THE ART OF DISCRETE AND APPLIED MATHEMATICS

ISSN 2590-9770
The Art of Discrete and Applied Mathematics 1 (2018) #P2.03
https://doi.org/10.26493/2590-9770.1220.3a1
(Also available at http://adam-journal.eu)

Subspace restrictions and affine composition for covering perfect hash families*

Charles J. Colbourn †, Erin Lanus

Computing, Informatics, and Decision Systems Engineering, Arizona State University, PO Box 878809, Tempe, AZ, 85287-8809, U.S.A.

To Mario Gionfriddo on his seventieth birthday.

Received 19 November 2017, accepted 1 August 2018, published online 3 August 2018

Abstract

Covering perfect hash families provide a very compact representation of a useful family of covering arrays, leading to the best asymptotic upper bounds and fast, effective algorithms. Their compactness implies that an additional row in the hash family leads to many new rows in the covering array. In order to address this, subspace restrictions constrain covering perfect hash family so that a predictable set of many rows in the covering array can be removed without loss of coverage. Computing failure probabilities for random selections that must, or that need not, satisfy the restrictions, we identify a set of restrictions on which to focus. We use existing algorithms together with one novel method, affine composition, to accelerate the search. We report on a set of computational constructions for covering arrays to demonstrate that imposing restrictions often improves on previously known upper bounds.

Keywords: Covering array, covering perfect hash family, affine composition, subspace restriction.

Math. Subj. Class.: 05B40, 05B15, 51E26, 68R05

1 Introduction

We develop effective construction techniques for combinatorial arrays called covering perfect hash families, which form a compact representation of covering arrays. Covering arrays arise in numerous applications in which interactions among options or factors are

*Thanks to Ryan Dougherty, Kaushik Sarkar, and Violet Syrotiuk for useful discussions, and to two anonymous referees for helpful comments.

†Research of CJC was supported in part by the National Science Foundation under Grant No. 1421058.

E-mail addresses: colbourn@asu.edu (Charles J. Colbourn), elanus@asu.edu (Erin Lanus)

to be measured; they are used in, for example, software testing [12, 13], hardware testing [10, 20], design of composite materials [2], computational learning [1, 9], and biological networks [14]. Computational methods to construct covering arrays often encounter difficulties when the array has many rows, many columns, or both. To alleviate this concern, covering perfect hash families were introduced in [21] and shown to provide a succinct representation of a class of covering arrays. In [6] they were used to establish the best known asymptotic upper bound on the fewest rows in a covering array. Also in [6], effective and simple algorithms were examined for their construction.

Covering perfect hash families have proved instrumental in obtaining many sizes of covering arrays that are the best currently known. Despite the compactness of the representation that they provide, their use lessens but does not remove the computational burden. We propose and analyze a method, affine composition, to combine small covering perfect hash families to make larger ones; this extends the range of array sizes for which computational methods are feasible. Moreover, the very compactness of the representation severely limits the possible numbers of rows in the covering arrays produced. We develop a method using subspace restrictions to produce covering arrays that are guaranteed to have at least a specified number of duplicated rows, which can be removed without altering the coverage. This provides finer control on the number of rows in the covering array, and hence often improves upon the coarser use of covering perfect hash families without restrictions.

Both of these contribute to the construction of covering arrays with fewer rows than the best previously known, and hence to a reduction in testing and measurement cost when the covering arrays are applied. In order to develop these notions, we first provide formal definitions and background.

Let q be a prime power. Let $\mathbb{F}_q$ be the finite field of order q. Let $\mathcal{R}_{t,q} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{q^t-1}\}$ be the set of all (row) vectors of length t with entries from $\mathbb{F}_q$, and let $\mathcal{T}_{t,q}$ be the set of all column vectors of length t with entries from $\mathbb{F}_q$, not all 0. A vector $\mathbf{x} \in \mathcal{T}_{t,q}$ is a *permutation vector* [21].

Lemma 1.1 (see [21])**.** *Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$ be a set of vectors from $\mathcal{T}_{t,q}$. The array $A = (a_{ij})$ formed by setting a_{ij} to be the product of $\mathbf{r}_i$ and $\mathbf{x}_j$ is a $q^t \times t$ matrix in which every row is distinct if and only if the $t \times t$ matrix $X = [\mathbf{x}_1 \cdots \mathbf{x}_t]$ is nonsingular.*

When $\mu \in \mathbb{F}_q \setminus \{0\}$, substituting $\mu\mathbf{x}_i$ for $\mathbf{x}_i$ does not change the multiset of rows produced, just their order. Define $\langle \mathbf{x} \rangle = \{\mu\mathbf{x} : \mu \in \mathbb{F}_q, \mu \neq 0\}$. When $\mathbf{x}$ is not all 0, we can select as the representative of $\langle \mathbf{x} \rangle$ the unique vector whose first nonzero coordinate is the multiplicative identity element. Let $\mathcal{V}_{t,q}$ be the set of representatives of the column vectors in $\mathcal{T}_{t,q}$. Let $\mathcal{U}_{t,q}$ be the set of vectors in $\mathcal{V}_{t,q}$ whose first coordinate is not zero. Then $|\mathcal{V}_{t,q}| = \frac{q^t-1}{q-1} = \sum_{i=0}^{t-1} q^i$, and $|\mathcal{U}_{t,q}| = q^{t-1}$.

A *covering perfect hash family* $\mathrm{CPHF}(n; k, q, t)$ is an $n \times k$ array $C = (\mathbf{c}_{ij})$ with entries from $\mathcal{V}_{t,q}$ so that, for every set $\{\gamma_1, \ldots, \gamma_t\}$ of distinct column indices, there is at least one row index ρ of C for which $[\mathbf{c}_{\rho\gamma_1} \cdots \mathbf{c}_{\rho\gamma_t}]$ is nonsingular; call this a *covering t-set* and say that the t-set of columns is *covered*. It is a *Sherwood covering perfect hash family*, $\mathrm{SCPHF}(n; k, q, t)$, if in addition each entry is in $\mathcal{U}_{t,q}$.

Let N, t, k, and v be positive integers with $k \geq t \geq 2$ and $v \geq 2$. A *covering array* $\mathrm{CA}(N; t, k, v)$ is an $N \times k$ array A in which each entry is from a v-ary alphabet Σ, and for every $N \times t$ sub-array B of A and every $\mathbf{x} \in \Sigma^t$, there is a row of B that equals $\mathbf{x}$.

When k is a positive integer, $[k]$ denotes the set $\{1, \ldots, k\}$. A *t-way interaction* is $\{(c_i, a_i) : 1 \leq i \leq t\}$ where $c_i \in [k]$, $c_i \neq c_j$ for $i \neq j$, and $a_i \in \Sigma$. Such an interaction

is an assignment of values from Σ to t of the k columns. An $N \times k$ array A *covers* the interaction $\iota = \{(c_i, a_i) : 1 \le i \le t, c_i \in [k], c_i \ne c_j \text{ for } i \ne j, \text{ and } a_i \in \Sigma\}$ if there is a row r in A such that $A(r, c_i) = a_i$ for $1 \le i \le t$. When there is no such row in A, ι is *not* covered in A. Hence a $\mathrm{CA}(N; t, k, v)$ covers all the t-way interactions on k columns on an alphabet of v symbols.

Covering arrays are used extensively for interaction testing in complex engineered systems. The k columns represent *factors* and the v values are the *levels* of the factors. The N rows form a *test suite* (each row is a *test*); and the coverage of interactions among the factors is limited to the *strength* t.

Denote by $\mathrm{CAN}(t, k, v)$ the smallest value of N for which a $\mathrm{CA}(N; t, k, v)$ exists. This is a *covering array number*, and for essentially all applications the goal is to minimize it. Applications also require that actual covering arrays be generated, and hence the focus is on explicit, practical construction methods. Online tables at [5] give the least upper bound on $\mathrm{CAN}(t, k, v)$ *by an explicit construction* for $2 \le t \le 6$, $2 \le v \le 25$, and $k \le 10\,000$.

The connection between CPHFs and CAs is central in this paper, so we include a standard proof for the following correspondence.

Lemma 1.2 (see [21])**.**

1. *There exists a* $\mathrm{CA}(n(q^t - 1) + 1; t, k, q)$ *if* C *is a* $\mathrm{CPHF}(n; k, q, t)$*; and*
2. *there exists a* $\mathrm{CA}(n(q^t - q) + q; t, k, q)$ *if* C *is an* $\mathrm{SCPHF}(n; k, q, t)$.

Proof. Let C be the covering perfect hash family. Replace each entry $\mathbf{c}_{ij}$ of C by the column vector obtained by multiplying $\mathbf{c}_{ij}$ by each $\mathbf{r}_\ell \in \mathcal{R}_{t,q}$ in the specified order. By Lemma 1.1, this produces a $\mathrm{CA}(nq^t; t, k, q)$. The product of each $\mathbf{c}_{ij}$ with $(0, \ldots, 0) \in \mathcal{R}_{t,q}$ is 0, so the resulting array contains n rows that contain only 0 entries. Remove $n - 1$ of these rows to form the $\mathrm{CA}(n(q^t - 1) + 1; t, k, q)$. Now when $\mathbf{c}_{ij} \in \mathcal{U}_{t,q}$, multiplication by $(\sigma, 0, \ldots, 0) \in \mathcal{R}_{t,q}$ always yields σ. For each $\sigma \in \mathbb{F}_q$, remove $n - 1$ of the rows in which each entry is σ to form the $\mathrm{CA}(n(q^t - q) + q; t, k, q)$. □

In generating the covering array from the CPHF, it may happen that rows are generated that only cover t-way interactions that are also covered by other rows. Such a row is *redundant*, and could be removed. Indeed by tracking coverage as rows of the covering array are generated, one could avoid generating some of these redundant rows. More generally, a post-optimization method [17] may reveal or produce further redundant rows. Because the covering array is typically much larger than the covering perfect hash family, however, effort can be saved by determining in advance certain rows of the covering array that are guaranteed to be redundant, thereby avoiding their generation and subsequent elimination.

The simplest way to ensure that a row is redundant in the covering array generated is that it be identical to another row; then it is *replicated* or *repeated*. Lemma 1.2 already accounts for the redundancy of $n - 1$ rows for CPHFs, and of $(n - 1)q$ rows for SCPHFs, by noting that they are replicated. Our goal here is to restrict the CPHF in such a way that many more rows are guaranteed to be replicated, and so reduce the size of the covering array generated without having to analyze its coverage during and after its generation.

Whether restricted or not, CPHFs are needed to apply Lemma 1.2. Few general direct constructions are known [18, 21, 24]; most arise from computation. Computational methods for SCPHFs include backtracking [21] and tabu search [25]. In [6], CPHFs are shown to lead to the best known asymptotic results on the existence of covering arrays. Indeed

the probabilistic methods lead to two classes of efficient algorithms for constructing covering arrays for much larger parameters than had been earlier handled, and the best known bounds were improved on for a wide range of parameters as a result.

In [6], a means to restrict the CPHFs to ensure that certain rows are replicated is outlined, and applied for strength $t = 3$. In Section 2, we define restrictions and consider the effect of imposing various restrictions on the expectation that a t-set of columns is covered, in order to determine the types of restrictions that appear to be promising. In Section 3 we develop a recursive composition strategy to accelerate the computational search for restricted CPHFs. In Section 4 we report on new bounds obtained by subspace restrictions, at the same time updating some of the computational results from [6].

2 Subspace restrictions

We limit how entries are placed in a CPHF so that redundant rows are generated in the application of Lemma 1.2; these can be removed. We denote by $\mathcal{F}_{t,p}$ the set of all p-tuples of distinct entries from $\{0, \ldots, t-1\}$. A *subspace restriction* for n rows of *dimension* p and *replication* r is an r-tuple $(x_1, \ldots, x_r)$ of distinct entries from $\{1, \ldots, n\}$ and an r-tuple $(U_1, \ldots, U_r)$ for which each $U_i \in \mathcal{F}_{t,p}$.

Let $A = (a_{ij})$ be a CPHF$(n; k, q, t)$ in which each entry a_{ij} is a permutation vector of length t. Write $a_{ij\ell}$ for the ℓth entry of this vector. Let S be the subspace restriction (for n rows), given by $(x_1, \ldots, x_r)$ and $(U_1, \ldots, U_r)$. Denote by u_{ab} the element of U_a in position b. Then A *satisfies* or *meets* the restriction if, when $1 \le c, d \le r$, $a_{x_c,j,u_{c\ell}} = a_{x_d,j,u_{d\ell}}$ for all $1 \le j \le k$ and $1 \le \ell \le p$; for short, A is S*-restricted*.

Table 1: A CPHF$(4; 18, 3, 4)$. Each permutation vector $(h_0, h_1, h_2, h_3)^T$ is written as $h_0h_1h_2h_3$.

1020 1002 1211 1112 1122 1001 1002 1202 1111 1222 1100 1010 1101 1010 1220 1022 1122 1110
1020 1001 1212 1112 1120 1001 1000 1200 1111 1222 1100 1011 1102 1010 1220 1020 1121 1111
1022 1221 1212 1001 1100 1110 1012 1020 1111 1122 1112 1202 1010 1102 1200 1220 1222 1201
1021 1220 1202 1011 1121 1120 1002 1000 1110 1100 1122 1212 1020 1121 1222 1201 1211 1221

Table 1 shows an example. Verifying that this is a CPHF entails checking, for each 4-set of columns, that in at least one row the four permutation vectors are covering. For instance, checking that the second, fifth, sixth, and seventh columns have a covering 4-set in some row is the same as checking that at least one of the matrices

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

is nonsingular over $\mathbb{F}_3$. The first two are not because they repeat columns; the third is not because the sum of the first and fourth rows equals the second. But the fourth matrix is nonsingular, so we have verified that one of the 3 060 possible 4-sets of columns is covering (the diligent reader can verify the rest). Every permutation vector in the CPHF of Table 1 has a 1 in coordinate 0; hence the CPHF satisfies the restriction $(1, 2, 3, 4)$ with $U_1 = U_2 = U_3 = U_4 = \{0\}$. More is true. In the third and fourth rows, in each column the

two permutation vectors agree in the first two coordinates, and hence the CPHF satisfies the restriction $(3,4)$ with $U_3 = U_4 = \{0,1\}$. Moreover, in the first and second rows, in each column the two permutation vectors agree in the first *three* coordinates, and hence the CPHF satisfies the restriction $(1,2)$ with $U_1 = U_2 = \{0,1,2\}$.

When $\mathcal{S}$ is a set of restrictions for n rows, and a CPHF$(n;k,q,t)$ meets each $S \in \mathcal{S}$, it is *$\mathcal{S}$-restricted*. Now suppose that A is an $\mathcal{S}$-restricted CPHF$(n;k,q,t)$. Suppose that $S \in \mathcal{S}$ consists of $(x_1,\ldots,x_r)$ and $(U_1,\ldots,U_r)$, and that each U_i is a p-tuple. For each $1 \le i \le r$, in the q^t rows obtained from the expansion of row x_i, let E_i be the q^p rows that arise using evaluations on t-sets that are 0 on all elements not in U_i. Then $E_1 = \cdots = E_r$; that is, each row in E_1 is replicated $r-1$ further times in the expansion of A by Lemma 1.2, and hence $(r-1)q^p$ rows are redundant in the covering array generated (although Lemma 1.2 removes some of them already).

In the example of Table 1, we noted the presence of three restrictions. The restriction of dimension 1 and replication 4 results in $3 \cdot 3 = 9$ redundant rows. The restriction of dimension 2 results in $3^2 = 9$ redundant rows, of which three were already found to be redundant using the restriction of dimension 1. Finally the restriction of dimension 3 results in $3^3 = 27$ redundant rows; three of these were already found to be redundant using the restriction of dimension 1, while none among the remaining 24 are made redudant by the restriction of dimension 2. Hence for our example, we can ensure that at least $9+6+24 = 39$ rows are redundant; rather than getting $4 \cdot 3^4 - 3 = 321$ rows, we get 285.

A general example of a subspace restriction is straightforward. When the restriction S has $r = n$, $(x_1,\ldots,x_n) = (1,2,\ldots,n)$, and $U_i = (0)$ for $1 \le i \le n$, an S-restricted CPHF$(n;k,q,t)$ is precisely an SCPHF$(n;k,q,t)$. (The entry in coordinate 0 must be nonzero for the array to be a CPHF.) Enforcing restrictions of larger dimension can increase the redundancy [6], but enforcing too many restrictions or restrictions of too large a dimension might result in more rows or fewer columns.

Next we explore the effect of imposing a restriction on the expectation that an array is a CPHF. To simplify the presentation, we fix a strength t and a prime power q, and denote the product $\prod_{i=a}^{b} \frac{q^t - q^i}{q^t}$ by $\pi_{a,b}$. We consider a fixed set of t columns and ask for the probability that the columns are covered within r rows of the CPHF. In the *basic* process, we impose no restriction, choosing each of the coordinates of each of t permutation vectors independently and uniformly at random from $\mathbb{F}_q$ for each of the r rows. The probability that the chosen set of columns is not covered in the basic process is $(1-\pi_{0,t-1})^r$. In the *restricted process*, we first choose the p entries specified in the restriction independently and uniformly at random for one row, using the same choice for all. The remaining coordinates of each permutation vector are then chosen randomly. The probability that the chosen set of columns is not covered in the restricted process is $(1-\pi_{0,p-1}) + \pi_{0,p-1}\left[(1-\pi_{p,t-1})^r\right]$. The restricted process has a larger failure probability; indeed it is larger by an amount equal to

$$(1-\pi_{p,t-1})\left[1-\left[1-\pi_{0,t-1}\right]^{r-1}\right].$$

We consider two cases to examine the effect of the dimension and replication of a restriction. We tabulate failure probabilities within r rows when there is a restriction of size r and dimension p. (When $p = 0$, there is no restriction.) First we give failure probabilities for $q = 25$ and $t = 6$ (see Table 2).

Table 2: Failure probabilities with restrictions for $q = 25$ and $t = 6$.

$p \downarrow r \rightarrow$	2	3	4
0	$.1730551453 \times 10^{-2}$	$.7199076267 \times 10^{-4}$	$.2994808332 \times 10^{-5}$
1	$.1730555215 \times 10^{-2}$	$.7199483800 \times 10^{-4}$	$.2998903189 \times 10^{-5}$
2	$.1730649273 \times 10^{-2}$	$.7209672112 \times 10^{-4}$	$.3101274621 \times 10^{-5}$
3	$.1733000718 \times 10^{-2}$	$.7464379962 \times 10^{-4}$	$.5660560231 \times 10^{-5}$
4	$.1791790606 \times 10^{-2}$	$.1383210872 \times 10^{-3}$	$.6964257727 \times 10^{-4}$
5	$.3263893163 \times 10^{-2}$	$.1730452999 \times 10^{-2}$	$.1669115393 \times 10^{-2}$

As one might expect, restrictions increase the failure probability, but for those of 'low' dimension the increase is modest. Next we examine failure probabilities for $q = 3$ and $t = 5$ (see Table 3).

Table 3: Failure probabilities with restrictions for $q = 3$ and $t = 5$.

$p \downarrow r \rightarrow$	2	3	4
0	.1924749034	.0844425161	.0370465884
1	.1937767036	.0868835477	.0402353448
2	.1977309220	.0942611526	.0498358606
3	.2100498330	.1168880686	.0789332757
4	.2516261575	.1892616707	.1684735084

Naturally, because q is much smaller the failure probabilities are much larger than for $q = 25$, even though the strength here is lower.

Now suppose that our goal is to make $2q^p$ rows redundant in the generated covering array. We can choose a restriction of dimension p and replicate it three times. An alternative is to choose a second restriction of dimension p. If we choose the two restrictions so that they have no rows in common, we can replicate each twice to get the same number of redundant rows that we would get by selecting one with replication three. Which should we prefer? An easy calculation shows that the failure probability after four rows is lower with two restrictions of replication two, than one of replication three along with an unrestricted row. In general when two restrictions of dimension p with replications r_1 and r_2 restrict disjoint sets of rows, failure probabilities are minimized when r_1 and r_2 are as equal as possible. Because setting $r_2 = 1$ imposes no restriction at all, our quick example says that $r_1 = r_2 = 2$ is better than $r_1 = 3$ and $r_2 = 1$. Hence we strive to choose many restrictions with replication two.

This ignores the fact that there may be too few rows to specify the desired number of restrictions. However, the requirement that the restrictions share no rows is too severe. For the analysis, we only require that every two restrictions sharing a row select different coordinates within that row. This ensures that the rows made redundant by one are not those made redundant by the other (with the exception of the all zero row, which is redundant in every case). Moreover, in an analysis of failure probabilities the effects of two restrictions are independent, because the entries in each are chosen independently of one another.

Consider restrictions of dimensions d_1 and d_2 that share s coordinates within a row. The impact is that the q^{d_1} rows made redundant by the first and the q^{d_2} rows made redundant by

the second have q^s rows in common (a subspace). Consequently, fewer rows are redundant than if the two restrictions acted independently. When restrictions share the same rows, even when on different coordinates, the effect on the failure probability can be dramatic: When $t = 5$, for example, a restriction of dimension 4 on $(0, 1, 2, 3)$ and a restriction of dimension 2 on $(3, 4)$, if replicated on the same two rows, are in fact a restriction of dimension 5, forcing a replicated row in the CPHF itself. Nevertheless, when s is small, the double coverage is also small, and there are cases in which permitting sharing is sensible.

3 Affine composition

Algorithms employed for unrestricted CPHFs from [6, 21, 25] extend in a natural way to search for restricted CPHFs, so we do not repeat them here. Instead we describe an additional approach. When one wants to make a covering array with 'many' columns, computational methods either require too much time, or yield an array with many more rows than anticipated. Yet the same methods can yield arrays with 'few' rows quickly when the number of columns is small. To take advantage of the efficacy of computational methods for smaller arrays, and still construct larger ones, recursive methods have been developed to use small arrays in a cut–and–paste method; for example, see [3, 4, 7, 8, 15, 16, 19]. Roughly speaking, a cut–and–paste (or composition) method starts with an array on k columns, forms m copies of the array written side by side to obtain mk columns, and then uses further rows to cover the as-yet-uncovered interactions.

Writing two copies of a $\mathrm{CPHF}(n; k, q, t)$ side by side is equivalent to duplicating each column. But then in the $(n \times 2k)$ array produced, every choice of t columns containing a duplicate leads to a singular matrix for *every* row of the CPHF. Hence exactly $\sum_{s=1}^{\lfloor t/2 \rfloor} \binom{k}{s}\binom{k-s}{t-2s}2^{t-2s}$ sets of t columns are noncovering. Although the uncovered t-sets are easily counted and characterized, there are many of them.

Let A be a $\mathrm{CPHF}(n; k, q, t)$. Consider the effect of multiplying coordinate c of every permutation vector in row r by a nonzero element $\mu_{r,c}$ of the finite field. This does not affect the (non)singularity of any of the $t \times t$ matrices that determine coverage. Indeed we can apply different nonzero multipliers $\mu_{r,c}$ for every $1 \le r \le n$ and $1 \le c \le t-1$ to form a new array B that is again a $\mathrm{CPHF}(n; k, q, t)$. Then we no longer simply duplicate columns, we can change them in a controlled way.

Similarly for any row we can choose a field element m and any two different coordinates c and d; then for each column, add m times the entry in coordinate c to the entry in coordinate d. Again, this does not affect the (non)singularity of any of the $t \times t$ matrices that determine coverage. This is most easily accomplished using an SCPHF, whose permutation vectors always have first coordinate equal to 1. Then setting $c = 0$, this amounts to m being an *adder*, which can be added to the entry of coordinate d of every permutation vector in the row. We can choose adders $\alpha_{r,c}$ for every $1 \le r \le n$ and $1 \le c \le t-1$, provided that the array is an SCPHF.

In general, given an $\mathrm{SCPHF}(n; k, q, t)$ A, multipliers $\mu_{r,c}$ and adders $\alpha_{r,c}$ for $1 \le r \le n$ and $1 \le c \le t-1$, we can create a new array by, for each row r, for every column, replacing the entry x in coordinate c by $\mu_{r,c}x + \alpha_{r.c}$, with arithmetic in the field. Hence each coordinate of each row undergoes an affine transformation. No matter how this is done, the resulting array is an $\mathrm{SCPHF}(n; k, q, t)$. There are $(q(q-1))^{n(t-1)}$ ways to choose these multipliers and adders. *Affine composition* applied to an $\mathrm{SCPHF}(n; k, q, t)$ A_0 selects $m-1$ arrays $A_1, \ldots, A_{m-1}$, each obtained by affine transformations of A.

When affine composition is applied to A_0 to form $A_1, \ldots, A_{m-1}$, not only is each an SCPHF when A_0 is, but each A_i meets *all* of the restrictions that A_0 does. In fact, although $[A_0\ A_1\ \cdots\ A_{m-1}]$ need not be an SCPHF because certain t-sets of columns are not covered, it does meet all restrictions that A_0 does. The question remains: Which affine composition should we apply in order to leave the fewest, or to leave a particular set, of uncovered t-sets of columns?

We consider various SCPHF$(n; k, q, t)$s, taking $m = 2$. Because considering all $(q(q-1))^{n(t-1)}$ affine transformations is too time-consuming, we adopt a greedy strategy. We consider each row in turn, and determine for some or all of the $(q(q-1))^{t-1}$ affine transformations how many uncovered t-sets of columns having at least one column from the original and from the transformed copy remain, if this transformation is applied in the current row. We choose transformations that lead to a smallest number of t-sets yet to cover. After the last row is processed, this smallest number is the number that must be dealt with in additional rows not produced in the composition. Tie-breaking is carried out by choosing the lexicographically first, so the method as implemented is deterministic. Random tie-breaking, or selection methods more clever than greedy, might result in further reductions.

Table 4: Numbers of noncovering t-sets after affine compositions.

$n; k, v, t$	$\times 1 + 0$	$\times 1 + \alpha$	$\times \mu + 0$	$\times \mu + \alpha$	$\times 1 + \alpha_c$	$\times \mu_c + 0$	$\times \mu_c + \alpha_c$
2;12,4,4	2706	548	678	535	486	619	463
3;21,4,4	16170	1876	2015	1739	1572	1964	1452
4;31,4,4	54405	3239	2941	2783	2621	2594	2099
5;45,4,4	171270	4584	4176	3698	4228	3765	3312
6;59,4,4	391819	5161	3911		4835	3546	
2;15,5,4	5565	1016	871	848	915	871	803
3;24,5,4	24564	1795	1611	1409	1634	1442	1236
4;40,5,4	119340	4001	3299	3037	3441	2869	2432
5;59,5,4	391819	5253	3635		4938	3337	
6;88,5,4	1320660	8451	4689		7964	4367	
2;18,7,4	9945	1109	985	940	1081	972	855
3;34,7,4	72369	2883	2421		2619	2202	
4;57,7,4	352716	4627	3140		4386	2983	
2;20,8,4	13870	1296	1242	1117	1224	1172	1069
3;38,8,4	101935	2979	2499		2737	2249	
4;67,8,4	577071	5461	3441		5191	3278	
2;22,9,4	18711	1606	1448	1330	1539	1364	1251
2;11,4,5	11550	2280	1763	1500	2070	1618	1389
3;15,4,5	46410	4856	2706	2350	4550	2706	
2;10,3,6	25320	7984	5081	4573	7524	4766	4274

In Table 4, we report results for the number of uncovered t-sets of columns after affine composition when the allowed affine transformations are limited in various ways. The column $(\times 1 + 0)$ gives numbers when every multiplier is 1 and every adder 0. This, of

course, is another way to say that the second array is an exact copy of the first. The column $(\times 1 + \alpha)$ always uses multiplier 1, and selects the *same adder* for every coordinate in the row. The column $(\times \mu + 0)$ always uses adder 0, and selects the *same multiplier* for every coordinate in the row. The column $(\times \mu + \alpha)$ selects the same adder, and the same multiplier, for every coordinate in the row. The column $(\times 1 + \alpha_c)$ always uses multiplier 1, and considers all q^{t-1} ways to select adders for the coordinates in the row. The column $(\times \mu_c + 0)$ always uses adder 0, and considers all $(q-1)^{t-1}$ ways to select multipliers for the coordinates in the row. The column $(\times \mu_c + \alpha_c)$ considers all $(q(q-1))^{t-1}$ ways to select adders and multipliers for the coordinates in the row.

Consider an arbitrary permutation vector in $\mathcal{U}_{t,q}$ and its images under the $(q(q-1))^{t-1}$ affine transformations. Every permutation vector in $\mathcal{U}_{t,q}$ appears precisely $(q-1)^{t-1}$ times among these images. Hence when $q(t-1)$ is large compared to the number of columns, the probability that duplicate columns arise is reduced. Indeed simply making a copy (without any nontrivial transformation) leaves far more uncovered sets of columns than even very restricted sets of affine transformations do. This is part of the explanation for the effectiveness of applying affine transformations.

Choosing the best affine transformation to apply seems impractical; indeed we did not fill in the blank entries in Table 4 because even the greedy strategy has either too many options to consider or too large an array to check repeatedly. Of course, it would be better to determine the most effective transformations without having to conduct a large search, but because this depends heavily on the structure of the SCPHF, we know of no way to do this. Hence we choose them randomly. The expected number of t-sets of columns that remain uncovered after affine composition depends not only on the parameters of the SCPHF and the number of copies made, but also on the structure of the SCPHF. Indeed it depends on the number of rows in which ℓ columns of the SCPHF are linearly independent, for every way to choose ℓ columns with $2 \le \ell < t$. Consider, for example, the situation with strength $t = 3$, and consider two columns. For any row with identical permutation vectors in these two columns, there is no possibility for affine composition to cover a 3-set of columns containing this pair. Then the probability that randomly chosen affine transformations succeed on a particular triple of columns depends upon in how many rows of the CPHF the two have identical entries.

In general, we wish to maximize the number of sets of ℓ columns that are linearly independent for all $\ell \le t$. Because the definition of a CPHF does not include such a strong condition, there can be SCPHFs on which affine composition yields a poor result.

After any affine composition is carried out, we anticipate that not all t-sets of columns will be covered (although within each of the m copies formed. all t-sets of columns are covered). Hence affine composition is not a means to avoid doing any computation, but rather a means to reduce to a substantially smaller problem.

4 Computations with subspace restrictions

Subspace restrictions give a natural way for redundant rows to be formed in the expansion of a CPHF into a covering array. For this to be worthwhile, the restricted CPHF must have more columns than does the unrestricted CPHF with one fewer row. (Otherwise the resulting covering array would in general have more rows without increasing the number of columns.) In the results to follow, we find that restrictions not only reduce the number of rows needed, but in many cases do not reduce the achievable number of columns.

We construct SCPHFs satisfying specific sets of restrictions. We always enforce the restriction $(x_1, \ldots, x_n) = (1, 2, \ldots, n)$, and $U_i = (0)$ for $1 \leq i \leq n$; this means we are talking about SCPHFs and not general CPHFs, and hence can employ affine composition as described. We also enforce restrictions of higher dimension. In particular, a restriction on two distinct rows i and j with $U_i = U_j = (0, 1, \ldots, p-1)$ is denoted by $\langle p \rangle_{i,j}$ or, more simply, $\langle p \rangle$. When multiple restrictions of this type are enforced, we require that they refer to disjoint sets of rows of the SCPHF; provided that they do, we can simply list the dimensions of the restrictions imposed. We use exponential notation, so that $\langle p \rangle^a$ requires that the restriction $\langle p \rangle$ be imposed on a disjoint pairs of rows.

In order to accelerate the computation, we use affine composition in some cases to make a 'large' fraction of the CPHF. We employ column resampling, random extension, and conditional expectation algorithms from [6]. The adaptation of each to incorporate any number of restrictions is straightforward. There are more intensive search techniques that yield more accurate results, but the simpler methods can be effectively applied for somewhat larger numbers of columns and symbols. So when we report that a certain number of columns can be realized, we fully expect that a more intensive search can find a solution with more columns (and, in some cases, has). Despite this, certain trends are evident, as one expects based on the failure probabilities.

Given a prime power q, number n of rows, strength t, and set $\mathcal{S}$ of restrictions, we tabulate the largest number k of columns found in an $\mathcal{S}$-restricted SCPHF$(n; k, q, t)$. When $q = 23$ and $t = 4$, rows in Table 5 indicate the value of n; columns indicate the restrictions enforced. Here C reports results for CPHFs, while $-$ reports results for SCPHFs (that is, no restrictions beyond the basic one).

Table 5: Improvements (shown in bold) on known covering array numbers when $t = 4$ and $q = 23$.

n	C	$-$	$\langle 2 \rangle$	$\langle 2 \rangle^2$	$\langle 3 \rangle$	$\langle 3 \rangle \langle 2 \rangle$	$\langle 3 \rangle^2$	$\langle 3 \rangle^3$
2	39	39	**39**		**30**			
3	98	98	**98**		**85**			
4	**250**	**245**	**240**	**227**	**196**	**194**	**170**	
5	**603**	**600**	**585**	**569**	497	484	389	
6	**1461**	**1365**	**1333**	**1192**	**1184**	**1174**	**1003**	**874**

Entries shown in bold are those that improve upon the previously best known size of a covering array for these parameters (all from [6]). It may be disappointing that by imposing three $\langle 3 \rangle$ restrictions, our methods construct only 874 columns rather than 1365. However, one must bear in mind that these restrictions force (at least) 36 432 redundant rows in the covering array. This accomplishes what we set out to do. Although we may have fewer columns, we generate fewer rows. Table 6 shows similar results for $t = 5$ and $q = 5$, using a more extensive set of restrictions. Improvements are again frequent.

Table 7 gives a complete set of results for strength $t = 4$ with $4 \leq q \leq 25$. (Henceforth we do not display every improvement for a covering array number in bold; see [5] to determine when an array is the best known.) Table 8 displays the results for $t = 5$ with $4 \leq q \leq 25$ having two, three, and four rows, while Table 9 displays results with five and six rows, and Table 10 gives results having seven or more rows.

Table 6: Improvements (shown in bold) on known covering array numbers when $t = 5$ and $q = 5$.

n	C	–	$\langle 2\rangle$	$\langle 2\rangle^2$	$\langle 3\rangle$	$\langle 3\rangle\langle 2\rangle$	$\langle 3\rangle^2$	$\langle 4\rangle$	$\langle 4\rangle\langle 2\rangle$	$\langle 4\rangle\langle 3\rangle$	$\langle 4\rangle^2$	$\langle 4\rangle^3$
2	12	12	12		12			**12**				
3	17	17	**17**		16			**16**				
4	24	**24**	23	23	**23**	22	**22**	21	21	21	**21**	
5	32	32	32	32	**32**	**31**	30	30	**30**	28	**28**	
6	44	44	**44**	41	42	40	40	**42**	39	39	**39**	**35**
7	59	59	**59**	58	**58**	**57**	**56**	**54**	52	**52**	**51**	**45**
8	81	81	**81**	**79**	**78**	**76**	**75**	**73**	**70**	**68**	**67**	**62**
9	107	107	**107**	106	106	**106**	**103**	**95**	93	**93**	**91**	**84**
10	**143**	142	**142**	141	**141**	**139**	**138**	**131**	**128**	**126**	**119**	**110**
11	196	196	196	**196**	191	**191**	**187**	**175**	**174**	**171**	**162**	**152**
12	266	266	266	**266**	**262**	**261**	**255**	242	**242**	**239**	**220**	**200**
13	346	346	**346**	**340**	333	333	327	333	**333**	**327**	**286**	**265**

Table 7: Number of columns found for various restrictions and strength four.

q	n	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 3\rangle^2$	$\langle 3\rangle^3$
4	2	13	12	11	9		
4	3	21	21	20	17		
4	4	32	31	31	27	23	
4	5	45	45	43	38	34	
5	2	15	15	14	11		
5	3	26	24	24	21		
5	4	41	40	40	34	30	
5	5	60	59	58	54	48	
5	6	90	89	88	83	74	65
5	7	141	138	132	125	110	103
5	8	206	205	198	172	157	157
5	9	306	301	286	265	232	214
5	10	465	457	434	392	355	314
5	11	700	680	657	559	502	480
5	12	1102	1013	1012	820	751	712
5	13	1607	1431	1425	1264	1127	1090
7	2	18	18	17	14		
7	3	34	34	33	30		
7	4	57	57	56	51	45	
7	5	101	99	98	89	79	
7	6	169	169	166	154	134	119
7	7	282	277	275	242	209	186
7	8	475	475	457	394	364	311
7	9	814	764	742	631	551	527
7	10	1338	1336	1334	1064	956	897
8	2	20	20	19	15		
8	3	38	38	37	33		
8	4	67	67	67	60	53	
8	5	122	121	118	109	94	
8	6	220	218	214	177	154	148
8	7	379	370	359	316	277	239
8	8	657	657	611	565	445	415
8	9	1161	1159	1155	918	808	732
9	2	22	22	21	17		
9	3	43	43	41	37		
9	4	79	78	76	69	60	
9	5	148	147	143	133	114	
9	6	270	276	275	223	188	183
9	7	487	484	462	405	356	309
9	8	896	847	845	681	596	556
9	9	1621	1503	1475	1243	1106	1025

q	n	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 3\rangle^2$	$\langle 3\rangle^3$
11	2	24	24	24	20		
11	3	49	49	47	43		
11	4	100	99	96	89	80	
11	5	207	202	199	166	141	
11	6	388	388	374	321	286	241
11	7	745	735	734	572	494	456
11	8	1508	1507	1474	1112	1009	934
13	2	28	27	27	21		
13	3	57	57	55	51		
13	4	124	124	121	106	94	
13	5	260	265	262	214	177	
13	6	524	524	494	431	371	324
13	7	1102	1027	1002	839	721	654
16	2	31	31	31	24		
16	3	71	71	69	61		
16	4	159	158	157	140	120	
16	5	357	357	344	293	246	
16	6	778	763	745	579	506	468
16	7	1666	1578	1527	1260	1102	1040
17	2	34	34	32	25		
17	3	75	74	71	63		
17	4	175	174	171	152	122	
17	5	382	380	366	320	258	
17	6	873	822	818	654	550	517
17	7	1778	1743			1323	1193
19	2	35	35	35	28		
19	3	83	83	79	72		
19	4	203	199	197	167	136	
19	5	455	455	443	381	301	
19	6	1056	1001	934	802	692	642
23	2	39	39	39	30		
23	3	98	98	98	85		
23	4	250	245	240	196	170	
23	5	603	600	585	497	389	
23	6	1461	1365	1333	1174	1003	874
25	2	42	42	41	33		
25	3	107	107	104	90		
25	4	277	274	265	215	187	
25	5	694	688	668	529	443	
25	6	1706	1584	1575	1286	1115	1037

Table 8: Number of columns found for various restrictions and strength five, $n \in \{2, 3, 4\}$.

	$n=2$					$n=3$					$n=4$					
q	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\langle 4\rangle^2$
3	12	11	10	10	8	13	13	12	12	12	16	16	15	15	15	14
4	11	11	11	11	10	15	15	15	15	14	20	20	20	19	18	17
5	12	12	12	12	12	17	17	17	16	16	24	24	23	23	21	21
7	14	14	14	14	12	22	21	21	21	19	31	31	31	31	29	27
8	15	15	15	15	13	23	23	23	23	22	36	36	36	35	33	30
9	16	16	15	15	14	25	25	24	24	22	39	39	39	39	36	33
11	17	17	17	16	15	29	29	28	28	26	47	47	47	47	42	38
13	19	19	18	18	16	31	31	31	31	29	55	55	55	53	48	44
16	20	20	20	20	18	38	38	38	36	33	66	66	64	64	56	53
17	21	20	20	20	18	39	39	39	37	34	70	70	70	68	61	56
19	22	22	22	22	19	41	41	40	40	37	78	78	77	75	68	60
23	24	24	24	24	21	48	48	47	46	43	90	90	88	87	81	72
25	25	25	24	24	21	49	49	49	49	42	98	97	96	94	88	76

Table 9: Number of columns found for various restrictions and strength five, $n \in \{5, 6\}$.

	$n=5$						$n=6$						
q	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\langle 4\rangle^2$	$\subset$	$-$	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\langle 4\rangle^2$	$\langle 4\rangle^3$
3	19	19	19	19	18	17	23	23	23	22	22	21	19
4	26	26	26	25	24	23	34	34	32	32	31	28	27
5	32	32	32	32	30	28	44	44	44	42	42	39	35
7	47	47	47	47	43	39	69	69	68	67	61	59	50
8	55	55	55	55	50	45	83	83	82	82	74	67	60
9	62	62	61	60	55	51	95	95	92	91	88	77	73
11	79	79	79	77	69	61	127	125	125	120	109	100	90
13	93	93	92	88	84	75	157	157	156	153	140	120	106
16	119	119	119	111	103	92	210	207	206	197	185	165	146
17	128	128	128	120	112	97	228	223	219	217	199	181	157
19	143	143	143	136	126	107	269	262	262	258	233	207	180
23	181	178	175	171	155	136	345	344	342	332	299	259	228
25	197	197	194	184	166	147	406	406	391	386	332	288	249

Table 10: Number of columns found for various restrictions and strength five, $n \geq 7$.

q	n	C	–	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\langle 4\rangle^2$	$\langle 4\rangle^3$
4	7	42	42	41	41	41	38	33
5	7	59	59	59	58	54	51	45
5	8	81	81	81	78	73	67	62
5	9	107	107	107	106	95	91	84
5	10	143	142	142	141	131	119	110
5	11	196	196	196	191	175	162	152
5	12	266	266	266	262	242	220	200
5	13	346	346	346	333	333	288	265
5	14	458	458	455	447	435	394	363
5	15	609	609	606	603	569	527	484
5	16				779	721	672	647
7	7	101	101	99	96	93	82	77
7	8	149	149	148	140	132	121	112
7	9	211	209	205	201	195	179	159
7	10	318	312	312	315	284	253	230
7	11	455	454	451	447	418	361	341
7	12	661	661	660	628	592	546	495
7	13						702	679
8	7	123	123	122	116	109	99	90
8	8	186	185	184	183	163	150	134
8	9	283	283	277	278	252	229	210
8	10	430	429	427	417	371	343	320
8	11	622	622	604	595	569	510	473
8	12					812	749	675

q	n	C	–	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\langle 4\rangle^2$	$\langle 4\rangle^3$
9	7	151	149	149	147	133	118	110
9	8	231	229	229	224	214	184	167
9	9	374	374	355	353	324	283	254
9	10	568	568	546	542	483	445	405
9	11				818	755	690	629
11	7	209	209	209	196	182	161	147
11	8	342	340	332	328	289	248	228
11	9	533	529	524	505	472	425	389
11	10				827	730	680	614
13	7	272	272	271	261	237	209	186
13	8	454	452	450	442	379	349	324
13	9	767	767	764	750	653	567	523
16	7	379	379	376	360	336	284	248
16	8	656	656	650	641	582	515	454
16	9							766
17	7	408	407	407	400	358	309	278
17	8	721	721	718	663	620	555	480
19	7	473	470	466	461	446	373	343
19	8			851	838	723	671	597
23	7	656	656	654	648	564	493	430
23	8							785
25	7	712	712	709	701	636	568	483

We illustrate a further useful application of restrictions using strength $t = 6$. Every time a restriction $\langle t-1 \rangle$ is enforced, $q^{t-1} - q$ additional rows become redundant in the covering array. By enforcing restrictions $\langle t-1 \rangle^q$, $q^t - q^2$ rows are redundant. On the other hand, each row of the SCPHF would employ only a slightly larger number of rows, $q^t - q$. To take advantage of this, compare the four situations with $q = 3$ and $t = 6$ in Table 11. The SCPHF produced is permitted to have n rows, but must satisfy the specified number of $\langle t-1 \rangle$ restrictions. The covering array generated has N rows; notice how close the values of N are. Which should we prefer? By computing failure probabilities after n rows subject to the restrictions, one finds that enforcing more restrictions gives lower failure probability, primarily because more restrictions allow more rows.

Table 11: Four restricted SCPHF$(n; k, 3, 6)$s that yield covering arrays with similar numbers of rows.

n	N	$\#\langle 5 \rangle$	Failure	k
15	10893	0	.0000044079	57
16	10899	3	.0000043357	57
17	10905	6	.0000042646	58
18	10911	9	.0000041948	61

We applied the simple computational methods to each; the computed failure probabilities suggest that we should choose more restrictions, and the largest number of columns produced agrees. This is not an isolated example. In Table 12 we report on similar computations for $q = 3$ and $t = 6$ with different numbers of rows and restrictions. In order to read this effectively, an entry ought to be compared with the one that is three columns to the left and one row above, because the resulting covering arrays have comparable number of rows. We reiterate that we have not found the maximum number of columns in general; indeed we may be very far from it. Nevertheless, it is important that when enough restrictions (and the right ones) are enforced, there is a possibility of improving on an unrestricted SCPHF with fewer rows.

Improvements arise frequently when $t = 6$ for larger values of q as well; we summarize the results in Table 13.

Naturally, other search techniques can be applied to make improvements, and other restrictions may prove useful in the construction of covering arrays. What we have shown is that worthwhile restrictions can often be enforced with little penalty in failure probability or in number of columns generated. In order to avoid substantial computaion, it would be of substantial interest to develop further geometric constructions of CPHFs using finite projective or affine spaces, particularly with an eye to which nontrivial restrictions can be enforced.

5 Concluding remarks

Two extensions of research on covering perfect hash families have been developed here. The first provides a flexible recursive technique for making large CPHFs from smaller ones; the remarkable feature of this approach is that rather than simply juxtaposing copies of smaller arrays, each copy can be transformed by affine mappings in order to enhance the coverage obtained. We have demonstrated that different affine transformations can have a

Table 12: Number of columns found in an $\mathrm{SCPHF}(n;k,3,6)$ satisfying $\langle 5\rangle^{\ell}$.

		Number ℓ of $\langle 5\rangle$ restrictions										
n	C	0	1	2	3	4	5	6	7	8	9	10
2	10	10	8									
3	11	11	11									
4	13	13	12									
5	15	16	15	14								
6	18	18	17	16	16							
7	20	21	19	19	18							
8	23	23	21	21	20	19						
9	27	27	24	24	24	23						
10	31	30	28	27	27	25	24					
11	34	33	31	30	30	29	27					
12	39	38	35	35	33	33	31	31				
13	44	43	40	40	37	39	36	35				
14	50	50	50	47	42	44	42	39	39			
15	57	57	54	51	50	48	47	46	44			
16	66	66	62	59	57	55	54	52	51	49		
17	73	71	70	68	66	64	62	58	58	53		
18	85	82	82	77	75	74	72	68	64	63	61	
19	99	98	93	87	86	82	77	73	73	70	69	
20	108	108			101	96	92	85	82	78	77	72
21	123	122						102	97	93	88	85
22	144	142									102	97

Table 13: Number of columns found for various restrictions and strength six.

q	n	C	–	$\langle 2\rangle$	$\langle 3\rangle$	$\langle 4\rangle$	$\langle 5\rangle$	$\langle 5\rangle^2$	$\langle 5\rangle^3$
4	2	11	11	11	11	10	9		
4	3	13	13	13	13	13	12		
4	4	16	16	16	16	15	15	14	
4	5	20	19	19	19	19	18	17	
4	6	23	23	23	23	23	22	21	19
4	7	28	28	27	27	26	26	24	22
4	8	33	33	33	33	32	31	31	28
4	9	40	40	40	39	38	38	36	35
4	10	49	49	48	48	46	46	43	43
4	11	58	58	57	57	54	54	53	50
4	12	73	73	73	72	69	66	63	60
4	13	90	86	86	84	82	78	76	74
4	14	107	107	107	103	100	96	93	89
4	15	128	128	128	127	127	119	112	107
4	16	157	156	153	150	146	141	134	131
5	2	11	11	11	11	11	10		
5	3	15	15	14	14	14	13		
5	4	19	19	18	18	18	17	16	
5	5	23	23	23	23	23	22	20	
5	6	30	30	28	27	27	26	25	24
5	7	36	36	35	35	34	33	32	29
5	8	46	45	45	44	44	43	38	37
5	9	59	59	56	56	54	53	48	47
5	10	75	74	74	72	69	68	64	59
5	11	93	93	93	92	90	85	82	76
5	12	121	121	121	120	118	111	103	95
5	13	156	156	149	149	146	142	131	121
7	2	13	13	13	12	12	11		
7	3	17	17	17	17	17	15		
7	4	23	23	22	22	22	21	20	
7	5	31	31	31	31	29	27	27	
7	6	41	41	40	39	38	38	36	33
7	7	55	55	55	52	51	49	46	44
7	8	76	76	76	75	70	68	63	58
7	9	103	103	103	101	95	93	85	80
7	10	140	140	138	135	132	127	116	106
8	2	13	13	13	13	13	12		
8	3	18	18	18	18	17	17		
8	4	26	26	26	25	25	23	22	
8	5	33	33	33	32	32	31	29	
8	6	47	47	47	46	45	42	39	36
8	7	65	65	65	65	61	58	54	48
8	8	94	94	94	91	88	83	77	69
8	9	130	130	130	128	127	118	109	102
9	2	14	14	13	13	13	12		
9	3	20	20	19	19	19	18		
9	4	27	26	26	26	25	24	23	
9	5	38	38	37	37	36	35	33	
9	6	55	55	53	53	51	49	44	39
9	7	78	76	75	75	72	69	64	58
9	8	112	112	112	108	107	99	92	83
11	2	15	15	14	14	14	13		
11	3	22	22	22	22	21	19		
11	4	31	31	31	30	30	28	26	
11	5	46	46	45	44	43	41	35	
11	6	68	67	66	66	61	60	54	47
11	7	100	100	100	97	95	88	83	74
11	8	150	150	147	145	141	132	121	107
13	2	16	16	16	16	15	14		
13	3	23	23	22	22	22	20		
13	4	35	34	34	33	33	30	29	
13	5	54	54	52	52	51	49	41	
13	6	82	81	81	81	77	71	65	59
13	7	124	124	124	124	121	111	102	91
16	2	16	16	16	16	16	15		
16	3	26	26	25	25	25	22		
16	4	41	40	39	39	39	35	33	
16	5	66	66	63	63	58	56	51	
16	6	104	104	104	103	98	89	82	73
16	7	164	163	159	156	150	141	127	113
17	2	17	17	16	16	16	15		
17	3	27	27	25	25	25	23		
17	4	42	42	41	41	41	38	35	
17	5	66	66	65	65	62	57	53	
17	6	108	108	108	106	105	96	86	77
17	7	178	177	176	171	167	156	139	125
19	3	29	29	27	27	27	25		
19	4	46	45	45	45	45	41	37	
19	5	76	74	74	74	70	65	59	
19	6	124	124	124	122	118	110	100	87
23	3	31	31	30	29	29	26		
23	4	52	52	51	51	51	45	41	
23	5	87	87	87	87	85	78	70	
23	6	148	148	146	145	145	133	119	106
25	3	31	31	31	31	30	28		
25	4	56	56	54	54	54	49	43	
25	5	96	95	95	93	91	82	76	
25	6	169	169	166	163	161	147	132	116

dramatic effect on the composition, but theoretical guarantees for the observed improvements are needed.

The second extension provides finer control on the number of rows obtained in the resulting covering array, using the notion of subspace restrictions. Extensive computations demonstrate the effectiveness of such restrictions on reducing the number of rows in the covering arrays produced, over a wide range of parameters. Although the algorithmic methods used are relatively fast, we have repeatedly remarked that there is no reason to believe that they yield sizes that are besr possible in general. More computationally expensive methods such as backtracking [21] and tabu search [25], when feasible, may improve upon the results obtained. Indeed, after this paper was completed, simulated annealing and post-optimization have been used to obtain more accurate sizes for a smaller range of parameters [11, 23].

Certainly further computation will yield further improvements, and the use of subspace restrictions and affine composition accelerate these computations. However, we anticipate that direct constructions (extending those in [18, 22, 24]) would be of most interest, particularly if they accommodate a variety of subspace restrictions.

References

[1] N. H. Bshouty and A. Costa, Exact learning of juntas from membership queries, in: R. Ortner, H. U. Simon and S. Zilles (eds.), *Algorithmic Learning Theory*, Springer, Cham, volume 9925 of *Lecture Notes in Artificial Intelligence*, 2016 pp. 115–129, doi:10.1007/978-3-319-46379-7_8, proceedings of the 27th International Conference (ALT 2016) held in Bari, October 19 – 21, 2016.

[2] J. N. Cawse, *Experimental design for combinatorial and high throughput materials development*, Technical Report 2002GRC253, GE Global Research, November 2002.

[3] M. Chateauneuf and D. L. Kreher, On the state of strength-three covering arrays, *J. Combin. Des.* **10** (2002), 217–238, doi:10.1002/jcd.10002.

[4] M. B. Cohen, C. J. Colbourn and A. C. H. Ling, Constructing strength three covering arrays with augmented annealing, *Discrete Math.* **308** (2008), 2709–2722, doi:10.1016/j.disc.2006.06.036.

[5] C. J. Colbourn, Covering array tables: $2 \leq v \leq 25$, $2 \leq t \leq 6$, $t \leq k \leq 10000$, 2005–2017, http://www.public.asu.edu/~ccolbou/src/tabby/catable.html.

[6] C. J. Colbourn, E. Lanus and K. Sarkar, Asymptotic and constructive methods for covering perfect hash families and covering arrays, *Des. Codes Cryptogr.* **86** (2018), 907–937, doi:10.1007/s10623-017-0369-x.

[7] C. J. Colbourn, S. S. Martirosyan, G. L. Mullen, D. Shasha, G. B. Sherwood and J. L. Yucas, Products of mixed covering arrays of strength two, *J. Combin. Des.* **14** (2006), 124–138, doi:10.1002/jcd.20065.

[8] C. J. Colbourn, S. S. Martirosyan, Tran Van Trung and R. A. Walker, II, Roux-type constructions for covering arrays of strengths three and four, *Des. Codes Cryptogr.* **41** (2006), 33–57, doi:10.1007/s10623-006-0020-8.

[9] P. Damaschke, Adaptive versus nonadaptive attribute-efficient learning, *Mach. Learn.* **41** (2000), 197–215, doi:10.1023/a:1007616604496.

[10] N. Graham, F. Harary, M. Livingston and Q. F. Stout, Subcube fault-tolerance in hypercubes, *Inf. Comput.* **102** (1993), 280–314, doi:10.1006/inco.1993.1010.

[11] I. Izquierdo-Marquez, J. Torres-Jimenez, B. Acevedo-Juárez and H. Avila-George, A greedy-metaheuristic 3-stage approach to construct covering arrays, *Inform. Sci.* **460–461** (2018), 172–189, doi:10.1016/j.ins.2018.05.047.

[12] D. R. Kuhn, R. N. Kacker and Y. Lei, *Introduction to Combinatorial Testing*, Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, CRC Press, 2013.

[13] D. R. Kuhn, D. R. Wallace and A. M. Gallo, Software fault interactions and implications for software testing, *IEEE Trans. Software Eng.* **30** (2004), 418–421, doi:10.1109/tse.2004.24.

[14] L. V. Lejay, D. E. Shasha, P. M. Palenchar, A. Y. Kouranov, A. A. Cruikshank, M. F. Chou and G. M. Coruzzi, Adaptive combinatorial design to explore large experimental spaces: approach and validation, *IEE Proceedings - Syst. Biol.* **1** (2004), 206–212, doi:10.1049/sb:20045020.

[15] S. Martirosyan and Tran Van Trung, On t-covering arrays, *Des. Codes Cryptogr.* **32** (2004), 323–339, doi:10.1023/b:desi.0000029232.40302.6d.

[16] S. S. Martirosyan and C. J. Colbourn, Recursive constructions of covering arrays, *Bayreuth. Math. Schr.* **74** (2005), 266–275.

[17] P. Nayeri, C. J. Colbourn and G. Konjevod, Randomized post-optimization of covering arrays, *European J. Combin.* **34** (2013), 91–103, doi:10.1016/j.ejc.2012.07.017.

[18] S. Raaphorst, L. Moura and B. Stevens, A construction for strength-3 covering arrays from linear feedback shift register sequences, *Des. Codes Cryptogr.* **73** (2014), 949–968, doi:10.1007/s10623-013-9835-2.

[19] G. Roux, *k-propriétés dans des tableaux de n colonnes: cas particulier de la k-surjectivité et de la k-permutivité*, Ph.D. thesis, Université de Paris, 1987, http://www.theses.fr/1987PA066096.

[20] G. Seroussi and N. H. Bshouty, Vector sets for exhaustive testing of logic circuits, *IEEE Trans. Inform. Theory* **34** (1988), 513–522, doi:10.1109/18.6031.

[21] G. B. Sherwood, S. S. Martirosyan and C. J. Colbourn, Covering arrays of higher strength from permutation vectors, *J. Combin. Des.* **14** (2006), 202–213, doi:10.1002/jcd.20067.

[22] J. Torres-Jimenez and I. Izquierdo-Marquez, Covering arrays of strength three from extended permutation vectors, *Des. Codes Cryptogr.* (2018), doi:https://doi.org/10.1007/s10623-018-0465-6.

[23] J. Torres-Jimenez and I. Izquierdo-Marquez, A simulated annealing algorithm to construct covering perfect hash families, *Math. Probl. Eng.* **2018** (2018), Article ID 1860673, doi:10.1155/2018/1860673.

[24] G. Tzanakis, L. Moura, D. Panario and B. Stevens, Constructing new covering arrays from LFSR sequences over finite fields, *Discrete Math.* **339** (2016), 1158–1171, doi:10.1016/j.disc.2015.10.040.

[25] R. A. Walker, II and C. J. Colbourn, Tabu search for covering arrays using permutation vectors, *J. Statist. Plann. Inference* **139** (2009), 69–80, doi:10.1016/j.jspi.2008.05.020.