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 \( \lambda \) 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 \( \lambda \) 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 \( CA_\lambda(N; t, k, v) \) is an \( N \times k \) array. In every \( N \times t \) subarray, each \( t \)-tuple occurs at least \( \lambda \) 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 $\text{CAN}(t, k, v)$. 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 transposed 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.

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

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 $\text{MCA}_\lambda(N; t, k, (v_1, v_2, \ldots, v_k))$ 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 $\text{CAN}(t, k, (v_1, v_2, \ldots, v_k))$ 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:

$$k = \left( \frac{N - 1}{\lceil \frac{N}{2} \rceil} \right)$$

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 ($M_1, \ldots, M_v$); 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 $\mathcal{P} = \{R_1, R_2, \ldots, R_k\}$ of partitions of $R$. A family of partitions is $t$-qualitatively independent when for every $t$ of the partitions $R_{i_1}, \ldots, R_{i_t}$, 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 $\text{CA}(t, k, 2)$. Naor and Naor [94] establish that $(k, t)$-universal sets arise as probability spaces with limited independence; indeed these have been extensively studied as $\epsilon$-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:

$$v' \leq \text{CAN}(t, k, v).$$
More generally for a 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

\[
(2) \quad \text{CAN}(t, k, (v_1, v_2, \ldots, v_k)) \leq \text{CAN}(t, k, (v_1 + 1, v_2, \ldots, v_k)
\]

and consequently

\[
(3) \quad \text{CAN}(t, k - 1, (v_2, \ldots, v_k)) \leq \text{CAN}(t, k(v_1, v_2, \ldots, v_k)
\]

Suppose \(A\) is a covering array \(\text{CA}(N; t, k, v)\) 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 \(\text{CA}(N'; t - 1, k - 1, v)\), where \(N'\) is the number of occurrences of \(x\) in row \(i\). Hence

\[
(4) \quad \text{CAN}(t - 1, k - 1, v) \leq \frac{1}{v} \text{CAN}(t, k, v).
\]

By the same token,

\[
(5) \quad \text{CAN}(t, k, (v_1, v_2, \ldots, v_k)) \geq v_1 \cdot \text{CAN}(t - 1, k - 1, (v_2, \ldots, v_k))
\]

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 \(\text{CA}(N; t, k, v)\). It follows from their results that

\[
\text{CAN}(t, k, v) \leq \frac{(t - 1)\log k}{\log \frac{v}{v - 1}}(1 + o(1)).
\]

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

\[
\text{CAN}(t, k, 2) \leq 2^t \log^2 \log k.
\]
Gargano, Kőrner, and Vaccaro [57] show that the ratio of $\text{CAN}(2, k, v)$ to $\log k$ is asymptotic to $\frac{1}{v}$. 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 $\text{CA}(N; t, k, v)$ exists, it has been established that $k$ is at least $\frac{2v}{\sqrt{v^t}}$ and at most $\frac{k_{v,N}4v}{\sqrt{v^t}}$, where $k_{v,N}$ 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 $\text{OA}(N; t, k, v)$ 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 $\text{OA}(2, k, v)s$, and to [66] for higher strength.

We mention one important direction here. A group divisible design $\langle V, G, B \rangle (k-GDD)$ is a set $V$ of $v$ elements, a partition $G$ 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 $g_1, \ldots, g_n$, the type of the $k$-GDD is $g_1 g_2 \cdots g_n$. Exponential notation is used, so that $g^\ell$ indicates $\ell$ occurrences of the group size $g$. A $k$-GDD of type $v^k$ is a transversal design $\text{TD}(k, v)$. 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 $v^k$ 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 $v^k$ exists, we can deduce that $\text{CAN}(2, n, v) \leq \frac{v^{k(n-1)}}{v^{k+1}}$. 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 $\text{CAN}(2, v+3, v) \leq v^2 + 2v$ when $v+1$ is a prime or prime power.
Lemma 3.1. A \((g, \ell)\)-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:

\[
\begin{align*}
(6, 8) &= (\infty, 0.1, 3, 0, 2, 1.4) & (6, 9) &= (\infty, 0.0, 1, 0, 0, 3, 2, 4) \\
(6, 10) &= (\infty, 0.0, 0.1, 1.4, 3, 0, 2) & (6, 11) &= (\infty, 0.0, 0.0, 1, 0, 4, 2, 3, 0) \\
(6, 12) &= (\infty, 0.0, 0.0, 0.1, 0.4, 2, 3, 0) & (7, 7) &= (\infty, 0.2, 1.4, 5, 3) \\
(7, 9) &= (\infty, 0.0, 2, 1, 4, 5, 3, 3) & (7, 10) &= (\infty, 0.0, 0.0, 1, 0, 3, 5, 4, 2) \\
(7, 11) &= (\infty, 0.0, 0.0, 1.0, 3, 5, 4, 2) & (7, 12) &= (\infty, 0.0, 0.0, 0.0, 1, 0, 3, 5, 4, 2) \\
(7, 13) &= (\infty, 0.0, 0.0, 0.0, 1.0, 3, 5, 4, 2) & (8, 11) &= (\infty, 0.1, 0.0, 0, 3, 5, 4, 2, 6) \\
(8, 12) &= (\infty, 0.0, 0.1, 0.6, 4, 6, 3, 2, 5) & (8, 13) &= (\infty, 0.0, 0.0, 1, 2, 6, 5, 3, 6, 2, 4) \\
(8, 14) &= (\infty, 0.0, 0.0, 1.0, 4, 6, 4, 3, 2, 5) & (9, 12) &= (\infty, 0.1, 6, 4, 5, 0, 7, 6, 2, 1, 3) \\
(9, 13) &= (\infty, 0.0, 0.0, 2, 6, 1, 7, 6, 3, 4, 5) & (9, 14) &= (\infty, 0.0, 0.0, 0, 2, 6, 1, 7, 6, 3, 4, 5) \\
(9, 15) &= (\infty, 0.0, 0.0, 0.0, 2, 6, 1, 7, 6, 3, 4, 5)
\end{align*}
\]

### 3.2. Some Direct Constructions for Strength Three

Here we examine group action focusing on strength three, following the
presentation in [19]. Let \( \Omega \) 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 \( \Omega \). If \( g \in G \) and \( M \) is a
\( k \times \ell \) matrix with entries in \( \Omega \), 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

$$F_1, F_2, \ldots, F_{2v-1}$$

be a one-factorization of the complete graph $\mathcal{G}$ on $X$. Let $\ell : E(\mathcal{G}) \rightarrow \Omega$ be an arbitrary function for which $\ell(e) \neq \ell(e')$ whenever $e, e' \in F_j$ for some $j$. Such a function exists since $|\Omega| = q + 1 > v$. Define the $2v \times (2v - 1)$ matrix $M$ with entries in $\Omega$ by $M[x, j] = \ell(e)$, where $e$ is the edge of $F_j$ 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 $F_j$ and $x_3$ is incident to some other edge $e'$ of $F_j$. Thus $\ell(e) = a$ and $\ell(e') = b$ for some $a \neq b$ in $\Omega$. 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 $F_j'$ 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 \( \text{Sym}\{0, 1, 2\}. \) 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 \( \text{CAN}(3, 6, 3) = 33. \)

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 2 & 1 & 1 & 2 & 0 & 0 & 0 & 2 & 1 & 2 & 0 & 2 & 1 & 1 & 0 & 0 & 1 & 1 & 2 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 1 & 2 &
\end{array}
\]

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

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

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 \( \ell : E(K_8) \rightarrow \Omega \) be any function such that \( \ell(e) \neq \ell(e') \) whenever \( e, e' \in x^j\mathcal{F} \) for some \( j \) and define the \( 8 \times 7 \) matrix \( M \) with entries in \( \Omega \) 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 $G = \text{Alt}\Omega$ be the alternating group of permutations of $\Omega$. 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 $\Omega$ 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.

```
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
```

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

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

**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 $C = (c_{ij})$ by setting $c_{i,(f-1)k+g} = a_{i,g}$ for $1 \leq i \leq N$, $1 \leq f \leq \ell$, and $1 \leq g \leq k$. Then set $c_{N+i,(f-1)k+g} = b_{i,f}$ for $1 \leq i \leq M$, $1 \leq f \leq \ell$, and $1 \leq g \leq k$. In essence, $k$ copies of $B$ are being appended to $\ell$ 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 $(d_1, \ldots, d_k)$ of an $N \times k$ array is a $k$-tuple in which the entry $d_i$ is the number of $\star$ entries in the $i$th 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 $\{1, \ldots, v\}$, 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($N; 2, k, (v_1, \ldots, v_k)$), $A$, with profile $(d_1, \ldots, d_k)$;
2. for each $1 \leq i \leq k$, an MCA($M_i; 2, \ell_i, (w_{1i}, \ldots, w_{1i,\ell_i})$), $B_i$, with profile $(f_{1i}, \ldots, f_{1i,\ell_i})$, and for which $w_{ij} \leq v_i$ for $1 \leq j \leq \ell_i$.

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

$$\text{MCA}(T; 2, \sum_{i=1}^k \ell_i (w_{1i}, \ldots, w_{1i,\ell_i}, \ldots, w_{ki,\ell_i})).$$
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 $i$th 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 $j$th 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. \qed

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 $\text{CA}(q^2; 2, q, q)$ from the finite field has $q$ disjoint constant rows, but its extension to a $\text{CA}(q^2; 2, q + 1, q)$ can have at most one, no matter how the symbols are relabeled. For example, taking two $\text{CA}(25; 2, 6, 5)$s allows us to produce a $\text{CA}(49; 2, 36, 5)$ while using instead one $\text{CA}(25, 2, 5, 5)$ with five disjoint constant rows allows us to obtain a $\text{CA}(45; 2, 30, 5)$. We examine a generalization that enables us to obtain a $\text{CA}(45; 2, 35, 5)$, 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 $\text{CA}(N; 2, k_1 + k_2, v)$ 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 $\text{OA}(2, q + 1, q)$ yields an
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, (ℓ_1, ℓ_2); v) both exist, then an SCA(N + M − v; 2, (k_1 ℓ_1, k_1 ℓ_2 + k_2 ℓ_1), v) also exists.

**Proof.** Let A consist of the first N − v rows of the SCA(N; 2, (k_1, k_2), v), and B consist of the first M − v rows of the SCA(M; 2, (ℓ_1, ℓ_2); v). Form the product C of A and B as above. Now append v new rows indexed N + M − 2v + i, 1 ≤ i ≤ v, so that in column f(k_1 + k_2) + g the entry is i if f ≤ ℓ_1 and g ≤ k_1, and is 1 otherwise. Then suppress all columns indexed f(k_1 + k_2) + g when f > ℓ_1 and g > k_1. This removes k_2 ℓ_2 columns from C.

We claim that the result R is an SCA(N + M − v; 2, ((k_1 ℓ_1, k_1 ℓ_2 + k_2 ℓ_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 ≤ x ≤ 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 ≤ x ≤ v. But the next M − v rows cover all pairs except possibly those of the form (1, x) for 1 ≤ x ≤ 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 $\text{SMCA}(N; 2, (k_1, k_2), (v_1, \ldots, v_{k_1+k_2}))$ to be an $N \times (k_1 + k_2)$ array in which the $i$th column has entries from a $v_i$-set (which we take to be $\{1, \ldots, v_i\}$), and for which selecting any two columns, we find all pairs first define an $\text{SMCA}(N; 2, (k_1, k_2), (v_1, \ldots, v_{k_1+k_2}))$ to be an $N \times (k_1 + k_2)$ array in which the $i$th column has entries from a $v_i$-set (which we take to be $\{1, \ldots, v_i\}$), 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 $\text{SMCA}(N; 2, (k_1, k_2), (v_1, \ldots, v_{k_1+k_2})), A$, with profile $(d_1, \ldots, d_{k_1+k_2})$;
2. for each $1 \leq i \leq k_1$, an $\text{SMCA}(M_i; 2, (\ell_{i1}, \ell_{i2}), (w_{i1}, \ldots, w_{1,\ell_{i1}+\ell_{i2}})), B_i$, for which $w_{ij} \leq v_i$ for $1 \leq j \leq \ell_{i1} + \ell_{i2}$; and
3. for each $k_1 < i \leq k_1+k_2$, an $\text{SMCA}(M_i; 2, (\ell_{i1}, 0), (w_{i1}, \ldots, w_{1,\ell_{i1}})), B_i$, for which $w_{ij} \leq v_i$ for $1 \leq j \leq \ell_{i1}$.

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

1. For $1 \leq i_1 < i_2 \leq k_1$, for each $\ell_{i_1,1} < j_1 \leq \ell_{i_1,1} + \ell_{i_1,2}$, and each $\ell_{i_2,1} < j_2 \leq \ell_{i_2,1} + \ell_{i_2,2}$, 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 $1 \leq x \leq \min(w_{i_1,j_1}, w_{i_2,j_2})$; and
2. For $1 \leq i_1 \leq k_1$ and $k_1 < i_2 \leq k_1 + k_2$, for each $\ell_{i_1,1} < j_1 \leq \ell_{i_1,1} + \ell_{i_1,2}$, and each $1 \leq j_2 \leq \ell_{i_2,1}$, 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 $1 \leq x \leq w_{i_1,j_1}$.

Then there exists an

$$\text{SMCA}(T; 2, \left(\sum_{i=1}^{k_1} \ell_{i1}, \sum_{i=k_1+1}^{k_1+k_2} \ell_{i1} + \sum_{i=1}^{k_1} \ell_{i2}\right)),$$
For example, a \((5,6)\)-distinct cover starter yields a \(CA(29)\), 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. \(\square\)

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 \(i\)th additional row, the symbol “1” in columns indexed by \((i, j)\) with \(i > k_1\) or \(j > \ell_{i1}\). 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 \(SCA(q^2; 2, (q, 1), q)\). \(\square\)

Some small examples are

\[
\begin{align*}
CA(15; 2, 15, 3) & \quad CA(21; 2, 54, 3) & \quad CA(27; 2, 189, 3) \\
CA(28; 2, 24, 4) & \quad CA(40; 2, 128, 4) & \quad CA(52; 2, 576, 4) \\
CA(45; 2, 35, 5) & \quad CA(65; 2, 200, 5) & \quad CA(85; 2, 1125, 5) \\
CA(91; 2, 63, 7) & \quad CA(133; 2, 490, 7) & \quad CA(175; 2, 3673, 7)
\end{align*}
\]

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 \((g, \ell)\)-distinct cover starter. Treat this as a row, and form the \(\ell\) 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) $(\infty, 0, 0, 0, 1, 3, 2)$
(5, 9) $(\infty, 0, 0, 0, 0, 1, 3, 2)$        (5, 10) $(\infty, 0, 0, 0, 0, 0, 1, 3, 2)$

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

CA(45; 2, 35, 5), CA(49; 2, 41, 5),
CA(53; 2, 48, 5), CA(57; 2, 55, 5),
CA(61; 2, 63, 5), CA(65; 2, 71, 5),
CA(69; 2, 80, 5),

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 $CA(3, 2k, 2)$, we begin by placing two $CA(N_3, 3, k, 2)$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 $k$ share the same value, such a triple is also covered. The remaining triples are handled by appending two $CA(N_2, 2, k, 2)$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.

\[ \text{CAN}(3, 2k, v) \leq \text{CAN}(3, k, v) + (v - 1)\text{CAN}(2, k, v). \]

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 \( \pi \) 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. \( \square \)

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.

\[ \text{CAN}(3, k\ell, v) \leq \text{CAN}(3, k, v) + \text{CAN}(3, \ell, v) + \text{CAN}(2, \ell, v) \times \text{DCAN}(2, k, v). \]

Proof. We suppose that the following all exist:

1. a CA\((N; 3, \ell, v)\) \( A \);
2. a CA\((M; 3, k, v)\) \( B \);
3. a CA\((R; 2, \ell, v)\) \( F \); 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 \( \ell \) copies of the first column of \( B \), then \( \ell \) 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 \( \pi \) on
the $v$ symbols of the array. Then in row $(i-1)R + r$ and column $(j,s)$ of $C_3$
place the entry $\pi^{d_{ij}}(f_{rs})$. 
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 $(i_1, j_1), (i_2, j_2), \text{and} (i_3, j_3)$ 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 $i_1, i_2, \text{and} i_3$ 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 $\pi$). 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 $(\sigma_1, \sigma_2)$ 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 $\pi$ is a $v$-cycle, some power of $\pi$ 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
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 \( \lambda \) 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.** \( CAN(3, q + 1, q + 1) \leq q^3 - q + \binom{q+1}{2} \times CAN(3, q + 1, 2) \)
when \( q \) is a prime power.

To create a \( CA(3, q + 1, q + 1) \) begin with an \( OD(3, q + 1, q + 1) \) 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 \( \binom{q+1}{2} \times CA(N_2; 3, q + 1, 2) \)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.** \( CAN(3, q + 1, q + 1) \leq q^3 - q + \binom{q+1}{2} \times CAN(3, q + 1, 2) - (q^2 - 1) \)
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 $(a, a, \ldots, a)$ and $(b, b, \ldots, b)$. We remove the $2 \times \binom{q+1}{2} = q^2 + q$ rows and add back in $q + 1$ rows of the form $(a, a, \ldots, a)$.

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 $(a, a, \ldots, a)$ 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 $\sigma$ 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 $\{1, \ldots, k\}$ is a linear space of order $k$ if every 2-subset $\{i, j\} \subseteq \{1, \ldots, k\}$ appears in exactly one $B \in \mathcal{B}$.

**Construction 4.10.** Let $q$ be a prime power. Let $\mathcal{B} = \{B_1, \ldots, B_b\}$ be a linear space on $K = \{1, \ldots, k\}$. 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 $B_i \in \mathcal{B}$, 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 $OA_2(2, k, v)$. Each 2-tuple is covered exactly $\lambda$ 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 \( \lambda \) 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.

<table>
<thead>
<tr>
<th>0 0 0 0 0 0 0 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 1 0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>0 0 0 0 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 0 1 0 0 0 1 0 1 0 1</td>
</tr>
<tr>
<td>1 1 0 0 1 1 0 0 1 0 1 0</td>
</tr>
<tr>
<td>1 0 0 1 1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>1 0 1 0 0 0 1 0 0 1 0 1</td>
</tr>
<tr>
<td>0 1 1 1 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

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 \( \lambda \) times. For example, if \( \gcd((n - 1)!, k) = 1 \), then the \( n \times k \) matrix \( D \) defined by \( D[i, j] = ij \mod k \) 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 \( T_i \neq \emptyset \),
1 \leq i \leq r and
\sum_{j=1}^{r} |T_j| = t.

Suppose also that D is a \( (k, \binom{i}{2} + 1; 1) \)-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 \( \Omega \) be a \( v \)-element set and let \( \mathcal{P} \) be a first-order predicate concerning a collection \( C \) of \( t \)-tuples with entries in \( \Omega \) that is invariant under coordinate permutation. For example the predicate \( \mathcal{P} \) could be one of

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

A \( k \times N \) array \( A \) with entries in \( \Omega \) is called a \((N; t, k, v)\)-\( \mathcal{P} \) array if each collection \( C \) 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 $\gcd\binom{N}{t}!, k = 1$ and a $(N; t, k, v)$-$P$ array exists, then there is a $((\binom{N}{t} + 1)N, t, k^2, v)$-$P$ array.

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

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

Consider the array $B$ defined by

$$B = \begin{bmatrix}
  B_{0,0} & B_{0,1} & \cdots & B_{0,\binom{N}{t}} \\
  B_{1,0} & B_{1,1} & \cdots & B_{1,\binom{N}{t}} \\
  \vdots & \vdots & \ddots & \vdots \\
  B_{k-1,0} & B_{k-1,1} & \cdots & B_{k-1,\binom{N}{t}}
\end{bmatrix}$$

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

$$\begin{bmatrix}
  B_{i,0} & B_{i,1} & \cdots & B_{i,\binom{N}{t}}
\end{bmatrix}$$

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

$$\begin{bmatrix}
  B_{x,0} & B_{x,1} & \cdots & B_{x,\binom{N}{t}}
\end{bmatrix}$$

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

This theorem can be iterated:

Theorem 4.13. If $\gcd\binom{N}{t}!, k = 1$ and a $(N; t, k, v)$-$P$ array exists, then there is a $(N(\binom{N}{t} + 1)^\ell, t, k^{2\ell}, v)$-$P$ 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$(2v - 1)(q^3 - q) + v)4^{1/3}(2v - 1)^{2j}v$ if $v \equiv 0, 1 \mod 3$, and a CA$(2v - 1)(q^3 - q) + v)4^{1/3}(2v - 3)^{2j}v$ 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

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

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

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

and when $v = 4$,

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

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

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

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.

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

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 $P$ 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 $P$ that the test covers. We select a test that covers that largest number of pairs in $P$, add it to the collection of tests, and repeat this step until $P = \emptyset$; 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 $|P|/L$ 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 \( \sigma_i \)
for this factor so that the choice of \( \sigma_i \) 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 \( \pi : \{1, \ldots, k\} \to \{1, \ldots, k\} \).
Now assume that values \( \tau_1, \ldots, \tau_i \) have been selected for factors
\( f_{\pi(1)}, \ldots, f_{\pi(i)} \). Select a value \( \tau_{i+1} \) 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 perfor-
mance in practice, their earlier guarantee does not apply because the test selec-
tion 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 $i$th 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 $i$th, 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 $\delta$ 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:

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

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 $\sigma$ of factor $f_s$. Then rewrite the second summation as

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

We choose $\sigma$ to maximize $\sum_{1 \leq j \leq k} \rho_{i,s,\sigma}$. It follows that $\sum_{1 \leq j \leq k} \rho_{i,s,\sigma} \geq \sum_{1 \leq j \leq k} \delta_{i,j}$. We then fix factor $f_s$ to have value $\sigma$, set $v_s = 1$, and update
the local densities setting $\delta_{i,s}$ to be $\rho_{i,x,\sigma}$. 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 $\delta$. 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 $N$ times
start with an empty covering array $C$
while there are uncovered $t$-tuples in $C$
repeat $M$ times
start with an empty test (row) $R$
set $Best = 0$
while free factors remain
rank all free factors according to a factor selection criterion
among factors tied for best, select a subset $T$
using a first factor tie-break
among factors in $T$, select a single factor $f$
using a second factor tie-break
all possible values for $f$ in $R$ using a level selection criterion
among all best values for $f$, select a subset $V$
using a first level tie-break
among values in $V$, select value $v$
using a 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 $\text{Size} < \text{MinArray}$ rows, set $\text{MinArray} = \text{Size}$
and $\text{BestArray} = C$
end repeat
report $\text{BestArray}$

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, (v1, . . . , v_k)) by first generating an MCA(2, k − 1, (v1, . . . , 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 \( \Sigma \) 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 \( \Sigma \), 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 Östergård [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 $e^{-\left(\frac{c(S') - c(S)}{K_BT}\right)}$, 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 $\alpha$ (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 along 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 \( 3^4 = 81 \) configurations. This is perhaps reasonable, but if 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.

<table>
<thead>
<tr>
<th>Config:</th>
<th>Web Browser</th>
<th>Operating System</th>
<th>Connection Type</th>
<th>Printer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Netscape(0)</td>
<td>Window(0)</td>
<td>LAN(0)</td>
<td>Local(0)</td>
</tr>
<tr>
<td></td>
<td>IE(1)</td>
<td>Macintosh(1)</td>
<td>PPP(1)</td>
<td>Networked(1)</td>
</tr>
<tr>
<td></td>
<td>Other(2)</td>
<td>Linux(2)</td>
<td>ISDN(2)</td>
<td>Screen(2)</td>
</tr>
</tbody>
</table>

Table 3: Four components, each with 3 configurations

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Browser</th>
<th>OS</th>
<th>Connection</th>
<th>Printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NetScape</td>
<td>Windows</td>
<td>LAN</td>
<td>Local</td>
</tr>
<tr>
<td>2</td>
<td>NetScape</td>
<td>Linux</td>
<td>ISDN</td>
<td>Networked</td>
</tr>
<tr>
<td>3</td>
<td>NetScape</td>
<td>Macintosh</td>
<td>PPP</td>
<td>Screen</td>
</tr>
<tr>
<td>4</td>
<td>IE</td>
<td>Windows</td>
<td>ISDN</td>
<td>Screen</td>
</tr>
<tr>
<td>5</td>
<td>IE</td>
<td>Macintosh</td>
<td>LAN</td>
<td>Networked</td>
</tr>
<tr>
<td>6</td>
<td>IE</td>
<td>Linux</td>
<td>PPP</td>
<td>Local</td>
</tr>
<tr>
<td>7</td>
<td>Other</td>
<td>Windows</td>
<td>PPP</td>
<td>Networked</td>
</tr>
<tr>
<td>8</td>
<td>Other</td>
<td>Linux</td>
<td>LAN</td>
<td>Screen</td>
</tr>
<tr>
<td>9</td>
<td>Other</td>
<td>Macintosh</td>
<td>ISDN</td>
<td>Local</td>
</tr>
</tbody>
</table>

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 \( \binom{4}{2} \times 3^2 = 54 \) 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


