in \{5, 8\}$*;*
2. $g = 4$ *and* $\ell \in \{5, 6, 7, 8, 9, 10\}$*;*
3. $g = 5$ *and* $\ell \in \{7, 8, 9, 10, 11, 12\}$*;*
4. $g = 6$ *and* $\ell \in \{9, 10, 11, 12, 13, 14\}$*;*
5. $g = 7$ *and* $\ell \in \{10, 11, 12, 13, 14, 15, 16\}$*;*
6. $g = 8$ *and* $\ell \in \{9, 11, 12, 13, 14, 15, 16, 17, 18\}$*;*
7. $g = 9$ *and* $\ell \in \{13, 14, 15, 16, 17, 18, 19, 20\}$.

Distinct cover starters are instead used in [39]. Next we give examples of distinct cover starters:

(6,8)	$(\infty,0,1,3,0,2,1,4)$	(6,9)	$(\infty,0,0,1,0,0,3,2,4)$
(6,10)	$(\infty,0,0,0,1,1,4,3,0,2)$	(6,11)	$(\infty,0,0,0,0,1,0,4,2,3,0)$
(6,12)	$(\infty,0,0,0,0,0,1,0,4,2,3,0)$	(7,7)	$(\infty,0,2,1,4,5,3)$
(7,9)	$(\infty,0,0,2,1,4,5,3,3)$	(7,10)	$(\infty,0,0,0,1,0,3,5,4,2)$
(7,11)	$(\infty,0,0,0,0,1,0,3,5,4,2)$	(7,12)	$(\infty,0,0,0,0,0,1,0,3,5,4,2)$
(7,13)	$(\infty,0,0,0,0,0,0,1,0,3,5,4,2)$	(8,11)	$(\infty,0,1,0,0,0,3,5,4,2,6)$
(8,12)	$(\infty,0,0,0,1,0,6,4,6,3,2,5)$	(8,13)	$(\infty,0,0,0,0,1,2,6,5,3,6,2,4)$
(8,14)	$(\infty,0,0,0,0,0,1,0,4,6,4,3,2,5)$	(9,12)	$(\infty,0,1,6,4,5,0,7,6,2,1,3)$
(9,13)	$(\infty,0,0,0,0,2,6,1,7,6,3,4,5)$	(9,14)	$(\infty,0,0,0,0,0,2,6,1,7,6,3,4,5)$
(9,15)	$(\infty,0,0,0,0,0,0,2,6,1,7,6,3,4,5)$		

3.2. Some Direct Constructions for Strength Three

Here we examine group action focussing on strength three, following the presentation in [19]. Let Ω be the set of v symbols on which we are to construct a CA$(N, 3, k, v)$. Let G be a group acting on the set Ω. If $g \in G$ and M is a $k \times \ell$ matrix with entries in Ω, then M^g is the $k \times \ell$ matrix whose $[i, j]$ entry is $M[i, j]^g$, the image of $M[i, j]$ under g. The matrix obtained by developing M by G is the $k \times \ell|G|$ matrix

$$M^G = [M^g : g \in G].$$

Let $C = C(k, \Omega)$ be the $k \times |\Omega|$ matrix that has a constant column with each entry equal to x, for each $x \in \Omega$. When the k rows are indexed by a set X, the notation $C(X, \Omega)$ is also used. The goal is to choose the matrix M and group G so that $[M^G, C]^T$ or just $[M^G]^T$ is a CA$(N, 3, k, v)$. For example, when $G = \text{Sym}\{0, 1, 2\}$ (the symmetric group on $\{0, 1, 2\}$), and

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 2 & 1 & 0 \end{bmatrix},$$

$[M^G, C]^T$ is a CA(3,4,3).

Theorem 3.2. *Let $v > 2$ be a positive integer, and let $q \geq v - 1$ be a prime power. Then there is a CA$(2v - 1)(q^3 - q) + v; 3, 2v, v)$.*

Proof. First we construct a CA(N; 3, $2v$, $q+1$), with

$$N = (2v-1)(q^3-q)+q+1.$$

Since q is a prime power, the group $G = PGL(q)$ is sharply 3-transitive on the projective line $\Omega = \mathbb{F}_q \cup \{\infty\}$. Under this group there are precisely five orbits of 3-tuples. These five orbits are determined by the pattern of the entries in their 3-tuples:

1. $\{[a,a,a]^T : a \in \Omega\}$
2. $\{[a,a,b]^T : a,b \in \Omega, a \neq b\}$
3. $\{[a,b,a]^T : a,b \in \Omega, a \neq b\}$
4. $\{[b,a,a]^T : a,b \in \Omega, a \neq b\}$
5. $\{[a,b,c]^T : a,b,c \in \Omega, a \neq b \neq c \neq a\}$

Let x_1, x_2, x_3 be any three rows of $[M^G, C(X,\Omega)]$. The 3-tuples with all equal entries occur on rows x_1, x_2, x_3 of $[M^G, C(X,\Omega)]$ since they occur in $C(X,\Omega)$. Thus to construct a CA(N; 3, $2v$, $q+1$) using this group we need only find a matrix M such that for each of the orbits 2-5 and each set of 3 rows there is a column of M that contains an orbit representative for the orbit on the chosen rows.

Let X be a $2v$-element set of vertices and let

$$\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{2v-1}$$

be a one-factorization of the complete graph $\mathcal{G}$ on X. Let $\ell : E(\mathcal{G}) \to \Omega$ be an arbitrary function for which $\ell(e) \neq \ell(e')$ whenever $e, e' \in \mathcal{F}_j$ for some j. Such a function exists since $|\Omega| = q+1 > v$. Define the $2v \times (2v-1)$ matrix M with entries in Ω by $M[x,j] = \ell(e)$, where e is the edge of $\mathcal{F}_j$ that is incident to $x \in X$.

It must be shown that each of the orbits 2, 3, 4, and 5 has a representative on rows x_1, x_2, x_3. The edge $e = \{x_1, x_2\}$ is an edge of some one-factor $\mathcal{F}_j$ and x_3 is incident to some other edge e' of $\mathcal{F}_j$. Thus $\ell(e) = a$ and $\ell(e') = b$ for some $a \neq b$ in Ω. Consequently

$$[M[x_1,j], M[x_2,j], M[x_3,j]] = [a,a,b]$$

and so orbit (2) is represented. Similarly, orbits (3) and (4) are represented. There are $2v-4$ one-factors that do not contain any of the edges $\{x_1,x_2\}, \{x_1,x_3\}$, or$\{x_2,x_3\}$. Thus, since $2v \geq 5$, there is a one-factor $\mathcal{F}_{j'}$ in which x_1, x_2, and x_3 are each incident to different edges. Consequently, column j' of M has distinct entries on rows x_1, x_2, and x_3 and thus orbit (5) is represented. Therefore $[M^G, C(X, \Omega)]^T$ is a CA$(N; 3, 2v, q+1)$, with $N = (2v-1)(q^3-q)+q+1$. To obtain a CA$(N'; 3, 2v, v)$ with $N' = (2v-1)(q^3-q)+v$ replace $q+1-v$ of the non-zero symbols with 0 and delete the $q+1-v$ extra columns of 0's in $C(X, \Omega)$. □

When $q = 2$, $PGL(2, q)$ is isomorphic to Sym$\{0, 1, 2\}$. Figure 1 depicts the CA(33; 3, 6, 3) array (transposed) that is constructed using this group by Theorem 3.2.

Theorem 3.3. *The covering number* $CAN(3, 6, 3) = 33$.

0 1 2 2 1	1 2 0 0 2	2 0 1 1 0	0 2 1 1 2	1 0 2 2 0	2 1 0 0 1	0 1 2
1 2 2 1 0	2 0 0 2 1	0 1 1 0 2	2 1 1 2 0	0 2 2 0 1	1 0 0 1 2	0 1 2
2 2 1 0 1	0 0 2 1 2	1 1 0 2 0	1 1 2 0 2	2 2 0 1 0	0 0 1 2 1	0 1 2
2 1 0 1 2	0 2 1 2 0	1 0 2 0 1	1 2 0 2 1	2 0 1 0 2	0 1 2 1 0	0 1 2
1 0 1 2 2	2 1 2 0 0	0 2 0 1 1	2 0 2 1 1	0 1 0 2 2	1 2 1 0 0	0 1 2
0 0 0 0 0	1 1 1 1 1	2 2 2 2 2	0 0 0 0 0	1 1 1 1 1	2 2 2 2 2	0 1 2

Figure 1: A minimal CA(33; 3, 6, 3) covering array, transposed

Proof. Östergärd has shown that CAN$(2, 5, 3) = 11$ (see [115]). Thus by inequality (4), CAN$(3, 6, 3) \geq 33$. But by Theorem 3.2, CAN$(3, 6, 3) \leq 33$. □

Example 3.4 illustrates a situation not using 3-transitive group.

Example 3.4. A CA(88; 3, 8, 4) covering array.

Let $X = \mathbb{F}8 = \mathbb{Z}_2[x]/(x^3+x+1)$ be the points of the complete graph K_8. Observe that x is a primitive root of X. Let $\mathcal{F}$ be the cosets of $H = \{0, 1\}$ as a subgroup of the additive group of $\mathbb{F}8$. Then

$$\mathcal{F} = \{\{0, 1\}, \{x, x^3\}, \{x^2, x^6\}, \{x^4, x^5\}\}$$

is a one factor and the partition

$$\{x\mathcal{F}, x^2\mathcal{F}, x^3\mathcal{F}, x^4\mathcal{F}, x^5\mathcal{F}, x^6\mathcal{F}\}$$

is a one-factorization of K_8. Let $\ell : E(K_8) \to \Omega$ be any function such that $\ell(e) \neq \ell(e')$ whenever $e, e' \in x^j\mathcal{F}$ for some j and define the 8×7 matrix M with entries in Ω by $M[x, j] = \ell(e)$ where e is the edge of $x^j\mathcal{F}$ that is incident to $x \in X$. One such matrix is given in Figure 2.

Let $G = \mathrm{Alt}\Omega$ be the alternating group of permutations of Ω. Then $[M^G, C(X, \Omega)]^T$ is the desired CA(88; 3, 8, 4) covering array. To see this, consider any three rows x_1,x_2,x_3 of $[M^G, C(X, \Omega)]$. It must be shown that, for each choice of a, b, c with $a \neq b \neq c \neq a$, each of the patterns $[a, a, a]^T$, $[a, a, b]^T$, $[a, b, a]^T$, $[b, a, a]^T$, and $[a, b, c]^T$ occurs on these rows. Patterns with three equal entries occur in $C = C(X, \Omega)$. Patterns with two equal entries and one different appear in $[M^G, C(X, \Omega)]$ on rows x_1, x_2, x_3 since G is 2-transitive on Ω and every pair occurs as an edge. This leaves only patterns with three distinct entries. There are exactly two orbits of such patterns under G since G is the alternating group on four symbols. Thus it suffices that in M there are two triples $[x, y, z]^T$ and $[x', y', z']^T$ on rows i, j, k for which the unique permutation in the symmetric group that sends one on to the other is an odd permutation.

$$\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 3 & 3 & 1 & 2 & 1 \\
1 & 0 & 2 & 3 & 3 & 1 & 2 \\
2 & 1 & 0 & 2 & 3 & 3 & 1 \\
1 & 2 & 1 & 0 & 2 & 3 & 3 \\
3 & 1 & 2 & 1 & 0 & 2 & 3 \\
3 & 3 & 1 & 2 & 1 & 0 & 2 \\
2 & 2 & 2 & 1 & 2 & 1 & 0
\end{array}$$

Figure 2: A matrix for the one-factorization in Construction 3.4

Corollary 3.5. *The covering number* $CAN(3, 8, 4) \leq 88$ *and* $CAN(2, 7, 4) \leq 22$.

Proof. The construction in Example 3.4 and the inequality (4) establish this result. □

Generalizations appear in [18], [20].

3.3. Direct Constructions for Strength $t \geq 4$

Beyond consequences of known orthogonal arrays [66], Sherwood [114] develops a compact "vector notation" for covering arrays, and demonstrates existence of a number of arrays with $t = 4$. This appears to be a promising search technique, but is too involved for presentation here.

4. Recursive Constructions.

In this section we describe recursive constructions, primarily for $t \in \{2, 3\}$.

4.1. Products of Strength Two Arrays

When a CA$(N; 2, k, v)$ and a CA$(M; 2, \ell, v)$ both exist, it is an easy matter to produce a CA$(N + M; 2, k\ell, v)$. To be specific, let $A = (a_{ij})$ be a CA$(N; 2, k, v)$ and let $B = (b_{ij})$ be a CA$(M; 2, \ell, v)$. Form an $(N + M) \times k\ell$ array $C = (c_{i,j})$ by setting $c_{i,(f-1)k+g} = a_{i,g}$ for $1 \le i \le N$, $1 \le f \le \ell$, and $1 \le g \le k$. Then set $c_{N+i,(f-1)k+g} = b_{i,f}$ for $1 \le i \le M$, $1 \le f \le \ell$, and $1 \le g \le k$. In essence, k copies of B are being appended to ℓ copies of A. Since two different columns of C arise either from different columns of A or from two different columns of B, the result is a CA$(N + M; 2, k\ell, v)$. This is the essence of the *block recursive construction* from [120], and can be traced back through work of Cohen and Fredman [27] to a product for qualitative independence by Poljak and Tuza [106]. Williams et al. [131], [133], [134] use this product construction in order to design tests for the interactions of nodes in a network and component based testing, and produce a software package called *TConfig*. Recent extensions are given in [39], [91].

It can happen that some of the entries of the array are not needed in order to cover all t-tuples. In this case, we can replace the entry of the array by $\star$, to indicate a “don’t care” position. When such a replacement is made, t-tuples containing a $\star$ are deemed not to match a t-tuple of the v symbols. The *profile* $(d_1, \dots, d_k)$ of an $N \times k$ array is a k-tuple in which the entry d_i is the number of $\star$ entries in the ith column.

In [39], two extensions of this simple concatenation are examined, to allow mixed level covering arrays and to exploit “don’t care” positions. The extension to mixed levels is treated in [91], without exploiting the $\star$ positions. To simplify the presentation, we assume a factor with v values always takes on values from $\{1, \dots, v\}$, and hence the corresponding column of the array contains only these symbols, and possibly $\star$.

Theorem 4.1. *Suppose that there exist*

1. *an* MCA$(N; 2, k, (v_1, \dots, v_k))$*, A, with profile* $(d_1, \dots, d_k)$*;*
2. *for each* $1 \le i \le k$*, an* MCA$(M_i; 2, \ell_i, (w_{i1}, \dots, w_{1,\ell_i}))$*, B_i, with profile* $(f_{i1}, \dots, f_{1,\ell_i})$*, and for which* $w_{ij} \le v_i$ *for* $1 \le j \le \ell_i$.

Then for $T = N + \max_{i=1}^{k}(M_i - d_i)$*, there exists an*

$$\text{MCA}\left(T; 2, \sum_{i=1}^{k} \ell_i (w_{11}, \dots, w_{1,\ell_1}, \cdots, w_{k1}, \dots, w_{k,\ell_k})\right).$$

Proof. Form an array C with $N + T$ rows and $\sum_{i=1}^{k} \ell_i$ columns, indexing columns as (i, j) for $1 \leq i \leq k$ and $1 \leq j \leq \ell_i$. On the first N rows, column (i, j) is column i of A. (At this stage, it is possible that $v_i > w_{ij}$; if so, each occurrence of $v_i - w_{ij}$ of the symbols can be changed to $\star$). Now for $i = 1, \ldots, k$, select the d_i rows of A in which the ith column contains a $\star$, and from the last $T - N$ rows choose any $M_i - d_i$ rows, so that M_i rows are selected in total. Then *on these rows in the chosen order*, place the entries of the jth column of B_i in the column (i, j), for $1 \leq j \leq \ell_i$.

Two columns (i_1, j_1) and (i_2, j_2) of the result cover all pairs when $i_1 \neq i_2$; indeed these are covered on the first N rows. When $i_1 = i_2$, restricting to rows arising from B_{i_1}, we find all pairs covered. □

While C is sufficient to cover all pairs, there is much redundant coverage. A simple argument takes advantage of this. Call a row of an MCA *constant* if the only pairs that it covers are of the form (x, x). Suppose that we require every ingredient, without loss of generality, to contain a constant row consisting of all "1" entries. If this is done, then by always selecting the same row of C in which to place the constant row of each B_i, the resulting matrix C has two constant rows of "1" entries. One is redundant and can be removed, leaving no pair uncovered but reducing the size of C. Although this can be done in general, it saves one test only. This simple argument imposes too stringent a condition, since we do not need duplicated rows to have redundant rows. Phrased more generally, then, since the rows of C arising from A, in two columns indexed by (i, j_1) and (i, j_2) cover *all* pairs of the form (x, x), we can delete all constant rows of each B_i prior to applying Theorem 4.1. This can reduce the size of C by many rows in some instances. Moura, Stardom, Stevens, and Williams [91] exploit constant rows effectively to reduce the size of the resulting array.

Unfortunately, in some cases we must sacrifice columns to obtain many constant rows. The standard CA(q^2; 2, q, q) from the finite field has q disjoint constant rows, but its extension to a CA(q^2; 2, $q+1$, q) can have at most one, no matter how the symbols are relabeled. For example, taking two CA(25; 2, 6, 5)s allows us to produce a CA(49; 2, 36, 5) while using instead one CA(25, 2, 5, 5) with five disjoint constant rows allows us to obtain a CA(45; 2, 30, 5). We examine a generalization that enables us to obtain a CA(45; 2, 35, 5), sacrificing one column in the product rather than one column in an ingredient.

For the sake of clarity, we develop a construction for covering arrays first, and then generalize to mixed level covering arrays. An SCA(N; 2, (k_1, k_2), v) is defined to be a CA(N; 2, $k_1 + k_2$, v) in which, for $1 \leq i \leq v$, row $N - v + i$ is a $(k_1 + k_2)$-tuple in which the first k_1 entries are equal to i and the last k_2 symbols are equal to 1. When q is a prime power, an $OA(2, q + 1, q)$ yields an

$\mathrm{SCA}(q^2; 2, (q, 1), q)$.

Now we turn to the main product construction for covering arrays:

Theorem 4.2. *If an* $SCA(N; 2, (k_1, k_2), v)$ *and an* $SCA(M; 2, (\ell_1, \ell_2); v)$ *both exist, then an* $SCA(N + M - v; 2, (k_1\ell_1, k_1\ell_2 + k_2\ell_1), v)$ *also exists.*

Proof. Let A consist of the first $N - v$ rows of the $\mathrm{SCA}(N; 2, (k_1, k_2), v)$, and B consist of the first $M-v$ rows of the $\mathrm{SCA}(M; 2, (\ell_1, \ell_2), v)$. form the product C of A and B as above. Now append v new rows indexed $N + M - 2v + i$, $1 \leq i \leq v$, so that in column $f(k_1+k_2)+g$ the entry is i if $f \leq \ell_1$ and $g \leq k_1$, and is 1 otherwise. Then suppress all columns indexed $f(k_1 + k_2) + g$ when $f > \ell_1$ and $g > k_1$. This removes $k_2\ell_2$ columns from C.

We claim that the result R is an $\mathrm{SCA}(N+M-v; 2, ((k_1\ell_1, k_1\ell_2+k_2\ell_1), v)$. To verify this, consider a column of the array. Restricting attention to the first $N - v$ and last v rows of R yields a column of A *unless* it corresponds to one of the first k_1 columns of A and one of the last ℓ_2 columns of B. By the same token, restricting attention to the last M rows of R yields a column of B *unless* it arises from one of the first ℓ_1 columns of B and one of the last k_2 of A. Hence for two different columns of R, if both arise from columns of A among the first k_1 and columns of B among the first ℓ_1, all pairs are covered since the columns in either A or B must be distinct. If two columns arise from the last k_2 of A, they necessarily both arise from the first ℓ_1 of B (the others were suppressed). Then provided that the columns arise from distinct columns of A, all pairs are covered. In the event that they arise from the same column of A, The first $N-v$ rows cover all constant pairs, i.e. those of the form (x, x) for $1 \leq x \leq v$. But the next $M - v$ rows cover all non-constant pairs. In the same way, if two columns arise from the last ℓ_2 of B, all pairs are covered provided that the columns from which they arise in B are distinct. In the event that they arise from the same column of B, again constant pairs are covered in the rows arising from B and non-constant pairs in the rows arising from A.

It remains to treat cases when one column arises from the first k_1 of A, the other from the last k_2; and in addition one column arises from the first ℓ_1 of B, the other from the last ℓ_2. Since columns arising from the last k_2 columns of A and also the last ℓ_2 columns of B have been suppressed, we may suppose that the first column arises from the first k_1 of A and last ℓ_2 of B; and the second from the last k_2 of A and first ℓ_1 of B. In this case, the first $N-v$ rows cover all pairs except possibly those of the form $(x, 1)$ for $1 \leq x \leq v$. But the next $M-v$ rows cover all pairs except possibly those of the form $(1, x)$ for $1 \leq x \leq v$. The only pair in doubt, namely (1,1), is covered in each of the last v rows. □

The ideas of Theorem 4.1 and Theorem 4.2 can be combined to obtain an improved product construction for mixed level covering arrays. In Theorem 4.2, the repeated use of a single array B is used to ensure that certain pairs are covered in the product C. In order to generalize, our task is to specify the relationship required among the ingredients $\{B_i\}$. We first define an SMCA$(N; 2, (k_1, k_2), (v_1, \dots, v_{k_1+k_2}))$ to be an $N \times (k_1 + k_2)$ array in which the ith column has entries from a v_i-set (which we take to be $\{1, \dots, v_i\}$), and for which selecting any two columns, we find all pairs first define an SMCA$(N; 2, (k_1, k_2), (v_1, \dots, v_{k_1+k_2}))$ to be an $N \times (k_1 + k_2)$ array in which the ith column has entries from a v_i-set (which we take to be $\{1, \dots, v_i\}$), and for which selecting any two columns, we find all pairs covered except possibly

- those of the form (x, x) if both columns are among the first k_1;
- those of the form $(x, 1)$ if the first column is among the first k_1, the second among the latter k_2; and
- (1,1) if both columns are among the latter k_2.

Theorem 4.3. *Suppose that there exist*

1. *an SMCA*$(N; 2, (k_1, k_2), (v_1, \dots, v_{k_1+k_2}))$*, A, with profile* $(d_1, \dots, d_{k_1+k_2})$*;*
2. *for each* $1 \le i \le k_1$*, an SMCA*$(M_i; 2, (\ell_{i1}, \ell_{i2}), (w_{i1}, \dots, w_{1,\ell_{i1}+\ell_{i2}}))$*,* B_i*, for which* $w_{ij} \le v_i$ *for* $1 \le j \le \ell_{i1} + \ell_{i2}$*; and*
3. *for each* $k_1 < i \le k_1 + k_2$*, an SMCA*$(M_i; 2, (\ell_{i1}, 0), (w_{i1}, \dots, w_{1,\ell_{i1}}))$*,* B_i*, for which* $w_{ij} \le v_i$ *for* $1 \le j \le \ell_{i1}$*.*

Write $T = N + \max_{i=1}^{k_1+k_2}(M_i - d_i)$*. Suppose further that*

1. *For* $1 \le i_1 < i_2 \le k_1$*, for each* $\ell_{i_1,1} < j_1 \le \ell_{i_1,1} + \ell_{i_1,2}$*, and each* $\ell_{i_2,1} < j_2 \le \ell_{i_2,1} + \ell_{i_2,2}$*, the* $(T - N) \times 2$ *matrix whose first column contains the entries in the first* $T - N$ *rows of column* j_1 *of* B_{i_1}*, and whose second column contains the entries in the first* $T - N$ *rows of column* j_2 *of* B_{i_2}*, contains every pair of the form* (x, x) *with* $1 \le x \le \min(w_{i_1,j_1}, w_{i_2,j_2})$*; and*
2. *For* $1 \le i_1 \le k_1$ *and* $k_1 < i_2 \le k_1 + k_2$*, for each* $\ell_{i_1,1} < j_1 \le \ell_{i_1,1} + \ell_{i_1,2}$*, and each* $1 \le j_2 \le \ell_{i_2,1}$*, the* $(T - N) \times 2$ *matrix whose first column contains the entries in the first* $T - N$ *rows of column* j_1 *of* B_{i_1}*, and whose second column contains the entries in the first* $T - N$ *rows of column* j_2 *of* B_{i_2}*, contains every pair of the form* $(x, 1)$*, for* $1 \le x \le w_{i_1,j_1}$*.*

Then there exists an

$$SMCA(T; 2, (\sum_{i=1}^{k_1} \ell_{i1}, \sum_{i=k_1+1}^{k_1+k_2} \ell_{i1} + \sum_{i=1}^{k_1} \ell_{i2}),$$

$$(w_{11}, \ldots, w_{1,\ell_{11}}, \cdots, w_{k_1,1}, \ldots, w_{k_1,\ell_{k_1,1}},$$
$$\cdots, w_{k_1+1,1}, \ldots, w_{k_1+1,\ell_{k_1+1,1}}, \cdots, w_{k_1+k_2,1}, \ldots, w_{k_1+k_2,\ell_{k_1+k_2,1}},$$
$$\cdots, w_{1,\ell_{11}+1}, \ldots, w_{1,\ell_{12}}, \cdots, w_{k_1,\ell_{k_1,1}+1}, \ldots, w_{k_1,\ell_{k_1,2}})).$$

Proof. The construction parallels that of Theorem 4.1. When placing the rows of B_i, the first $T - N$ rows of B_i are always placed on the last $T - N$ rows, in the same order, in C. Columns are then permuted so that those indexed by (i, j) with $1 \leq i \leq k_1$ and $1 \leq j \leq \ell_{i1}$ are placed in the first $\sum_{i=1}^{k_1} \ell_{i1}$ positions.

The verification is similar. □

Such an SMCA can be completed to an MCA by the addition of at most m rows, where $m = \max\{w_{ij} : 1 \leq i \leq k_1, 1 \leq j \leq \ell_{i1}\}$. This is done by placing, in the ith additional row, the symbol "1" in columns indexed by (i, j) with $i > k_1$ or $j > \ell_{i,1}$. In the remaining columns, we place the symbol i if $i \leq w_{ij}$, and $\star$ otherwise.

We illustrate the product constructions by establishing some applications leading to new covering array numbers. We focus on the case when every factor has the same number of levels, since tables are available in the literature [89], [120].

Lemma 4.4. *When q is a prime power, and $r \geq 0$ is any integer, there is an SCA($(r + 1)q^2 - rq; 2, (q^{r+1}, (r + 1)q^r), q$) and hence a CA($(r + 1)q^2 - rq; 2, q^{r+1} + (r + 1)q^r, q$).*

Proof. Apply Theorem 4.2 inductively r times using an SCA($q^2; 2, (q, 1), q$). □

Some small examples are

CA(15; 2, 15, 3)	CA(21; 2, 54, 3)	CA(27; 2, 189, 3)
CA(28; 2, 24, 4)	CA(40; 2, 128, 4)	CA(52; 2, 576, 4)
CA(45; 2, 35, 5)	CA(65; 2, 200, 5)	CA(85; 2, 1125, 5)
CA(91; 2, 63, 7)	CA(133; 2, 490, 7)	CA(175; 2, 3673, 7)

These improve upon the construction in [120]. In order to exploit the power of Theorem 4.2 more fully, direct constructions for distinct cover starters can be employed. Start with a (g, ℓ)-distinct cover starter. Treat this as a row, and form the ℓ cyclic shifts of this, obtaining an $\ell \times \ell$ array. Then add a new column with all entries equal to 0. Develop this $\ell \times (\ell + 1)$ array modulo $g - 1$ to form an $\ell(g - 1) \times (\ell + 1)$ array. Adding the g constant rows, we obtain a SCA($\ell(g - 1) + g; 2, (\ell, 1), g$), and hence a CA($(\ell + 1)(g - 1) + 1; 2, \ell + 1, g$). For example, a (5,6)-distinct cover starter yields a CA(29; 2, 7, 5).

However, distinct cover starters have many more effective applications, via Theorem 4.2. For purposes of illustration, consider the case when $g = 5$. Here are some distinct cover starters:

$(5, 5)$	$(\infty, 0, 1, 3, 2)$	$(5, 6)$	$(\infty, 0, 0, 1, 3, 2)$
$(5, 7)$	$(\infty, 0, 0, 0, 1, 3, 2)$	$(5, 8)$	$(\infty, 0, 0, 0, 0, 1, 3, 2)$
$(5, 9)$	$(\infty, 0, 0, 0, 0, 0, 1, 3, 2)$	$(5, 10)$	$(\infty, 0, 0, 0, 0, 0, 0, 1, 3, 2)$

Each $(5, \ell)$-distinct cover starter gives an $\mathrm{SCA}(4\ell + 5; 2, (\ell, 1), 5)$. Apply Theorem 4.2 to obtain $\mathrm{SCA}(4(\ell_1+\ell_2)+5; 2, (\ell_1\ell_2, \ell_1+\ell_2), 5)$ for $5 \leq \ell_1, \ell_2 \leq 10$. Choosing ℓ_1 and ℓ_2 as nearly equal as possible, we obtain the following covering arrays:

$$\begin{array}{ll} \mathrm{CA}(45; 2, 35, 5), & \mathrm{CA}(49; 2, 41, 5), \\ \mathrm{CA}(53; 2, 48, 5), & \mathrm{CA}(57; 2, 55, 5), \\ \mathrm{CA}(61; 2, 63, 5), & \mathrm{CA}(65; 2, 71, 5), \\ \mathrm{CA}(69; 2, 80, 5), & \end{array}$$

and so on. Comparing with the bounds in [120], these produce improvements whenever the number of factors exceeds 30. Further, the construction can be applied recursively. Indeed, with 65 tests we can treat 200 factors with five values each, obtaining a substantial improvement by applying Theorem 4.2 twice rather than once.

4.2. Roux-type Constructions

In [115], a theorem from Roux's Ph.D. dissertation [108] is presented.

Theorem 4.5. *$CAN(3, 2k, 2) \leq CAN(3, k, 2) + CAN(2, k, 2)$.*

Proof. To construct a $\mathrm{CA}(3, 2k, 2)$, we begin by placing two $\mathrm{CA}(N_3, 3, k, 2)$s side by side. We now have a $N_3 \times 2k$ array. If one chooses any three columns whose indices are distinct modulo k, then all triples are covered. The remaining selection consists of a column x from among the first k, its copy among the second k, and a further column y. When the two columns whose indices agree modulo k shate the same value, such a triple is also covered. The remaining triples are handled by appending two $\mathrm{CA}(N_2, 2, k, 2)$s side by side, the second being the bit complement of the first. Therefore if we choose two distinct columns from one half, we choose the bit complement of one of these, thereby handling all remaining triples. This gives us a covering array of size $N_2 + N_3$. □

Chateauneuf et al. [20] prove a generalization, which we repeat here.

Theorem 4.6.

$$CAN(3, 2k, v) \leq CAN(3, k, v) + (v - 1)CAN(2, k, v).$$

Proof. Begin as in Theorem 4.5 by placing two CA(N_3; 3, k, v)s side by side. Let C be a CA(N_2; 2, k, v). Let π be a cyclic permutation of the v symbols. Then for $1 \leq i \leq v - 1$, we append N_2 rows consisting of C and $\pi^i(C)$ placed side-by-side. The verification is as for Theorem 4.5. □

See [7], [64], [65], [88] for generalizations, particularly to cases when $t > 3$. Cohen, Colbourn, and Ling [34] develop a substantial generalization to permit the number of factors to be multiplied by $\ell \geq 2$ rather than two; this is the *k-ary Roux construction*, described in detail next. To carry this out, we require difference covering arrays, introduced earlier.

Theorem 4.7.

$$CAN(3, k\ell, v) \leq CAN(3, k, v) + CAN(3, \ell, v) +$$
$$CAN(2, \ell, v) \times DCAN(2, k, v).$$

Proof. We suppose that the following all exist:

1. a CA(N; 3, ℓ, v) A;
2. a CA(M; 3, k, v) B;
3. a CA(R; 2, ℓ, v) F; and
4. a DCA(Q; 2, k, v) D.

We produce a CA($N + M + QR$; 3, $k\ell$, v) C (see Figure 3). For convenience, we index the $k\ell$ columns of C by ordered pairs from $\{1, \ldots, k\} \times \{1, \ldots, \ell\}$. C is formed by vertically juxtaposing three arrays, C_1 of size $N \times k\ell$, C_2 of size $M \times k\ell$, and C_3 of size $QR \times k\ell$. We describe the construction for each in turn.

C_1 is produced as follows. In row r and column (i, j) of C_1 we place the entry in cell (r, j) of A. Thus C_1 consists of k copies of A placed side by side. This is illustrated in Figure 4.

C_2 is produced as follows. In row r and column (i, j) of C_2 we place the entry in cell (r, i) of B. Thus C_2 consists of ℓ copies of the first column of B, then ℓ copies of the second column, and so on (see Figure 5).

To construct C_3 (see Figure 6), let $D = (d_{ij} : i = 1, \ldots, Q; j = 1 \ldots, k)$ and $F = (f_{rs} : r = 1, \ldots, R; s = 1, \ldots, \ell)$. Choose a cyclic permutation π on

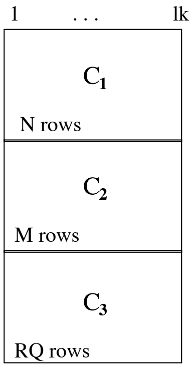

Figure 3: k-ary Roux Construction

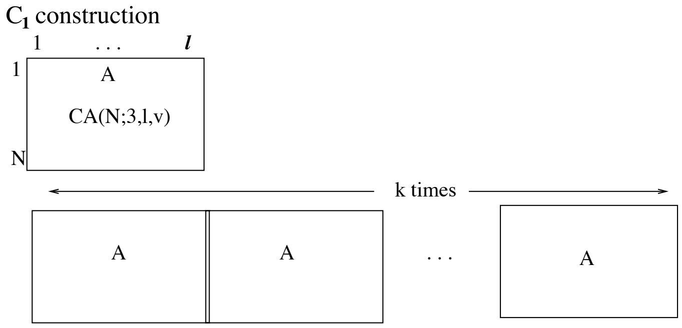

Figure 4: Construction of C_1

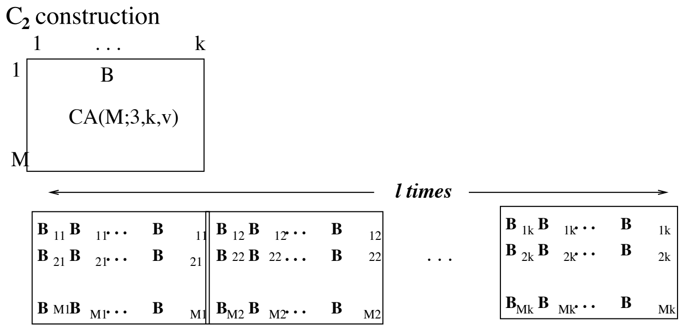

Figure 5: Construction of C_2

the v symbols of the array. Then in row $(i-1)R+r$ and column (j,s) of C_3 place the entry $\pi^{d_{ij}}(f_{rs})$.

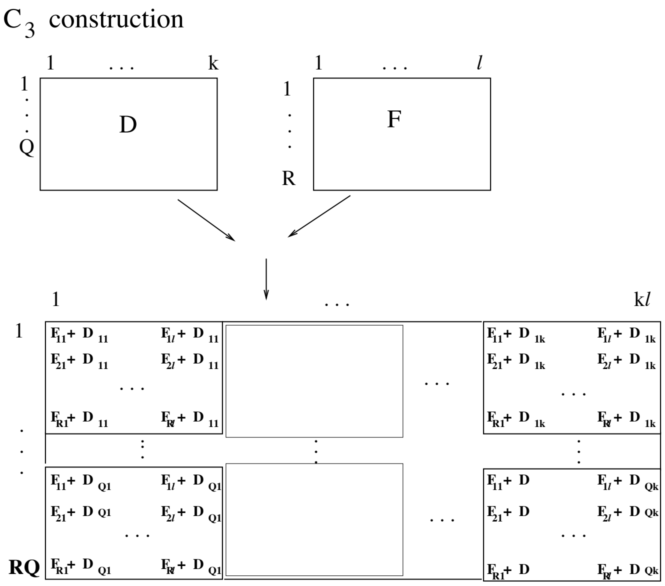

Figure 6: Construction of C_3

We verify that C is indeed a CA$(N + M + QR; 3, k\ell, v)$. The only issue is to ensure that every 3 columns of C cover each of the v^3 3-tuples. Select three columns (i_1, j_1), (i_2, j_2), and (i_3, j_3) of C. If j_1, j_2 and j_3 are all distinct, then these three columns restricted to C_1 arise from three different columns of A, and hence all 3-tuples are covered. Similarly, if i_1, i_2, and i_3 are all distinct, then restricting the three columns to C_2, they arise from three distinct columns of B and hence again all 3-tuples are covered.

So we suppose without loss of generality that $i_1 = i_2 \neq i_3$ and $j_1 \neq j_2 = j_3$. The structure of C_3 consists of a $Q \times k$ block matrix in which each copy is a permuted version of F (under a permutation that is a power of π). That $i_1 = i_2$ indicates that two columns are selected from one column of this block matrix, and that i_3 is different means that the third column is selected from a different column of the block matrix. Now consider a selection $(\sigma_1, \sigma_2, \sigma_3)$ of symbols in the three chosen columns of C (actually, of C_3). Each selection of (σ_1, σ_2) appears in each block of the Q permuted versions of F appearing in the indicated column of the block matrix. Now suppose that $\sigma_3 = \pi^i(\sigma_2)$; since π is a v-cycle, some power of π satisfies this equality. Considering the permuted versions of F appearing in the columns corresponding to i_3, we observe that since D is an array covering all differences modulo v, in at least one row of the block matrix, we find that the block X in column i_3 and the block Y in column

i_2 satisfy $Y = \pi^i(X)$. Hence every choice for σ_3 appears with the specified pair (σ_1, σ_2). □

This can be improved upon: we do not need to cover triples when $\sigma_3 = \sigma_2$ since these are covered in C_1. Nor do we need to cover 3-tuples when $\sigma_1 = \sigma_2$, since these are covered in C_2. So we can eliminate some rows from F since we do not need to cover pairs whose symbols are equal in F. This modification improves further on the bounds.

4.3 Ordered Design Construction

Often other designs can be used as the basis of a "recursive method". We describe an example from [34]. An *ordered design* $OD_\lambda(t, k, v)$ is a $\lambda \cdot \binom{v}{t} \cdot t! \times k$ array with v entries such that

1. each column has v distinct entries, and
2. every t columns contain each row tuple of t distinct entries precisely λ times.

When $\lambda = 1$ we write $OD(t, k, v)$. An $OD(3, q+1, q+1)$ exists when q is a prime power [37]. We use an ordered design as an ingredient for building a CA$(3, q+1, q+1)$ since it already covers all triples with distinct entries, having the minimal number of blocks. This handles many but not all of the triples required. The covering array is completed by covering the remaining triples. We describe a general construction next.

Construction 4.8. CA$N(3, q+1, q+1) \leq q^3 - q + \binom{q+1}{2} \times CAN(3, q+1, 2)$ when q is a prime power.

To create a CA$(3, q+1, q+1)$ begin with a $OD(3, q+1, q+1)$ of size $N_3 = (q+1) \times q \times (q-1)$. This covers all triples of the form (a, b, c) where $a \neq b \neq c \neq a$. To complete the covering array we need to cover all of the triples of the form (a, a, b), (a, b, b), (a, b, a) and (a, a, a). These are exactly the triples covered by a CA$(N_2; 3, q+1, 2)$ on symbol set $\{a, b\}$. Since a and b can be any of $\binom{q+1}{2}$ combinations we append $\binom{q+1}{2}$ CA$(N_2; 3, q+1, 2)$s to the N_3 rows of the ordered design. This gives us a CA$(3, q+1, q+1)$.

Unnecessary coverage of triples occurs. In fact, any triple of the form (a, a, a) is covered at least q times rather than once. We therefore relabel entries in the CA$(N_2; 3, q+1, 2)$s to form a constant row; deleting these reduces the number of rows required by $\binom{q+1}{2}$. We can save even more:

Construction 4.9. CA$N(3, q+1, q+1) \leq q^3 - q + \binom{q+1}{2} \times CAN(3, q+1, 2) - (q^2 - 1)$ when q is a prime power and there are two disjoint rows in the CA$(3, q+1, 2)$.

In Construction 4.8 we exploit overlap in coverage of triples that occurs if each of the CA(N_2; 3, $q+1$, 2)s has two disjoint rows. In this case we remap the two disjoint rows, without loss of generality, to the form $(a, a, ..., a)$ and $(b, b, ..., b)$. We remove the $2 \times \binom{q+1}{2} = q^2 + q$ rows and add back in $q + 1$ rows of the form $(a, a, ..., a)$.

We give an example using CA(3, 6, 6). The ordered design has 120 rows. There are 15 combinations of two symbols. In Construction 4.8, we create a CA(3, 6, 2) with 12 rows. We therefore add back in 180 rows. This gives us a CA(3, 6, 6) of size 300. This is smaller than the bound reported by a construction in [20], and matches that found by annealing in [32]. Removing 15 constant rows lowers this bound to 285. For Construction 4.9, we find a CA(12; 3, 6, 2) having two disjoint rows (see Table 1). Therefore we remove 30 rows of the type $(a, a, ..., a)$ for a total of 270 rows. We add back in six rows, one for each symbol, to achieve a covering array of size 276. This improves on both reported bounds above.

We generalize further. A (2,1)-*covering array*, denoted by

$$TOCA(N; 3, k, v; \sigma)$$

is an $N \times k$ array containing σ or more disjoint constant rows, in which every $N \times 3$ subarray contains every 3-tuple of the form (a, a, b), (a, b, a), and (b, a, a) with $a \neq b$, and contains every 3-tuple of the form (a, a, a). $TOCAN(3, k, v; \sigma)$ denotes the minimum number N of rows in such an array.

A set $\mathcal{B}$ of subsets of $\{1, \dots, k\}$ is a *linear space* of order k if every 2-subset $\{i, j\} \subseteq \{1, \dots, k\}$ appears in exactly one $B \in \mathcal{B}$.

Construction 4.10. Let q be a prime power. Let $\mathcal{B} = \{B_1, \dots, B_b\}$ be a linear space on $K = \{1, \dots, k\}$. Let $\emptyset \subseteq L \subseteq K$. Suppose that for each $B_i \in \mathcal{B}$ there exists a $TOCA(N_i; 3, q+1, |B_i|; |B_i \cap L|)$. Then there exists a CA($q^3 - q + |L| + \sum_{i=1}^{b}(N_i - |B_i \cap L|)$; 3, $q+1$, $q+1$).

We start with an $OD(3, q+1, q+1)$ and for each $B_i \in B$, we construct the TOCA on the symbols of B_i with the constant rows (to be removed) on the symbols of $B_i \cap L$. Then $|L|$ constant rows complete the covering array.

4.4. Construction Using Generalized Hadamard Matrices

There is the opportunity to develop "constructions" when some of the "ingredients" are not known at all. We illustrate this next, again following the presentation in [34]. The basic plan is to simply construct a large portion of a covering array to use as a seed. Consider an $OA_\lambda(2, k, v)$. Each 2-tuple is covered exactly λ times. Some 3-tuples are also covered. Indeed, among the v 3-tuples containing a specified 2-tuple, at least one and at most $\min(v, \lambda)$

are covered. If λ of the v are covered for every 2-tuple, the orthogonal array is *supersimple*. Little is known about supersimple orthogonal arrays except when k is small. However our concern is only that "relatively many" triples are covered using "relatively few" rows. This is intentionally vague, since our intent is only to use the rows of the orthogonal array as a seed for a strength three covering array. A natural family of orthogonal arrays to consider arise from generalized Hadamard matrices (see [36]). We have no assurance that the resulting orthogonal arrays are supersimple, but instead choose generalized Hadamard matrices since they provide a means to cover many of the triples to be covered by the covering array. Although orthogonal arrays in general may be useful in constructions here, those from generalized Hadamard matrices appear frequently to cover either only one, or all v, of the triples containing a specified pair; this regularity appears to be beneficial. The most important remark here is that, given such a generalized Hadamard matrix, it is not at all clear what "ingredients" are needed to complete it to a covering array in general, despite the fact that in any specific case we can easily enumerate the triples left uncovered.

0	0	0	0	0	0
1	1	1	1	1	1
0	1	1	0	1	0
0	0	1	1	0	1
0	0	0	0	1	1
0	1	0	1	1	1
1	1	0	1	0	0
1	1	0	0	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
0	1	1	1	0	0

Table 1: $TOCA(12; 3, 6, 2; 2)$

4.5 Using Perfect Hash Families

A $(k, n; \lambda)$-*difference matrix* is a $n \times k\lambda$ matrix D, with entries in $\mathbb{Z}_k$, in which the vector difference of any pair of rows contains every member of $\mathbb{Z}_k$ exactly λ times. For example, if $\gcd((n-1)!, k) = 1$, then the $n \times k$ matrix D defined by $D[i, j] = ij \bmod k$ is a $(k, n; 1)$-difference matrix.

Lemma 4.11. (Atici et. al. [3]) *Suppose* $T_1, T_2, \ldots, T_r \subseteq \mathbb{Z}_k$*, where* $T_i \neq \emptyset$*,*

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Table 2: $TOCA(13; 3, 10, 2; 2)$

$1 \leq i \leq r$ and

$$\sum_{j=1}^{r} |T_j| = t.$$

Suppose also that D is a $(k, \binom{t}{2} + 1; 1)$-difference matrix. Then there is an integer x such that the r sets

$$T'_j = \{t + D[x, j] : t \in T_j\}$$

$1 \leq j \leq r$ are all disjoint.

In [19], Lemma 4.11 is used to establish a slight generalization of Theorem 4.1 in [3]. Let Ω be a v-element set and let $\mathcal{P}$ be a first-order predicate concerning a collection $\mathcal{C}$ of t-tuples with entries in Ω that is invariant under coordinate permutation. For example the predicate $\mathcal{P}$ could be one of

1. The t-tuple $[a, a, ..., a]$ is in $\mathcal{C}$, where $a \in \Omega$ is a fixed symbol.
2. The set of all t-tuples with exactly two distinct entries are in $\mathcal{C}$.
3. There is a t-tuple in $\mathcal{C}$ with distinct entries.
4. All of the t-tuples with entries in Ω occur in $\mathcal{C}$.

A $k \times N$ array A with entries in Ω is called a $(N; t, k, v)$-$\mathcal{P}$ array if each collection $\mathcal{C}$ of t-tuples found among the columns of any $t \times N$ subarray satisfies the predicate $\mathcal{P}$. For example if $\mathcal{P}$ is predicate 3 then the array A is a perfect hash family [3? 6, 15, 129] or a t-separating family [105]. If $\mathcal{P}$ is predicate 4, then A is the transpose of a covering array.

Theorem 4.12. *If gcd* $(\binom{t}{2}!, k) = 1$ *and a* $(N; t, k, v)$-$\mathscr{P}$ *array exists, then there is a* $((\binom{t}{2} + 1)N, t, k^2, v)$-$\mathscr{P}$ *array.*

Proof. (This proof from [19] parallels one in [3].) Let A be a $(N; t, k, v)$-$\mathscr{P}$ array with rows labeled by $\mathbb{Z}_k$ and columns by $\{1, 2, \dots, N\}$. Let D be a $(k, \binom{t}{2} + 1; 1)$-difference matrix and for $x \in \mathbb{Z}_k$, let A_x denote the array

$$A_x[i, j] = A[i + x, j], \qquad i \in \mathbb{Z}_k, \quad j = 1, 2, ..., N$$

Consider the array B defined by

$$B = \begin{array}{|c|c|c|c|}
\hline
B_{0,0} & B_{0,1} & \cdots & B_{0,\binom{t}{2}} \\
\hline
B_{1,0} & B_{1,1} & \cdots & B_{1,\binom{t}{2}} \\
\hline
\vdots & \vdots & \ddots & \vdots \\
\hline
B_{k-1,0} & B_{k-1,1} & \cdots & B_{k-1,\binom{t}{2}} \\
\hline
\end{array}$$

where $B_{i,j} = A_{D[i,j]}$ and the rows of B are indexed by $\mathbb{Z}_k \times \mathbb{Z}_k$ so that the rows of

$$\begin{array}{|c|c|c|c|}
\hline
B_{i,0} & B_{i,1} & \cdots & B_{i,\binom{t}{2}} \\
\hline
\end{array}$$

are indexed by $\{i\} \times \mathbb{Z}_k$. To see that B is the desired $((\binom{t}{2} + 1)N; t, k, v)$-$\mathscr{P}$ array consider a set of t rows $\mathscr{R}$ of B. Set $T_i = \{x \in \mathbb{Z}_k : (i, x) \in \mathscr{R}\}$. Let $T_{i_1}, T_{i_2}, \dots, T_{i_r}$ denote the T_i's that are nonempty. Then by Lemma 4.11 there is an integer x such that the r sets $T'_{i_j} = \{t + D[x, j] : t \in T_{i_j}\}$ for $1 \le j \le r$ are all disjoint. Thus they comprise t distinct rows of the $k \times (\binom{t}{2} + 1)N$ subarray

$$\begin{array}{|c|c|c|c|}
\hline
B_{x,0} & B_{x,1} & \cdots & B_{x,\binom{t}{2}} \\
\hline
\end{array}$$

Each component $B_{x,j}$ is a copy of A with the rows permuted. Thus the t rows satisfy predicate $\mathscr{P}$. Therefore B is a $((\binom{t}{2} + 1)N, t, k^2, v)$-$\mathscr{P}$ array. □

This theorem can be iterated:

Theorem 4.13. *If gcd*$(\binom{t}{2}!, k) = 1$ *and a* $(N; t, k, v)$-$\mathscr{P}$ *array exists, then there is a* $(N(\binom{t}{2} + 1)^j, t, k^{2^j}, v)$-$\mathscr{P}$ *array.*

Applying this to covering arrays we obtain:

Theorem 4.14. *Let $v > 2$ be a positive integer, and let $q \geq v - 1$ be a prime power. Then, for all j, there is a CA$((2v-1)(q^3-q)+v)4^j3(2v-1)^{2^j}v$ if $v \equiv 0, 1$ mod 3, and a CA$((2v-1)(q^3-q)+v)4^j3(2v-3)^{2^j}v$ if $v \equiv 2$ mod 3.*

Proof. There is, by Theorem 3.2, a CA$(2v-1)(q^3-q)+v32vv$. Deleting columns leaves a covering array. If $v \equiv 0, 1$ mod 3, then $\gcd(6, 2v-1) = 1$ and if $v \equiv 2$ mod 3, then $\gcd(6, 2v-3) = 1$. Apply Theorem 4.13 (to the transpose of the covering array) with $t = 3$ and $k = 2v - 1$ when $v \equiv 0, 1$ mod 3 or $k = 2v - 3$ when $v \equiv 2$ mod 3. □

With the parameters in Theorem 4.14, Chateauneuf et al. [19] establish that

$$\mathrm{CAN}(3, k, v) \leq \begin{cases} \dfrac{(2v-1)(q^3-q)+v}{(\log(2v-1))^2}(\log k)^2 \text{ if } v \equiv 0, 1 \text{ mod } 3 \\ \dfrac{(2v-1)(q^3-q)+v}{(\log(2v-3))^2}(\log k)^2 \text{ if } v \equiv 2 \text{ mod3} \end{cases}$$

where $q \geq v - 1$ is a prime power. In particular when $v = 3$,

$$\mathrm{CAN}(3, k, 3) \leq \frac{33}{(\log 5)^2}(\log k)^2 \approx 6.1209(\log k)^2,$$

and when $v = 4$,

$$\mathrm{CAN}(3, k, 4) \leq \frac{172}{(\log 7)^2}(\log k)^2 \approx 21.8240(\log k)^2.$$

For $v = 4$ this can be improved this using the CA(88;3,8,4) from Example 3.4:

$$\mathrm{CAN}(3, k, 4) \leq \frac{88}{(\log 7)^2}(\log k)^2 \approx 11.1658(\log k)^2$$

The bound can be improved asymptotically to be on the order of $v^2 \log k$ using probabilistic techniques [60]. However, the technique developed in [19] is entirely constructive.

5. Algorithms for Covering Arrays.

Few exact existence results are known. Östergård [101] showed that $N \leq 11$ for $v = 3, k = 5$, while Sloane [115] mentions that Applegate showed that $N = 11$ using an integer programming method. Backtracking and exact integer programming techniques appear to be infeasible except when the parameters are quite small; see [135] for a formulation. For this reason, we concentrate on heuristic techniques.

5.1. Greedy Algorithms

Numerous greedy methods have been proposed to construct covering arrays. We focus first on methods that generate covering arrays "one row at a time", following the presentation in [35], [128].

5.1.1 The Logarithmic Guarantee

Suppose that we are to test a system with k factors $f_1, \dots, f_k$. The factor f_i is permitted to take on any of v_i *levels* or *values*, which we denote by $\{\sigma_{i,j} : j = 1, \dots, v_i\}$. The objective is to produce an MCA(N; 2, k, $(v_1 ... v_k)$).

The AETG system attempts to make a 'small' covering array using a greedy strategy. It selects a single test at a time, repeating this until all pairs are covered in at least one of the selected tests. Since the objective is to minimize the number of tests, AETG concentrates on the selection of each test to maximize the number of previously uncovered pairs that are covered by this test. The paper makes two main contributions [23]:

1. It shows a *logarithmic* bound on the number of tests needed as a function of k.
2. It describes a (greedy) heuristic for the selection of tests.

The first relies on a conceptually simple construction method for covering arrays. Having selected some (partial) collection of tests, we record the pairs $\mathcal{P}$ yet to be covered. Among the $\prod_{i=1}^{k} v_i$ possible tests, there can be substantial variation in the number of pairs in $\mathcal{P}$ that the test covers. We select a test that covers that largest number of pairs in $\mathcal{P}$, add it to the collection of tests, and repeat this step until $\mathcal{P} = \emptyset$; at this point, we have the covering array. This is a greedy method, and by no means guarantees the minimum size of the possible test suite constructed. However, it does ensure that at each stage, at least $|\mathcal{P}|/L$ new pairs are covered where L is the product of the two largest of the sizes $\{v_i\}$. This in turn ensures that the size of the test suite constructed is bounded by a logarithmic function of the number k of factors (see [23] for details).

However, the authors do not propose an algorithm for finding the test that covers a maximum number of uncovered pairs, and instead adopt a greedy heuristic to produce each new test in turn. We review their method here. Each test is selected from a pool of M candidate tests, where M is a constant chosen in advance. To generate each candidate, first select a factor f_i and a value σ_i for this factor so that the choice of σ_i for f_i appears in the maximum number of uncovered pairs. Set $\pi(1) = i$. Then choose a random permutation of the indices of the remaining factors to form a permutation $\pi : \{1, \ldots, k\} \rightarrow \{1, \ldots, k\}$. Now assume that values $\tau_1, \ldots, \tau_i$ have been selected for factors $f_{\pi(1)}, \ldots, f_{\pi(i)}$. Select a value τ_{i+1} for factor $f_{\pi(i+1)}$ by selecting that value which yields the maximum number of new pairs with the selected values for the i factors already fixed.

Repeating this process M times exploits the randomness in the factor ordering, and yields different tests from which to select. Naturally, one selects the best in terms of newly covered pairs, and adds it to the test suite.

While the authors note that this appears to exhibit a logarithmic performance in practice, their earlier guarantee does not apply because the test selection does not ensure that a selected test covers the maximum possible number of new pairs. In view of the next result, this is not surprising.

Given a collection $\mathcal{P}$ of uncovered pairs, and a specified number p, it is NP-complete to determine if there exists a test covering at least p pairs. See [111]. Membership in NP is straightforward, since one can nondeterministically select a test, and compute in polynomial time (deterministically) the number of pairs of $\mathcal{P}$ covered. To establish NP-hardness, we give a reduction from MAX 2SAT (“Given a logical formula in 2-conjunctive normal form, and an integer p, is there a truth assignment to the variables that makes at least p clauses true?”). This problem is NP-complete (see, for example, [59]). Let $\mathcal{F}$ be a formula in 2-conjunctive normal form with k logical variables. We form k factors, each with two levels, ‘true’ and ‘false’. For each clause of $\mathcal{F}$, treat as a covered pair the truth assignment to the two variables which makes the clause false. Then $\mathcal{P}$ contains all pairs not covered in this way. Now we determine whether there is a test which covers at least p pairs of $\mathcal{P}$. A test corresponds directly to a truth assignment for $\mathcal{F}$, and an uncovered pair to a clause which evaluates to true. Thus the existence of such a test is equivalent to a solution to MAX 2SAT.

In plain terms, what this says is that an efficient technique to select a test covering the maximum number of uncovered pairs is unlikely to exist. However, this leaves an unsatisfactory situation. The current proof of logarithmic growth hinges on selecting such a test! Fortunately, the situation is not as bad as it first appears. Indeed a more careful reading of the proof of logarithmic growth establishes that one does not need to find a test which covers the maximum number of uncovered pairs. All one needs is to find a test that covers the *average* number of uncovered pairs.

It appears that the test selection in AETG does not ensure this, although for practical purposes unless M is quite small, the likelihood that at least one of the M candidates has this property is high. Nevertheless, it is reasonable to ask for a test selection technique that *guarantees* to cover at least the average number. We pursue this next.

The lack of a guarantee results primarily from the greedy nature of the test selection. In particular, when selecting a value for the ith factor, only its interaction with the first $i-1$ factors is considered. This can (and does) result in a selection which make selections for the later factors less desirable. Indeed if there are 100 factors, and we are selecting a value for the fifth, for example, its interaction with the later 95 factors is arguably more important than its interaction with the first four. We use this intuition to suggest an alternate approach.

We consider the construction of a test suite with k factors. The number of levels for factor i is denoted by v_i. For factors i and j, we define the *local density* to be $\delta_{i,j} = \frac{r_{i,j}}{v_i v_j}$ where $r_{i,j}$ is the number of uncovered pairs involving a value of factor i and a value of factor j. In essence, $\delta_{i,j}$ indicates the fraction of pairs of assignments to these factors which remain to be tested. We define the *global density* to be $\delta = \sum_{1 \leq i < j \leq k} \delta_{i,j}$. At each stage, we endeavour to find a test covering at least δ uncovered pairs.

To select such a test, we repeatedly fix a value for each factor, and update the local and global density values. At each stage, some factors are *fixed* to a specific value, while others remain *free* to take on any of the possible values. When all factors are fixed, we have succeeded in choosing the test. Otherwise, select a free factor f_s. We have $\delta = \sum_{1 \leq i < j \leq k} \delta_{i,j}$, which we separate into two terms:

$$\delta = \sum_{\substack{1 \leq i < j \leq k \\ i,j \neq s}} \delta_{i,j} + \sum_{\substack{1 \leq i \leq k \\ i \neq s}} \delta_{i,s}.$$

Whatever level is selected for factor f_s, the first summation is not affected, so we focus on the second.

Write $\rho_{i,s,\sigma}$ for $\frac{1}{v_i}$ times the number of uncovered pairs involving some level of factor f_i, and level σ of factor f_s. Then rewrite the second summation as

$$\sum_{\substack{1 \leq i \leq k \\ i \neq s}} \delta_{i,s} = \frac{1}{v_s} \sum_{\sigma=1}^{v_s} \sum_{\substack{1 \leq i \leq k \\ i \neq s}} \rho_{i,s,\sigma}.$$

We choose σ to maximize $\sum_{\substack{1 \leq i \leq k \\ i \neq s}} \rho_{i,s,\sigma}$. It follows that $\sum_{\substack{1 \leq i \leq k \\ i \neq s}} \rho_{i,s,\sigma} \geq \sum_{\substack{1 \leq i \leq k \\ i \neq s}} \delta_{i,s}$. We then fix factor f_s to have value σ, set $v_s = 1$, and update the local densities setting $\delta_{i,s}$ to be $\rho_{i,s,\sigma}$. In the process, the density has not been decreased (despite some possible – indeed necessary – decreases in some local densities).

We iterate this process until every factor is fixed. The factors could be fixed in *any order at all*, and the final test has density at least δ. Of course it is possible to be greedy in the order in which factors are fixed. If we apply this method to the case where each factor has the same number of levels, the density is the average number of uncovered pairs in a test that could be selected, and we guarantee to select a test with at least this number of uncovered pairs.

5.1.2 A framework for one-row-at-a-time greedy methods

Following [128], we present pseudocode for greedy methods that build one row at a time.

```
set $MinArray$ to $\infty$
repeat $\boxed{N}$ times
  start with an empty covering array $C$
  while there are uncovered $t$-tuples in $C$
    repeat $\boxed{M}$ times
      start with an empty test (row) $R$
      set $Best = 0$
      while free factors remain
        rank all free factors according to a
            $\boxed{\text{factor selection criterion}}$
        among factors tied for best, select a subset $T$
            using a $\boxed{\text{first factor tie-break}}$
        among factors in $T$, select a single factor $f$
            using a $\boxed{\text{second factor tie-break}}$
        all possible values for $f$ in $R$ using
            a $\boxed{\text{level selection criterion}}$
        among all best values for $f$, select a subset $V$
            using a $\boxed{\text{first level tie-break}}$
        among values in $V$, select value $v$
            using a $\boxed{\text{second level tie-break}}$
        fix factor $f$ to value $v$ in test $R$
      end while
      If $R$ covers $\sigma > Best$ $t$-tuples uncovered
          in $C$, set $Best = \sigma$, $B = R$
    end repeat
    add row $B$ to $C$
  end while
```

```
if $C$ has $Size < MinArray$ rows, set $MinArray = Size$
    and $BestArray = C$
end repeat
report $BestArray$
```

To instantiate this method, a number of decisions must be made, shown in boxes in the skeleton above. Greedy algorithms for the construction of covering arrays have four major decision points to define. These include:

1. the number of covering array repetitions;
2. the number of test case candidates to generate;
3. factor ordering heuristics (including tie-breaking); and
4. level section heuristics (including tie-breaking).

We call these *layers* in the instantiation of the method. The specification of the four layers dominates the accuracy and efficiency of such algorithms.

Layer one - covering array repetitions At the top layer, covering arrays may be generated numerous times and the smallest size covering array is kept. This is only appropriate for methods that have elements of randomness. Larger numbers of repetitions require lengthier execution times, and consistency from one run to the next cannot be ensured when random selections are made.

Layer two - multiple candidates During test generation, an algorithm may generate multiple candidate tests and then evaluate them to add the best one to the final covering array. Multiple candidates for a row allow the exploration of different combinations and can improve the likelihood of obtaining a more accurate result.

Layer three- factor ordering The order in which factors in a row are fixed has the largest impact in the framework. Poor selection early in the generation of a row percolates to the remaining factors in the row that have yet to be fixed. Factors may be ranked

1. by the number of levels associated with a factor;
2. by number of uncovered pairs involving this factor and the fixed factors;
3. by the expected number of pairs covered including both fixed and free factors (*density*); or
4. randomly.

The TCG [127] algorithm exemplifies factor ordering based on the number of levels associated with a factor. The factors are ordered on decreasing cardinalities. DDA [35] exemplifies ordering based on a heuristic that calculates the expected number of pairs that will be covered among fixed and free factors. AETG [23], [24] utilizes random factor ordering. No published greedy method orders factors based only on the number of uncovered pairs; however, we include this factor ordering method for thoroughness.

Three of the factor ordering methods can suffer from ties. To break ties, one of the following tie breaking schemes may be used:

1. Take lexicographically first;
2. Take one at random;
3. Take one with the most uncovered pairs remaining.

Not all tie-breaking rules given succeed in completely resolving a tie; hence a second round of tie-breaking can be needed. Tie-breaking is not usually needed beyond two levels. However, if a tie still occurs at this point, it is broken in this implementation by using specified order selection of *take first* to ensure that only a single candidate remains.

Layer four - level selection Level selection is the fourth layer in the framework which attempts to locally maximize the number of pairs that will be covered, or are expected to be covered for the row. Levels can be chosen

1. by number of uncovered pairs involving this value of the current factor and the fixed factors;
2. by the expected number of pairs involving this value of the current factor covered including both fixed and free factors (*density*); or
3. randomly.

AETG [23], [24], [25], [26] and TCG [127] select a level based on the number of uncovered pairs to be covered. DDA [35] selects a level based on a heuristic that includes a calculation of the pairs that will be covered, and the likelihood of pairs that can be covered with free factors.

5.2. Greedy Variations

Cohen, Litsyn, and Zémor [28] suggest a method that, rather than building the array one row at a time, instead chooses a set of candidate columns, and iteratively removes columns so that what remains is a covering array. To make this procedure effective, the initial set of candidate columns must be restricted in some manner to avoid exponential growth. For the initial selection of columns, the authors suggest using the codewords of an error-correcting code. This is extended in Cheng, Dumitrescu, and Schroeder [22], [53].

In terms of practical application, the variant most studied is a scheme that adjoins both rows and columns in a greedy fashion, called *In-Parameter-Order (IPO)* [123], [140]. IPO generates a mixed covering array MCA$(2, k, (v_1, \ldots, v_k))$ by first generating an MCA$(2, k-1, (v_1, \ldots, v_{k-1}))$. Then it adjoins an additional column, greedily choosing the value in each row of the new column to maximize the number of uncovered pairs (this is *horizontal growth*). It can happen that horizontal growth fails to cover all of the pairs involving values in the new column. A second expansion, *vertical growth*, adds rows to cover all remaining pairs. Vertical growth to add the minimum number of rows is straightforward, but the method for horizontal growth is heuristic. Moreover, the structure of pairs remaining to be covered after horizontal growth determines the number of rows that vertical growth must add.

5.3 Hill Climbing

Hill climbing, simulated annealing, and tabu search are variants of the state space search technique for solving combinatorial optimization problems. With a general optimization problem the hope is that the found solution is close to an optimal one. With many design problems we *know* (from the cost) when we have reached an optimal solution. On the other hand, approximations in these cases are of little value.

An optimization problem can be specified as a set Σ of feasible solutions (or states) together with a cost $c(S)$ associated with each $S \in \Sigma$. An optimal solution corresponds to a feasible solution with overall (i.e. global) minimum cost. We define, for each $S \in \Sigma$, a set T_S of transformations (or transitions), each of which can be used to change S into another feasible solution S'. The set of solutions that can be reached from S by applying a transformation from T_S is called the neighbourhood $N(S)$ of S.

We start by randomly choosing an initial feasible solution and then generate a randomly chosen transformation of the current feasible solution S. If the transformation results in a feasible solution S' of equal or lower cost, then S' is accepted as the new current feasible solution. If S' is of higher cost, we reject this solution and check another randomly chosen neighbour of the current feasible solution. This allows us to randomly *walk* around Σ, without reducing the goodness of our current solution. Hill climbing has the potential to get stuck in a local minimum (or *freeze*), so stopping heuristics are required. To increase the chance of forming a good solution we repeat the random walk (or *trial*) a number of times, each time beginning with a random initial feasible solution.

In the hill climbing algorithm the current feasible solution is an approximation S to a covering array in which certain t-subsets are not covered. The cost function is based on the number of t-subsets that are not covered, so that a covering array itself will have a cost of zero. A potential transformation is made by selecting one of the k sets belonging to S and then replacing a random point in this k-set by a random point not in the k-set. The number of blocks remains constant throughout the hill climbing trial.

We set loose upper and lower bounds on the size of an optimal array and then use a binary search process to find the smallest sized covering array in this interval. An alternative method is to start with the size of a known test suite and search for a solution. This of course uses less computational resources, but the required test suite size must be known ahead of time. Ideally in a real system this is the method to use.

5.4. Tabu Search

Tabu search generalizes hill-climbing by allowing the current feasible solution to be replaced by a poorer one. However it restricts the changes using *tabu* and *aspiration* lists. The first prohibits moves that are deemed unattractive, such as reversing a move just made. The second assigns higher weight to moves that achieve a desired property (aspiration). Nurmela [99] describes numerous successful searches using tabu search.

5.5. Simulated Annealing

Simulated annealing has been used by Nurmela and Östergärd [100], to construct covering designs which have a structure very similar to covering arrays. Stevens [117], Stardom [116], and Cohen and her colleagues [30], [31], [32], [33], [34] have applied simulated annealing to the search for covering arrays.

Simulated annealing uses the same approach as hill climbing but allows the algorithm, with a controlled probability, to make choices that reduce the quality of the current solution. The idea is to avoid getting stuck in a bad configuration while continuing to make progress. If the transformation results in a feasible solution S' of higher cost, then S' is accepted with probability $e^{-(c(S')-c(S))/K_B T}$, where T is the controlling temperature of the simulation and K_B is a constant. The temperature is lowered in small steps with the system being allowed to approach "equilibrium" at each temperature through a sequence of transitions (or Markov chain) at this temperature. Usually this is done by setting $T := \alpha T$, where α (the *control decrement*) is a real number slightly less than 1. After an appropriate stopping condition is met, the current feasible solution is taken as an approximation to the solution of the problem at hand. Again, we improve our chances of obtaining a good solution by running a number of trials.

5.6. Great Deluge Algorithm

One further heuristic search technique is the *great flood* or *great deluge* algorithm [50], [51], and a variant thereof called *threshold accepting*. These follow a strategy similar to simulated annealing but often display more rapid convergence. Instead of using probability to decide on a move when the cost is higher, a worse feasible solution is chosen if the cost is less than the current threshold. This threshold value is sometimes referred to as the *water level* which, in a profit maximizing problem, would be rising rather than falling (as is happening in this case). As the algorithm progresses, the threshold is reduced moving it closer to zero. These seem not to have been explored for the construction of covering arrays.

5.7. Genetic Algorithms

In the world of heuristic search, genetic algorithms have proven to be quite competitive. The basic idea is to maintain a population of putative solutions, and to evolve the population from one generation to the next by two operators. *Mutation* makes small local changes in putative solutions, while *crossover* combines part of one solution and part of another. Survival into the new generation is determined by the relative *fitness* of each new putative solution. Roughly speaking, this determines how "close" it is to the desired covering array.

Until this time the application to covering arrays has been very little explored. Stardom [116] reports initial work that is not encouraging. Despite this, the selection of an appropriate representation aalong with mutation and crossover operators may well make genetic algorithms competitive in this arena. More research is needed.

6. Applications.

6.1. Interaction Software Testing

Software testing comprises a significant proportion of the cost of any software project, yet failures still occur. Indeed a 2002 report from the National Institute of Standards and Technology (NIST) reports the alarming conclusion that inadequate software testing incurs a cost of $59.5 billion annually, and attributes a substantial amount to failures of software embedded in hardware [97]. Carroll [13], [14] describes the cost of inadequate software testing as well. In recent years the DOD has begun initiatives to use more commercial off the shelf software (COTS) [98]. Testing methods have striven to develop adequate models [43], [55], [63], [102], [110], [126], develop appropriate coverage measures [12], [136], relate coverage to reliability and fault detection [42], [68], [75], [84], [137], and develop automatic test generators [10], [44], [72], [113]. Furthermore, applications in testing communications systems and storage systems have also been treated [21], [103], [130].

Each component may be tested and certified individually; however, system faults are likely to occur from unexpected interactions [134]. If even a small percentage of these faults can be prevented through integration testing this may have a significant financial impact. When working with components, we would ideally like to test all combinations of components prior to system release. However the size of a test suite required to test all possible combinations can be prohibitive in even a moderately sized project.

As a compromise, combinatorial design techniques may be applied to instead guarantee a specific fixed level of interaction testing, i.e. pairwise or t-wise testing. Although this does not provide us with complete coverage, it has been shown to be successful in finding a large number of faults when used in interface testing [45], [54], [67], [70], [81], [109].

At the current time there are two distinct areas of active research on combinatorial designs for software interaction testing. As we have seen, combinatorialists are focusing on building smaller designs of higher interaction strength. The software testing community is focusing on greedy search algorithms to build these in a more flexible environment, one that more closely matches real testing needs [23], [24], [26], [27], [45], [54], [123], [127], [133], [140]. Software testers cannot be expected to master each technique for the generation of software test suites, or even to determine which of the many available methods is best for their application. A multidimensional approach for this problem is found in [24], but the solution is limited in scope. What is needed is a usable tool that puts these techniques at their disposal by presenting a single, unified means to construct test suites employing the best of each technique individually and collectively.

We give an example. The following Internet based software system allows customers to use a variety of browsers. They may also be using different operating systems, network connection types and printer configurations. In order to completely test this system we want to test our software on all of the possible supported configurations. If we have the system shown in Table 3, we would want to test combinations, such as (Netscape,Windows,LAN,Local) and (Netscape,Windows,LAN,Networked). In order to test all possible interactions we would need 3^4 or 81 configurations. This is perhaps reasonable, but if we we extend this to 10 components with 4 possible settings each, we would need $4^{10} = 1,048,576$ test configurations. Instead we can guarantee that we can test all pairs of interactions or all t-way interactions [9], [23], [86], [54], [119], [133], [140]. In Table 4 we can cover all pairs of interactions for the example in Table 3 using only 9 different tests. And in the example of 10 components each with 4 possible settings we can cover all pairs of interactions using at most 29 tests.

	Component			
	Web Browser	**Operating System**	**Connection Type**	**Printer Config**

Config:	Netscape(0)
	IE(1)
	Other(2)
		Window(0)
		Macintosh(1)
		Linux(2)
			LAN(0)
			PPP(1)
			ISDN(2)
				Local(0)
				Networked(1)
				Screen(2)

Table 3: Four components, each with 3 configurations

Test Case	Browser	OS	Connection	Printer
1	NetScape	Windows	LAN	Local
2	NetScape	Linux	ISDN	Networked
3	NetScape	Macintosh	PPP	Screen
4	IE	Windows	ISDN	Screen
5	IE	Macintosh	LAN	Networked
6	IE	Linux	PPP	Local
6	IE	Linux	PPP	Local
7	Other	Windows	PPP	Networked
8	Other	Linux	LAN	Screen
9	Other	Macintosh	ISDN	Local

Table 4: Test Suite to Cover all Pairs from Table 3

The concept of pair-wise coverage has been used across many disciplines including medicine, agriculture and manufacturing [66]. It has entered the software testing community, appearing in practitioner's guidebooks [73], [92], and provided in simple spreadsheet formats [40], [41]. The use of covering arrays in software testing was pioneered by Mandl [86] and Brownlie et al. [9], [104], and statistical foundations were explored in [46], [47], [48], [49], [85], [93]. Empirical results indicate that testing of all pairwise interactions in a software system indeed finds a large percentage of existing faults [45], [81]. Indeed, Burr et al. [11] provide more empirical results to show that this type of test coverage leads to useful *code* coverage as well. Dalal *et al.* present empirical results to argue that the testing of all pairwise interactions in a software system finds a large percentage of the existing faults [45]. Dunietz *et al.* link the effectiveness of these methods to software code coverage. They show that high code block coverage is obtained when testing all two-way interactions, but higher subset sizes are needed for good path coverage [54]. Kuhn et al. examined fault reports for three software systems. They show that 70% of faults can be discovered by testing all two-way interactions, while 90% can be detected by testing all three way interactions. Six-way coverage was required in these systems to detect 100% of the faults reported [81]. This study was followed by similar experiments, such as one of 109 software-controlled medical devices that were recalled by the U.S. Food and Drug Administration (FDA) [82]. These experiments found that 97% of the flaws in these 109 cases could be detected with pair-wise testing of parameter settings. Only three devices required coverage higher than two.

Williams et al. [134] quantify the coverage for a particular interaction strength. For instance, if we have four factors, any new test case can contribute at most $\binom{4}{2}$, or 6 new covered pairs. Further, if each factor has three levels, there are a total of $\binom{4}{2}\ 3^2 = 54$ possible pairs that must be covered. Therefore any one new test case increases our coverage by at most 11.1% [134]. A similar method is described by Dunietz et al. [54]. We would expect that a real testing environment has the ability to capture variable coverage requirements for a given test suite. The nature of a covering array as it is defined only guarantees a coverage of $t-$subsets. We may also wish to have coverage of some subsets of size t' for values of $t' > t$, where for example, these higher-order subsets correspond to costly interaction failures. A combinatorial model for variable strength coverage is introduced in [30], [32], [31]. An interesting special case arises when a graph G is used to specify that certain pairs of factors must have all pairs of values covered, while other pairs of factors need not. This problem of constructing a *covering array on the graph G* poses many interesting combinatorial questions [22], [90], but is beyond our scope here.

6.2. Hardware Testing

Imagine a circuit with k inputs that we are to test. Within the circuit, the input signals interact through arithmetic and logical operations to determine an output vector. From the specification, we know the output vector that should be produced, but errors may occur. As with software, we expect errors to be evinced by setting a fraction of the inputs to specified high or low values. Tang *et al.* [124], [125], Boroday et al. [8], and Chandra et al. [17] study hardware (circuit testing) in this environment, proposing test coverage that includes each of the 2^t input settings for each subset of t inputs. Seroussi and Bshouti [111] give a comprehensive treatment. In each case, the test suite is a binary covering array of strength t. Dumer [52] examines the related question of isolating memory faults, and again uses binary covering arrays.

6.3. Testing Advanced Materials

Cawse [16] reports on experimental plans for mixture experiments, in which materials are combined to yield improved strength, flexibility, melting point, and the like. In this application, *avoiding* certain combinations is not just desirable – rather it is necessary, as certain mixtures can be explosive, or toxic. With this in mind, the goal is to consider a representative sample of mixtures, under a variety of controlled environmental conditions. Cawse advocates the use of covering arrays for such experimentation.

6.4. Interactions Regulating Gene Expression

Regulation of developmental and biological processes depends upon certain signals (hormones) impact the expression of a particular gene. Multiple signals may interact in regulation, by jointly inhibiting or enhancing gene expression. Numerous examples are given in [112]. Identification of signal interactions cannot be achieved by examining all possible signal combinations, but a (by now) familiar theme emerges. Interactions among few signals are those of most interest. Covering arrays provide precisely the experimental plan to ensure that all "small" potential interactions are explored. Shasha *et al.* [112] provide details on the application.

7. Conclusions.

An exhaustive survey of current knowledge concerning covering arrays would treat orthogonal arrays, probabilistic techniques, and techniques for strength $t > 3$ in more detail than we have done. Instead we have chosen some specific constructions in order to illustrate direct constructions, recursive constructions, and heuristic algorithms. At the same time, applications to testing problems have been outlined, in part to convey to a mathematical audience problems faced in practical testing.

Perhaps the synergy between the mathematical research and the software testing application is the most promising for future research on covering arrays. As first suggested in [24], effective test generators must use both mathematical insight and heuristic computation to make covering arrays for the broad applications intended. our emphasis here has been on the combinatorial constructions that can profit from heuristic computation of ingredients.

Among areas requiring substantial further research, we mention the development of more effective lower bounds for "small" covering arrays. In the arena of heuristic computation, comprehensive evaluations of the great deluge algorithm and of genetic algorithms are needed. In terms of combinatorial constructions, it appears that methods for the construction of orthogonal arrays are concerned substantially with *balancing* appearances of t-tuples, while construction of covering arrays are relieved of this burden. This is the forte of Roux-type constructions, on which we have focussed here. We expect investigations of similar constructions that focus on coverage rather than balance to provide further powerful constructions.

Acknowledgments. Thanks to the U.S. Army Research Office and the Consortium for Embedded and Internetworking Technologies for financial support. Thanks to Jim Cawse, Mark Chateauneuf, Myra Cohen, Jim Collofello, Sid Dalal, Jeff Dinitz, Peter Gibbons, Alan Hartman, Dean Hoskins, Don Kreher, Alan Ling, Sosina Martirosyan, Lucia Moura, Gary Mullen, George Sherwood, Neil Sloane, Brett Stevens, Renée Turban, and Robby Walker for discussions about various aspects of the research outlined here. Thanks to Lucia Gionfriddo, Mario Gionfriddo, and Gaetano Quattrocchi for their great hospitality, and the opportunity to present this work.

REFERENCES

[1] N. Alon, *Explicit construction of exponential sized families of k-independent sets,* Discrete Mathematics, 58 (1986), pp. 191–193.

[2] N. Alon - O. Goldreich - J. Hastad - R. Peralta, *Simple constructions of almost k-wise independent random variables,* Random Structures and Algorithms, 3 (1992), pp. 289–304; addendum in Random Structures and Algorithms, 4 (1993), pp. 119–120.

[3] M. Atici - S.S. Magliveras - D.R. Stinson - W.-D. Wei, *Some recursive constructions for perfect hash functions,* J. Combinatorial Designs, 4 (1996), pp. 353–363.

[4] J. Azar - R. Motwani - J. Naor, *Approximating probability distributions using small sample spaces,* Combinatorica, 18 (1998), pp. 151–171.

[5] J. Bierbrauer - H. Schellwatt, *Almost independent and weakly biased arrays: efficient constructions and cryptologic applications,* Advances in Cryptology (Crypto 2000), Lecture Notes in Computer Science, 1880 (2000), pp. 533–543.

[6] S.R. Blackburn, *Perfect hash families: probabilistic methods and explicit constructions,* J. Combinatorial Theory (A), 92 (2000), pp. 54–60.

[7] S.Y. Boroday, *Determining essential arguments of Boolean functions (Russian),* Proc. Conference on Industrial Mathematics, Taganrog, 1998, pp. 59–61.

[8] S.Y. Boroday - I.S. Grunskii, *Recursive generation of locally complete tests,* Cybernetics and Systems Analysis, 28 (1992), pp. 20–25.

[9] R. Brownlie - J. Prowse - M. S. Phadke, *Robust testing of AT&T PMX/S tar MAIL using OATS,* AT& T Technical Journal, 71 (3): 41–7, 1992.

[10] C.J. Burgess, *Software testing using an automatic generator of test data,* Proc. First International Conference on Software Quality Management (M. Ross, editor), 1993, pp. 541–556.

[11] K. Burr - W. Young, *Combinatorial test techniques: Table-based automation, test generation and code coverage,* In Proc. of the Intl. Conf. on Software Testing Analysis Review, 1998, San Diego.

[12] K. Burroughs - A. Jain - R.L. Erickson, *Improved quality of protocol testing through techniques of experimental design,* Proc. SuperComm, IEEE Conf. Communications, 1994, pp. 745–752.

[13] C.T. Carroll, *The cost of poor testing: A U.S. government study I,* EDPACS: The EDP Audit, Control, and Security Newsletter, 31, 1 (2003), pp. 1-17.

[14] C.T. Carroll, *The cost of poor testing: A U.S. government study II,* EDPACS: The EDP Audit, Control, and Security Newsletter, 31,2 (2003), pp. 1–16.

[15] J. Carter - M. Wegman, *Universal classes of hash functions,* J. Computer and Systems Sciences, 18 (1979), pp. 143–154.

[16] J.N. Cawse (ed.), *Experimental Design for Combinatorial and High Throughput Materials Development,* John Wiley & Sons, New York, 2003.

[17] A.K. Chandra - L.T. Kou - G. Markowsky - S. Zaks, *On sets of boolean n-vectors with all k-projections surjective,* Acta Informatica, 20 (1983), pp. 103–111.

[18] M. Chateauneuf, *Covering Arrays,* Ph.D. Thesis, Michigan Technological University, 2000.

[19] M. Chateauneuf - C.J. Colbourn - D.L. Kreher, *Covering arrays of strength three,* Designs Codes and Cryptography, 16 (1999), pp. 235–242.

[20] M. Chateauneuf - D.L. Kreher, *On the state of strength-three covering arrays,* J. Combinatorial Designs, 10(4) (2002), pp. 217–38.

[21] D. Chays - Y. Deng - P. Frankl - S. Dan - F. Vokolos - E. Weyuker, *An AGENDA for testing relational database applications,* J. Software Testing, Verification, and Reliability, 14,10 (2004), pp. 1–29.

[22] C. Cheng - A. Dumitrescu - P. Schroeder, *Generating small combinatorial test suites to cover input-output relationships,* Proceedings of the Third International Conference on Quality Software (QSIC '03), Dallas, November 2003, pp. 76–82.

[23] D.M. Cohen - S.R. Dalal - M.L. Fredman - G.C. Patton, *The AETG system: an approach to testing based on combinatorial design,* IEEE Transactions on Software Engineering, 23 (7) (1997), pp. 437–44.

[24] D.M. Cohen - S.R. Dalal - M.L. Fredman - G.C. Patton, *Method and system for automatically generating efficient test cases for systems having interacting elements,* United States Patent, Number 5, 542, 043 (1996).

[25] D.M. Cohen - S.R. Dalal - A. Kajla - G.C. Patton, *The automatic efficient test generator,* Proc. Fifth International Symp. Software Reliability Engineering, IEEE, Los Alamitos CA, 1994, pp. 303–309.

[26] D.M. Cohen - S.R. Dalal - J. Parelius - G.C. Patton, *The combinatorial design approach to automatic test generation,* IEEE Software, 13 (5) (1996), pp. 83–8.

[27] D.M. Cohen - M.L. Fredman, *New techniques for designing qualitatively independent systems,* J. Combinatorial Designs, 6 (6) (1998), pp. 411–16.

[28] G. Cohen - S. Litsyn - G. Zémor, *On greedy algorithms in coding theory,* IEEE Trans. Inform. Theory, 42 (1996), pp. 2053–2057.

[29] G. Cohen - G. Zémor, *Intersecting codes and independent families,* IEEE Transactions on Information Theory IT-40 (1994), pp. 1872–1881.

[30] M.B. Cohen, *Designing Test Suites for Software Interaction Testing,* Ph. D. Thesis, University of Auckland, 2004.

[31] M.B. Cohen - C.J. Colbourn - J.S. Collofello - P.B. Gibbons - W.B. Mugridge, *Variable Strength Interaction Testing of Components,* In Proc. Intl. Computer Software and Applications Conference, (COMPSAC 2003), 2003, Dallas TX, pp. 413–418.

[32] M.B. Cohen - C.J. Colbourn - P.B. Gibbons - W.B. Mugridge, *Constructing test suites for interaction testing,* In Proc. Intl. Conf. on Software Engineering, (ICSE 2003), 2003, pp. 38–48, Portland OR.

[33] M.B. Cohen - C.J. Colbourn - A.C.H. Ling, *Augmenting Simulated Annealing to Build Interaction Test Suites,* In Proc. Intl. Symposium Software Requirements Engineering, (ISSRE 2003), pp. 394–405.

[34] M.B. Cohen - C.J. Colbourn - A.C.H. Ling, *Constructing Strength 3 Covering Arrays with Augmented Annealing,* Discrete Mathematics, to appear.

[35] C.J. Colbourn - M.B. Cohen - R.C. Turban, *A Deterministic Density Algorithm for Pairwise Interaction Coverage,* Proc. IASTED International Conf. Software Engineering, February 2004, Innsbruck Austria, pp. 245–252.

[36] C.J. Colbourn - W. De Launey, *Difference Matrices,* in [37], pp. 287–297.

[37] C.J. Colbourn - J.H. Dinitz (editors), *The CRC Handbook of Combinatorial Designs,* CRC Press, Boca Raton, 1996.

[38] C.J. Colbourn - J.H. Dinitz, *Making the MOLS table,* In Computational and Constructive Design Theory, 1996. (W.D. Wallis, ed.) Kluwer Academic Press, pp. 67–134.

[39] C.J. Colbourn - S. Martirosyan - G.L. Mullen - D.E. Shasha - G. Sherwood - J.L. Yucas, *Concerning covering arrays of strength two,* J. Combinatorial Designs, to appear.

[40] G.T. Daich, *New Spreadsheet Tool Helps Determine Minimal Set of Test Parameter Combinations,* CrossTalk: Journal of Defense Software Engineering (August 2003), pp. 26–30.

[41] G.T. Daich, *Testing combinations of parameters made easy,* In Proc. IEEE Systems Readiness Conference, 2003, pp. 379–384.

[42] S.R. Dalal - J.R. Horgan - J.R. Kettenring, *Reliable software and communication: Software quality, reliability, and safety,* Proc. Int. Conf. Software Engineering (ICSE93), May 1993, pp. 425–435.

[43] S.R. Dalal - A. Jain - N. Karunanithi - J.M. Leaton - C.M. Lott, *Model-based testing of a highly programmable system,* Proc. International Symp. Software Reliability Engineering, IEEE, 1998, pp. 174–178.

[44] S.R. Dalal - A. Jain - G.C. Patton - M. Rathi - P. Seymour, *AETG^{SW}Web: A web-based service for automatic efficient test generation from functional requirements,* Proc. Second IEEE Workshop on Industrial Strength Formal Specification Techniques, IEEE Press, 1998, pp. 84–85.

[45] S.R. Dalal - A.J.N. Karunanithi - J.M.L. Leaton - G.C.P. Patton - B.M. Horowitz, *Model-based testing in practice,* In Proc. of the Intl. Conf. on Software Engineering, (ICSE '99), 1999, pp. 285–94, New York.

[46] S.R. Dalal - C.L. Mallows, *When should one stop testing software?,* Journal of the American Statistical Association, 83 (1988), pp. 872–879.

[47] S.R. Dalal - C.L. Mallows, *When to stop testing software? Some exact results,* Lecture Notes in Statistics, 75 (1992), pp. 267–276.

[48] S.R. Dalal - C.L. Mallows, *Buying with exact confidence,* Annals Applied Probability, 2 (1992), pp. 752–765.

[49] S.R. Dalal - C.L. Mallows, *Factor-covering designs for testing software,* Technometrics, 40 (1998), pp. 234–243.

[50] G. Dueck, *New Optimization Heuristics - The Great Deluge Algorithm and the Record-to-Record Travel,* J. Computational Physics, 104 (1993), pp. 86–92.

[51] G. Dueck - T. Scheuer, *Threshold Accepting: A general purpose optimization algorithm appearing superior to simulated annealing,* J. Computational Physics, 90 (1990), pp. 161–75.

[52] I.I. Dumer, *Asymptotically optimal codes correcting memory defects of fixed multiplicity,* Problemy Peredachi Informatskii, 25 (1989), pp. 3–20.

[53] A. Dumitrescu, *Efficient algorithms for generation of combinatorial covering suites,* Proceedings of the the 14-th Annual International Symposium on Algorithms and Computation (ISAAC '03). Lecture Notes in Computer Science, Vol. 2906, 2003, pp. 300–308.

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

[55] J. Duran - S. Ntafos, *An evaluation of random testing,* IEEE Transactions on Software Engineering SE-10 (1984), pp. 438–444.

[56] L. Gargano - J. Körner - U. Vaccaro, *Sperner theorems on directed graphs and qualitative independence,* J. Combinatorial Theory (A), 61 (1992), pp. 173–192.

[57] L. Gargano - J. Körner - U. Vaccaro, *Sperner capacities,* Graphs and Combinatorics, 9 (1993), pp. 31–46.

[58] L. Gargano - J. Körner - U. Vaccaro, *Capacities: from information to extremal set theory,* J. Combinatorial Theory (A), 68 (1994), pp. 296–316.

[59] M.X. Goemans - D.P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming,* Journal of the ACM (JACM), 42 (6) 1995, pp. 1115-1145.

[60] A.P. Godbole - D.E. Skipper - R.A. Sunley, *t-Covering arrays: upper bounds and Poisson approximations,* Combinatorics, Probability and Computing, 5 (1996), pp. 105–118.

[61] M. Grindal - J. Offutt - S. F. Andler, *Combination testing strategies: a survey,* J. Software Testing, Verification and Reliability, 15 (2005), pp. 97–133.

[62] M. Grindal - B. Lindström - A. J. Offutt - S. F. Andler, *An evaluation of combination strategies for test case selection Technical Report HS-IDA-TR-03-001,* Computer Science, University of Skövde, Sweden, 2003.

[63] D. Hamlet - R. Taylor, *Partition testing does not inspire confidence,* IEEE Transactions on Software Engineering SE-16 (1990), pp. 1402–1412.

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

[65] A. Hartman - L. Raskin, *Problems and algorithms for covering arrays,* Discrete Math., 284 (2004), pp. 149–156.

[66] A. Hedayat - N. Sloane - J. Stufken, *Orthogonal Arrays,* Springer-Verlag, New York, 1999.

[67] E. Heller, *Using DOE structures to generate software test cases,* Proc. Twelfth International Conference on Testing Computer Software (Washington DC), 1995, pp. 33–39.

[68] J.R. Horgan - S. London - M.R. Lyu, *Achieving software quality with testing coverage measures,* IEEE Computer, 27 9 (1994), pp. 60–69.

[69] D.S. Hoskins - R.C. Turban - C.J. Colbourn, *Experimental Designs in Software Engineering: D-Optimal Designs and Covering Arrays,* Proc. SIGSOFT 2004/FSE–12 Workshop on Interdisciplinary Software Engineering Research (WISER 2004), November 2004, to appear.

[70] J. Huller, *Reducing time to market with combinatorial design method testing,* Proc. 10th Annual International Council on Systems Engineering (INCOSE2000), Minneapolis MN, July 2000.

[71] N. Ido - T. Kikuno, *Lower bounds estimation of factor-covering design sizes,* J. Combinatorial Designs, 11 (2003), pp. 89–99.

[72] D.C. Ince, *The automatic generation of test data,* Computer Journal, 30 (1987), pp. 63–69.

[73] C. Kaner - J. Bach - B. Pettichord, *Lessons Learned in Software Testing,* John Wiley & Sons, New York, 2001.

[74] G. Katona, *Two applications (for search theory and truth functions) of Sperner type theorems,* Periodica Math., 3 1973, pp. 19–26.

[75] T.M. Khoshgoftaar - I.C. Munson, *Predicting software development errors using software complexity metrics,* IEEE Journal on Selected Areas in Communications SAC-8 (1990), pp. 253–261.

[76] D. Kleitman - J. Spencer, *Families of k-independent sets,* Discrete Math, 6 1973, pp. 255–262.

[77] N. Kobayashi - T. Tsuchiya - T. Kikuno, *Applicability of non-specification-based approaches to logic testing for software,* Proc. Intl. Conf. Dependable Systems and Networks, July 2001, pp. 337–346.

[78] N. Kobayashi - T. Tsuchiya - T. Kikuno, *A new method for constructing pair-wise covering designs for software testing,* Information Processing Letters, 81 (2002), pp. 85–91.

[79] J. Körner - M. Lucertini, *Compressing inconsistent data,* IEEE Trans. Information Theory, 40 (1994), pp. 706–715.

[80] J. Körner - G. Simonyi, *A Sperner type theorem and qualitative independence,* J. Combinatorial Theory (A), 59 (1992), pp. 90–103.

[81] D. Kuhn - M. Reilly, *An investigation into the applicability of design of experiments to software testing,* Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002, pp. 91–95.

[82] R. Kuhn - D. Wallace - A. Gallo, *Software Fault Interactions and Implications for Software Testing,* IEEE Transactions of Software Engineering, June 2004, 30 (6), to appear.

[83] K. Kurosawa - T. Johansson - D.R. Stinson, *Almost k-wise independent sample spaces and their cryptologic applications,* Advances in Cryptology (Eurocrypt 97), Lecture Notes in Computer Science, 1233 (1997), pp. 409–421.

[84] Y.K. Malaija - N. Li - F. Karcich - R. Skibbe, *The relationship between test coverage and reliability,* Proc. Fifth International Conference on Software Reliability Engineering, 1994, pp. 186–195.

[85] C.L. Mallows, *Covering designs in random environments,* in: The Practice of Data Analysis (D. R. Brillinger, L.T. Frenholz, and S. Morgenthaler; editors), Princeton University Press, Princeton NJ, 1997, pp. 235–245.

[86] R. Mandl, *Orthogonal latin squares: an application of experiment design to compiler testing,* Communications of the ACM, 28 (10) (1985), pp. 1054–8.

[87] E. Marczewski, *Independence d'ensembles et prolongement de mesures,* Colloq. Math., 1 (1948), pp. 122–132.

[88] S. Martirosyan - Tran Van Trung, *On t-covering arrays,* Designs, Codes and Cryptography, 32 (2004), pp. 323–339.

[89] K. Meagher - B. Stevens, *Group construction of covering arrays,* J. Combinatorial Designs, 13 (2005), pp. 70–77.

[90] K. Meagher - B. Stevens, *Covering arrays on graphs,* J. Combinatorial Theory, Series B, 95 (2005), pp. 134–151.

[91] L. Moura - J. Stardom - B. Stevens - A. Williams, *Covering arrays with heterogeneous alphabet sizes,* J. Combinatorial Designs, 11 (2003), pp. 413–432.

[92] J.D. Musa - A. Iannino - K. Okumoto, *Software Reliability: Measurement, Prediction, Application,* McGraw Hill, 1987.

[93] V. Nair - D. James - W. Ehrlich - J. Zevallos, *A statistical assessment of some software testing strategies and application of experimental design techniques,* Statistica Sinica, 8 1 (1998), pp. 165–184.

[94] J. Naor - M. Naor, *Small-bias probability spaces: efficient constructions and applications,* SIAM J. Computing, 22 (1993), pp. 838–856.

[95] M. Naor - O. Reingold, *On the construction of pseudo-random permutations: Luby-Rackoff revisited,* Proc. Twenty-ninth ACM Symposium on the Theory of Computing, 1997, ACM Press, pp. 189–199.

[96] M. Naor - L.J. Schulman - A. Srinivasan, *Splitters and near-optimal randomization,* In Proc. 36th Annual Symp. Foundations of Computer Science (FOCS), IEEE, 1996, pp. 182–191.

[97] National Institute of Standards and Technology, *The Economic Impacts of Inadequate Infrastructure for Software Testing,* U.S. Department of Commerce, May 2002.

[98] B.D. Nordwall, *Buying Off-the-Shelf Challenges Military,* Aviation Week & Space Technology, 146 (18), 1997.

[99] K. Nurmela, *Upper bounds for covering arrays by tabu search,* Discrete Applied Math., 138 (2004), pp. 143–152.

[100] K. Nurmela - P.R.J. Östergård, *Constructing covering designs by simulated annealing,* Technical report, Digital Systems Laboratory, Helsinki Univ. of Technology, 1993.

[101] P.R.J. Östergård, *Constructions of mixed covering codes,* Technical report, Digital Systems Laboratory, Helsinki Univ. of Technology, 1991.

[102] T.J. Ostrand - M.J. Balcer, *The category-partition method for specifying and generating functional tests,* Communications of the ACM, 31 (1988), pp. 676–686.

[103] W.B. Perkinson, *A methodology for designing and executing ISDN feature tests using automated test systems,* Proc. IEEE Globecom 92, December 1992, pp. 399–403.

[104] M.S. Phadke, *Quality Engineering Using Robust Design,* Prentice-Hall Inc., New Jersey, 1989.

[105] S. Poljak - A. Pultr - V. Rödl, *On qualitatively independent partitions and related problems,* Discrete Applied Math., 6 (1983), pp. 193–205.

[106] S. Poljak - Z. Tuza, *On the maximum number of qualitatively independent partitions,* J. Combinatorial Theory (A), 51 (1989), pp. 111–116.

[107] A. Réyni, *Foundations of Probability,* Wiley, New York, 1971.

[108] G. Roux, *k-Propriétés dans les tableaux de n colonnes: cas particulier de la k-surjectivité et de la k-permutivité,* Ph. D. Thesis, Université de Paris, 1987.

[109] J. Ryu - M. Kim - S. Kang - S. Seol, *Interoperability test suite generation for the TCP data part using experimental design techniques,* In Proc. Thirteenth Int. Conf. Testing Communicating Systems (Ottawa, Canada), 2000, pp. 127–142.

[110] P.J. Schroeder - P. Faherty - B. Korel, *Generating expected results for automated black-box testing,* Proc. IEEE Intl. Conf. Automated Software Engineering (ASE2002), September 2002, pp. 139–148.

[111] G. Seroussi - N.H. Bshouty, *Vector sets for exhaustive testing of logic circuits,* IEEE Transactions on Information Theory, 34 (1988), pp. 513–522.

[112] D.E. Shasha - A.Y. Kouranov - L.V. Lejay - M.F. Chou - G.M. Coruzzi, *Using combinatorial design to study regulation by multiple input signals: A tool for parsimony in the post-genomics era,* Plant Physiology, 127 (2001), pp. 1590-1594.

[113] G.B. Sherwood, *Efficient testing of factor combinations,* Proc. Third International Conf. Software Testing, Analysis, and Review (Jacksonville FL), 1994.

[114] G.B. Sherwood - S.S. Martirosyan - C.J. Colbourn, *Covering Arrays of Higher Strength from Permutation Vectors,* J. Combinatorial Designs, to appear.

[115] N. Sloane, *Covering arrays and intersecting codes,* J. Combinatorial Designs, 1 (1993), pp. 51–63.

[116] J. Stardom, *Metaheuristics and the search for covering and packing arrays,* Master's thesis, Simon Fraser University, 2001.

[117] B. Stevens, *Transversal Covers and Packings,* Ph. D. Thesis, Mathematics, University of Toronto, 1998.

[118] B. Stevens - A.C.H. Ling - E. Mendelsohn, *A direct construction of transversal covers using group divisible designs,* Ars Combinatoria, 63 (2002), pp. 145–159.

[119] B. Stevens - E. Mendelsohn, *Efficient software testing protocols,* In Proc. of Center for Advanced Studies Conf. (Cascon '98), 1998, pp. 270–293, Ontario.

[120] B. Stevens - E. Mendelsohn, *New recursive methods for transversal covers,* J. Combinatorial Designs, 7 (3) (1999), pp. 185–203.

[121] B. Stevens - E. Mendelsohn, *Packing arrays and packing designs,* Designs Codes and Cryptography, 27 (2002), pp. 165–176.

[122] B. Stevens - L. Moura - E. Mendelsohn, *Lower bounds for transversal covers,* Designs Codes and Cryptography, 15 (1998), pp. 279–299.

[123] K.C. Tai - L. Yu, *A test generation strategy for pairwise testing,* IEEE Transactions on Software Engineering, 28 (2002), pp. 109–111.

[124] D.T. Tang - C.L. Chen, *Iterative exhaustive pattern generation for logic testing,* IBM Journal Research and Development, 28 (1984), pp. 212–219.

[125] D.T. Tang - L.S. Woo, *Exhaustive test pattern generation with constant weight vectors,* IEEE Trans. Computers, 32 (1983), pp. 1145–1150.

[126] K. Tatsumi - S. Watanabe - Y. Takeuchi - H. Shimokawa, *Conceptual support for test case design,* Proc. 11th Intl. Computer Software and Applications Conf. (COMPSAC), October 1987, pp. 285–290.

[127] Y.-W. Tung - W.S. Aldiwan, *Automating test case generation for the new generation mission software system,* In Proc. IEEE Aerospace Conf., 2000 pp. 431–437.

[128] R.C. Turban - C.J. Colbourn - M.B. Cohen, *A Framework of Greedy Methods for Constructing Interaction Tests,* preprint, 2004.

[129] M. Wegman - J. Carter, *New hash functions and their use in authentication and set equality,* J. Computer and System Sciences, 22 (1981), pp. 265–279.

[130] C.H. West, *Protocol validation - principles and applications,* Computer Networks and ISDN Systems, 24 (1992), pp. 219–242.

[131] A.W. Williams, *Determination of test configurations for pair-wise interaction coverage,* In Proc. Thirteenth Int. Conf. Testing Communication Systems, 2000, pp. 57–74.

[132] A.W. Williams, *Software Component Interaction Testing,* Ph. D. thesis, University of Ottawa (Canada), 2002.

[133] A.W. Williams - R.L. Probert, *A practical strategy for testing pair-wise coverage of network interfaces,* In Proc. Seventh Intl. Symp. on Software Reliability Engineering, 1996, pp. 246–54.

[134] A.W. Williams - R.L. Probert, *A measure for component interaction test coverage,* In Proc. ACS/IEEE Intl. Conf. on Computer Systems and Applications, 2001, pp. 301–311.

[135] A.W. Williams - R.L. Probert, *Formulation of the interaction test coverage problem as an integer program,* In Proc. 14th Intl. Conf on Testing of Communicating Systems, 2002, pp. 283–98.

[136] W.E. Wong - J.R. Horgan - S. London - A.P. Mathur, *Effect of test set minimization on fault detection effectiveness,* Proc. Seventeeth International Conf. Software Engineering, IEEE, 1995, pp. 41–50.

[137] C. Yilmaz - M.B. Cohen - A. Porter, *Covering arrays for efficient fault characterization in complex configuration spaces,* In Proc. Int. Symp. Software Testing and Analysis (ISSTA), 2004, pp. 45–54.

[138] J. Yin, *Constructions of difference covering arrays,* J. Combinatorial Theory (A), 104 (2003), pp. 327–339.

[139] J. Yin, *Cyclic difference packing and covering arrays,* Des. Codes Crypt., to appear.

[140] L. Yu - K.C. Tai, *In-parameter-order: a test generation strategy for pairwise testing,* In Proc. Third IEEE Intl. High-Assurance Systems Engineering Symp., 1998, pp. 254–261.

Computer Science and Engineering
Arizona State University
P.O. Box 878809
Tempe, Arizona 85287, (U.S.A.)
e-mail: colbourn@asu.edu