CONDITIONAL EXPECTATION ALGORITHMS FOR COVERING ARRAYS

CHARLES J. COLBOURN

ABSTRACT. An efficient conditional expectation algorithm for generating covering arrays has established a number of the best known upper bounds on covering array numbers. Despite its theoretical efficiency, the method requires a large amount of storage and time. In order to extend the range of its application, we generalize the method to find covering arrays that are invariant under the action of a group, reducing the search to consider only orbit representatives of interactions to be covered. At the same time, we extend the method to construct a generalization of covering arrays called quilting arrays. The extended conditional expectation algorithm, as expected, provides a technique for generating covering and quilting arrays that reduces the time and storage required. Remarkably, it also improves on the best known bounds on covering array numbers in a variety of parameter situations.

1. INTRODUCTION

Let N, k, t, and v be positive integers with $k \geq t$. Let C be an $N \times k$ array with entries from an alphabet Σ of size v; we typically take $\Sigma = \{0, \ldots, v-1\}$. Choose a t-tuple $(\nu_1, \ldots, \nu_t)$ with $\nu_i \in \Sigma$ for $1 \leq i \leq t$ and a tuple of t columns $(c_1, \ldots, c_t)$ with $c_i \in \{1, \ldots, k\}$, and $c_i \neq c_j$. Then $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ is a *t-way interaction*. The array *covers* the t-way interaction $\{(c_i, \nu_i) : 1 \leq i \leq t\}$ if, in at least one row ρ of C, the entry in row ρ and column c_i is ν_i for $1 \leq i \leq t$. Array C is a *covering array* $\mathsf{CA}(N; t, k, v)$ of *strength* t if every t-way interaction is covered.

Covering arrays are employed in applications in which experimental factors interact (see [6,8,19,20]). When applied in testing, columns correspond to experimental *factors*, and the symbols in the column form *values* or *levels* for the factor. Each row specifies the values to which to set the factors for an experimental run. We denote by $\mathsf{CAN}(t, k, v)$ the minimum N for which a $\mathsf{CA}(N; t, k, v)$ exists. Because $\mathsf{CAN}(1, k, v) = v$ and $\mathsf{CAN}(t, k, 1) = 1$, we generally assume that $k \geq t \geq 2$ and $v \geq 2$. The determination of

$\mathsf{CAN}(t,k,v)$ has been actively studied; see [5,8,11,20,21,26] for background. For fixed t and v, only $\mathsf{CAN}(2,k,2)$ has been determined exactly (see [20]).

The standard definition of covering array asks for all t-way interactions to be covered. In [17], that requirement is relaxed in a manner that we describe next. The *species* of a t-way interaction $\mathcal{S} = \{(c_i,\nu_i) : 1 \le i \le t\}$ is the multiset $\{\nu_i : 1 \le i \le t\}$; hence a species in general encompasses a number of specific t-way interactions. Often we are not concerned with the specific symbols used in defining the species. Then the *family* of a species is its orbit under the action of the symmetric group on v letters, and hence a family consists of a set of species, and therefore also a set of t-way interactions.

Let $\mathbb{S}$ be a set of species for t and v. An $N \times k$ array with v symbols is an $\mathbb{S}$-*quilting array* if every interaction whose species is in $\mathbb{S}$ is covered. The notation $\mathbb{S}$-$\mathsf{QA}(N;t,k,v)$ is used for such an array when $\mathbb{S}$ contains interactions of strength at most t, and $\mathbb{S}$-$\mathsf{QAN}(t,k,v)$ is the smallest N for which an $\mathbb{S}$-$\mathsf{QA}(N;t,k,v)$ exists. An $\mathbb{S}$-$\mathsf{QA}(N;t,k,v)$ is equivalent to a $\mathsf{CA}(N;t,k,v)$ when $\mathbb{S}$ contains all possible species of t-way interactions.

2. Conditional Expectation Algorithms

Bryce and Colbourn [2, 3] develop an efficient algorithm for generating covering arrays when v and t are fixed. Their method derandomizes a one-row-at-a-time random algorithm using the "method of conditional expectations", as described in more detail in [12, 13]. The method is a specialization of the technique of Stein [34], Lovász [29], and Johnson [24] to covering arrays; applied to covering arrays, it considers adding one row at a time that maximizes the number of newly covered t-way interactions. Naively this involves examining all possible next rows, but their number is exponential in k. Cohen, Litsyn, and Zémor [7] develop a variant that considers only rows from a suitably chosen larger orthogonal array.

The Stein-Lovász-Johnson method yields a bound: A $\mathsf{CA}(N;t,k,v)$ exists whenever

$$v^t\binom{k}{t}\left(\frac{v^t-1}{v^t}\right)^N < 1$$

Hence when v and t are fixed, $\mathsf{CAN}(t,k,v)$ is upper bounded by a constant multiple of $\log k$. The crucial feature of the method in [2, 3], however, is that it admits a polynomial time implementation (polynomial in k) when t and v are fixed. The essential idea is to build the array one row at a time, ensuring that each row covers *at least the number of as-yet-uncovered t-way interactions* that is the expected number covered by a row chosen uniformly at random. (This differs from earlier methods in that it does not insist that

a row cover the maximum, just the average.) To select one row to add, it starts with the 'empty' row $(\star, \ldots, \star)$ and fills in one entry at a time. In doing this, it ensures that the expected number of newly covered t-way interactions, among all ways to convert the remaining $\star$ entries to symbols of Σ, does not decrease. In this way, the row covers at least as many new t-way interactions as the expectation for a row selected entirely at random. The final step is to determine how to replace a single $\star$ with a symbol of Σ. To do this, the method selects a $\star$ entry, and for each $\sigma \in \Sigma$ it computes the expected number of newly covered t-way interactions, conditioned on replacing the $\star$ entry by σ. Then it chooses a symbol that gives the largest conditional expectation.

A few key observations underlie the efficiency of the method. First, the number of t-way interactions to cover is $v^t\binom{k}{t}$, which is polynomial in k when v and t are fixed. Secondly, there are $O(\log k)$ rows and k columns, so entries are to be selected $O(k \log k)$ times. Thirdly, and most importantly, because the expectation of a sum equals the sum of expectations, to determine the expected number of newly covered t-way interactions, it suffices to determine for each as-yet-uncovered interaction the probability with which it is covered. This in turn requires considering only the t columns that arise in the specific interaction, the remaining $k - t$ having no effect on the probability of occurrence. There are $\binom{k}{t}$ ways to choose columns for a t-way interaction. We compute the expected number of as-yet-uncovered t-way interactions *on these columns* that are covered by a randomly completed row. In fact, because we use this only to select a symbol in a specific column γ, the expectation for any set of columns not involving γ is independent of this choice, and hence we need only consider the $\binom{k-1}{t-1}$ t-sets of columns that contain γ.

AVERAGE_COVERING_ARRAY, shown in Figure 1, gives the conditional expectation algorithm. When $\mathcal{T}_{t,k,v}$ is the set of all $v^t\binom{k}{t}$ t-way interactions on k columns and v symbols, AVERAGE_COVERING_ARRAY($\mathcal{T}_{t,k,v}$) with $\mathbf{orb}(S) = \{S\}$ and $\mathbf{orbrep}(\mathcal{A}) = \mathcal{A}$ produces a $\mathsf{CA}(N; t, k, v)$ or fewer rows whenever

$$|\mathcal{T}_{t,k,v}| \left(\frac{v^t - 1}{v^t}\right)^N < 1 \text{ or equivalently } N > \log_{v^t/(v^t-1)}(|\mathcal{T}_{t,k,v}|).$$

More generally, let $\mathbb{S}$ be a set of species for t and v. When $\mathcal{S} = \{T \in \mathcal{T}_{t,k,v} : \text{the species of } T \text{ is in } \mathbb{S}\}$, AVERAGE_COVERING_ARRAY($\mathcal{S}$) produces an $\mathbb{S}$-quilting array.

Perhaps surprisingly, this conditional expectation method (under the name of the *Density Algorithm*) has proved very successful in finding smallest known covering arrays for a variety of choices of t, k, and v [2, 3, 16,

```
AVERAGE_COVERING_ARRAY(X)
    // X is a set of t-way interactions to cover
    X_0 ← X; ρ ← 0; 𝓛 ← ∅
    while X_ρ ≠ ∅ do
        y ← SELECT_AVERAGE_ROW(X_ρ)
        𝓛 ← 𝓛 ∪ {y}
        X_{ρ+1} ← X_ρ \ {R = {(γ_1, ν_1), ..., (γ_t, ν_t)} ∈ X_ρ :
                          {(γ_1, y_{γ_1}), ..., (γ_t, y_{γ_t})} ∈ orb(R)}
        ρ ← ρ + 1
    return 𝓛
SELECT_AVERAGE_ROW(X)
    r^(0) ← {⋆}^k
    for i from 1 to k do
        Choose a coordinate γ for which r_γ^(i-1) = ⋆
        maxcov ← 0
        for σ ∈ Σ
            z ← r^(i-1); z_γ ← σ; cov ← 0
            for {γ_1, ..., γ_t} with γ_i < γ_{i+1} for 1 ≤ i < t, and γ ∈ {γ_1, ..., γ_t}
                cov ← cov + EXPECTED_COMPLETIONS({γ_1, ..., γ_t}, X, z)
            if cov ≥ maxcov {maxcov ← cov; b ← z}
        r^(i) ← b
    return r^(k)
EXPECTED_COMPLETIONS(C, X, x) // for the uniform distribution
    F ← {γ ∈ C : x_γ = ⋆}; F̄ ← C \ F; count ← 0
    𝓐 ← orbrep({(a_{γ_1}, ..., a_{γ_t}) ∈ Σ^t : a_{γ_i} = x_{γ_i}) for γ_i ∈ F̄}
    for each T ∈ 𝓐
        if ∃S ∈ X with T ∈ orb(S) then count ← count + 1
    return count/|𝓐|
```

FIGURE 1. Conditional Expectation Algorithm

17, 27], particularly in conjunction with the post-optimization technique in [31, 32]; see [9] for current best known covering array numbers. Bryce and Colbourn [2, 3] examine two practical decisions, studied in more detail in [4]. By randomizing the order in which SELECT_AVERAGE_ROW considers symbols for each new entry in the row, because different symbols may yield the same value of *maxcov*, the choice of symbol is randomized to break ties randomly. While every row that SELECT_AVERAGE_ROW can produce provides at least the average number of newly covered t-way interactions, it may produce many different *candidate* rows. Hence we can generate a fixed number of candidates for each row, and choose a candidate to add that covers the most as-yet-uncovered t-way interactions. Once symbol selection is

randomized, the row chosen may vary, which causes a change to the set of t-way interactions remaining to be covered, in turn affecting the choice of the remaining rows. Thus when AVERAGE_COVERING_ARRAY($\mathcal{T}$) is executed again, a different covering array can result, which may have fewer rows. Hence we could perform a fixed number of *repetitions* of AVERAGE_COVERING_ARRAY($\mathcal{T}$) and select a smallest array. While more candidates and more repetitions (within reason) do appear to yield smaller covering arrays, naturally they impact execution time.

Although efficient, the method is limited in practice to values of k for which the coverage status of all $v^t\binom{k}{t}$ interactions can be maintained. While these can be readily computed whenever needed to avoid substantial storage requirements, the time required to examine each interaction repeatedly is large. In this paper, we explore an approach to alleviate the growth of the number of t-way interactions to an extent. Naturally, one wants to reduce the size of the search space. A sensible way to do this appears to be to assume some group action on the symbols, on the columns, or both. Computational methods that assume group actions on the array appear in [5, 10, 14, 28, 30], for example. Here we extend the conditional expectation algorithm to incorporate group actions on the set of symbols.

Suppose that $\mathcal{T}$ is a set of t-way interactions to be covered. Suppose that Γ is a (permutation) group acting on the symbols in Σ. Under the action of Γ, every t-way interaction T forms an orbit $\mathbf{orb}(T)$. We require that $T \in \mathcal{T}$ if and only if $\mathbf{orb}(T) \subseteq \mathcal{T}$. Under the action of Γ, if $T' \in \mathbf{orb}(T)$ then $\mathbf{orb}(T') = \mathbf{orb}(T)$, so any member of $\mathbf{orb}(T)$ serves as an *orbit representative* for $\mathbf{orb}(T)$. When $\mathcal{S}$ is a set of t-way interactions that is closed under the action of Γ, $\mathbf{orbrep}(\mathcal{S})$ denotes a minimal set of orbit representatives for $\mathcal{S}$.

In the same way, Γ acts on the set of possible rows, partitioning them into orbits. We extend the basic method by first specifying the functions $\mathbf{orb}(\cdot)$ and $\mathbf{orbrep}(\cdot)$ as dictated by Γ, and then setting $\mathcal{S} \subseteq \mathcal{T}^{\Gamma}_{t,k,v}$ to be a set of orbit representatives of t-way interactions. AVERAGE_COVERING_ARRAY($\mathcal{S}$) then produces a set S of rows. The rows in the orbit of S form an array that covers all t-way interactions in all orbits of interactions in $\mathcal{S}$; of course, the number of rows in the latter array may be as much as $|\Gamma|$ times the size of the array produced by AVERAGE_COVERING_ARRAY($\mathcal{S}$), because each orbit representative of a row may yield an orbit of size up to $|\Gamma|$.

Correctness follows the same argument as for the basic method. However, the guarantee on the number of rows is problematic. Because orbits of t-way interactions can in general have different size, and orbits of rows can also have different size, the expectation that a specific orbit of t-way interactions has a representative covered by a row selected at random is

therefore not a constant, but depends on the size of the orbit of t-way interactions. When t-way interactions partition into orbits of different sizes, an analysis of the number of rows needed could account for the different probabilities with which each orbit of t-way interactions has a representative covered by a randomly selected row.

Instead we choose Γ and the set of t-way interactions to be covered so that the orbits of t-way interactions all have full length $|\Gamma|$. Then the analysis for the number of rows required proceeds as in the basic method. Let us consider a specific case, taking Γ to be the cyclic group of order v. Then all orbits of t-way interactions are full. The resources required by AVERAGE_COVERING_ARRAY are substantially reduced. The number of rows to be found is reduced by a factor of that is expected to be approximately v, and the number of t-way interactions whose status must be known is also reduced by a factor of v. The price to be paid is that, in selecting a new row, we must check that a t-way interaction *has an orbit representative* in a completion of the partial row under construction, rather than checking that the interaction itself appears in it. This is easily done. The real questions are: What impact does this have on the practicality of the method? Does it continue to produce covering arrays of competitive sizes? We address these in Section 3.

Maintaining information about the coverage of orbit representatives of t-way interactions enables us to reduce space requirements, and appears to reduce the time required for the method as well. The savings depend on the order of Γ; larger groups ought to lead to faster generation using less space. Therefore, in addition to employing cyclic groups, we employ Frobenius groups. When v is a prime power, take the elements to be $\mathbb{F}_v$, and Γ to be the permutations $\{x \mapsto ax + b : a, b \in \mathbb{F}_v, a \neq 0\}$. Orbits of t-way interactions under Γ then have length $v(v-1)$ or v; not all have full length. However, when the orbit of $\{(c_1, \nu_1), \ldots, (c_t, \nu_t)\}$ has length v, it must happen that $\nu_1 = \cdots = \nu_t$. Consider the set $\mathcal{S}$ of all t-way interactions whose species is not of this form. Then Γ partitions $\mathcal{S}$ into full length orbits, and we can compute AVERAGE_COVERING_ARRAY($\mathcal{S}$) to cover all other interactions. Applying the action of Γ and then adding v constant rows yields a covering array. Determining whether a row covers an orbit representative of a t-way interaction under the Frobenius group is straightforward, so to further assess the method we also consider forming covering arrays that are invariant under the actions of Frobenius groups.

Naturally many other groups could be considered as well. Here we focus on the trivial, cyclic, and Frobenius groups in order to explore the consequences of incorporating a group action.

3. Computational Results

In the tables reported at [9], the original density algorithm of [2,3] has yielded best known results when $4 \leq t \leq 6$ and v is 'small'. Therefore we undertook an extensive set of computations using the variants of the method using cyclic and Frobenius groups for various choices of (t, v).

k	Den	SA	CD	FD	*FD	k	Den	SA	CD	FD	*FD
5	*81*		96	87	81	6	*111*		132	123	117
7	*123*		159	135	135	8	*135*		168	135	135
9	*135*		195	177	173	10	*159*	164	207	201	195
11	*183*		222	219	209	12	*201*		237	231	222
13	*219*		252	243	238	14	*237*	249	270	261	253
15	*237*	277	279	273	264	16	*237*	277	288	285	277
17	*297*	287	300	291	285	18	*297*	300	312	303	296
19	*311*	313	321	315	310	20	*315*	321	333	321	317
21	*315*	338	342	333	329	22	*315*	347	348	339	337
23	*315*	359	360	351	346	24	389	370	369	363	355
25	*384*	370	375	369	363	26	*393*	377	381	369	366
27	*393*	383	387	381	378	28	*393*	391	396	387	383
29	*393*	406	402	393	392	30	*393*	401	411	405	400
31	446	424	420	405	401	32	454	431	423	411	409
33	461	438	429	417	416	34	468	440	435	423	422

Table 1. $\mathsf{CA}(N; 4, k, 3)$

In Tables 1 and 2 we report upper bounds for $\mathsf{CAN}(4, k, 3)$ with $5 \leq k \leq 100$. In the column labelled 'Den', known bounds are reported for a variety of methods. When in plain font, the result is from the original density algorithm [3,27]. When in *slanted* font, the result is from [22] when $k = 5$, [15] when $k \in \{6, 7, 11, 12, 13\}$, [1] when $k \in \{8, 9\}$, [33] when $k \in \{10, 16\}$, and [37] when $20 \leq k \leq 23$ and $26 \leq k \leq 30$. When $k \in \{19, 25\}$, the array is constructed as in [37], after which the array is improved by a post-optimization technique. The post-optimization technique is described in [31, 32]; we use it to attempt to eliminate rows from solutions found, often with great success.

In the column labelled 'SA', we report the bounds produced by a sophisticated simulated annealing algorithm [1, 35, 36]. Within the range reported, this method produced the best known upper bounds prior to the results reported here. In the column labelled 'CD', we report on the results from an implementation of Average_Covering_Array using the action of the cyclic group. In the column labelled 'FD', we report on the results from an implementation of Average_Covering_Array using the

k	Den	SA	CD	FD	k	Den	SA	CD	FD	k	Den	SA	CD	FD
35		440	441	429	36		456	447	435	37		460	453	441
38		465	459	447	39		468	465	453	40	504	472	468	453
41	510	484	474	465	42	513	488	477	471	43	522	494	483	471
44	526	497	486	477	45	530	497	492	483	46	534	506	495	483
47	538	510	501	489	48	546	516	504	495	49		520	510	495
50		520	516	501	51		531	519	507	52	562	534	519	507
53	567	537	525	507	54	572	537	528	519	55	575	537	531	519
56	581	548	537	525	57	584	553	540	531	58	588	558	540	531
59	592	558	546	537	60		558	549	537	61	601	567	552	543
62	606	570	561	543	63	607	574	558	549	64		574	561	549
65		574	564	555	66	620	585	567	555	67		587	576	561
68		590	576	561	69		590	579	567	70	629	590	582	573
71		601	588	573	72		601	588	579	73		607	591	579
74		607	591	585	75		607	597	585	76		613	603	585
77		615	603	591	78		618	609	591	79		620	609	597
80		620	609	597	81		628	612	597	82		631	612	603
83		631	618	609	84		632	621	609	85		632	621	609
86		639	624	615	87		643	630	615	88		643	630	615
89		648	630	627	90		649	639	627	91		650	639	627
92		650	639	627	93		650	645	633	94		650	645	633
95		650	645	639	96		661	645	639	97		662	657	639
98		663	657	645	99		663	657	645	100		663	657	645

TABLE 2. CA(N; 4, k, 3)

action of the Frobenius group. The final column, labelled '*FD', reports some improvements on the results from the Frobenius group by applying post-optimization.

As expected, increasing the size of the group reduces both the storage and the time requirements. The reduced time requirement enables one to consider more repetitions and more candidates, but in order to keep the comparison 'fair', for each k, we attempt to use the same numbers of repetitions and candidates for AVERAGE_COVERING_ARRAY with the trivial, cyclic, and Frobenius groups. Despite the acceleration of the method, these computations are large. When $k = 100$, there are 3,921,225 ways to choose 4 columns. Thus there are 317,619,225 4-way interactions to cover for the trivial group, 105,873,075 orbits of interactions to cover for the cyclic group, and 50,975,925 for the Frobenius group. Because the method breaks ties randomly, the result reported need not be the one produced if the method is run again. Computations for the trivial group are limited, because the time and space required for the method render it impractical.

In the range of k for which density with trivial group has been applied, it is never competitive with the simulated annealing results. However, when we turn to results for the cyclic group, the contrast is striking. The method now produces results that are substantially better than those for the trivial group, typically reducing the number of rows generated by 10%. The improvement is perhaps most surprising when one considers that it comes with smaller requirements in space and time as well!

One might have expected that limiting the search to covering arrays that are invariant under the cyclic group would result in failing to consider some of the smallest covering arrays. That may still be the case. Except when k is very small, heuristic methods are unlikely to produce optimal covering arrays. Therefore, while it is possible that the restriction on the covering array imposed by the group action may indeed cause us to exclude certain arrays (including perhaps the optimal ones), the set of covering arrays invariant under the group action remains sufficiently rich to find best known solutions. Indeed the improvement obtained using the cyclic group is such that the method now improves upon *all* of the simulated annealing results as well!

Another surprise is in store. Using instead the action of the Frobenius group, a consistent improvement over the cyclic results is observed: Typically, a 1-2% reduction in the number of rows is obtained. Again we have better accuracy with less time and less space.

What accounts for the consistency of the improvement of the Frobenius and cyclic solutions over the basic ones? The reasons are not immediately obvious. The number of rows produced by AVERAGE_COVERING_ARRAY has an upper bound that is logarithmic in the number of orbits of t-way interactions to be covered. When the group is nontrivial, each row produced then yields an orbit of rows. Compare the bound for the method with the trivial group and with the cyclic group:

$$N > \log_{v^t/(v^t-1)} \left(v^t \binom{k}{t} \right) \text{ and } N > v \log_{v^{t-1}/(v^{t-1}-1)} \left(v^{t-1} \binom{k}{t} \right).$$

The bound for the cyclic group is better! (This can be verified by some algebraic manipulation.) The surprise is therefore not that assuming the action of a group can be better in theory, rather that it appears to be better in practice as well.

The results produced by AVERAGE_COVERING_ARRAY are typically not best possible, even when they are the best known. To explore this, we applied post-optimization to the results in Table 1 from the Frobenius group (which are the best produced here). In each case, post-optimization succeeds in removing rows, certifying that the results obtained are not the

best possible. At this time, it is impossible to assess how close the results are to the best possible, because there is no useful lower bound with which to compare.

Next we perform a more detailed set of computations for $v = 3$ and $t = 5$. Here we examine both covering arrays and quilting arrays for various families of species. Because of their application in the recursive methods of [17], we examine three families of species: $\mathbb{S}_8 = \{\{a, a, b, b, c\}\}$, $\mathbb{S}_7 = \mathbb{S}_8 \cup \{\{a, a, a, b, c\}\}$, and $\mathbb{S}_6 = \mathbb{S}_7 \cup \{\{a, a, a, b, b\}\}$. By restricting the interactions to be covered to those whose species family is in $\mathbb{S}_i$, we produce quilting arrays for this family. When a cyclic or Frobenius group acts, it does not change the family of the species, and hence it partitions the set of all interactions to cover into (full-length) orbits. For quilting arrays other than covering arrays, there are no published results with which to compare. To effect a comparison, we adapted the post-optimization technique to start with a covering array and eliminate rows to form a quilting array.

We report results in Table 3. The first band of bounds is for covering array numbers $\mathsf{CAN}(5, k, 3)$; the second is for $\mathbb{S}_6$-$\mathsf{QAN}(5, k, 3)$, the third for $\mathbb{S}_7$-$\mathsf{QAN}(5, k, 3)$, and the fourth for $\mathbb{S}_8$-$\mathsf{QAN}(5, k, 3)$. For covering arrays, we report results from various direct constructions: [22] for $k = 6$, [15] for $k = 7$, [38] for $k = 8$, and [37] for $k \in \{10, 13, 16, 19\}$. We report results for simulated annealing [1, 35, 36]. Then we report results for AVERAGE_COVERING_ARRAY with the trivial, cyclic, and Frobenius groups in the rows labelled Dens, CD, and FD. Finally, we applied postoptimization to each and report results in rows labelled *Dens, *CD, and *FD. The best results are shown in **boldface**.

For quilting arrays for $\mathbb{S}_s$, $s \in \{6, 7, 8\}$, the row labelled 'Postop' gives the result obtained using post-optimization for quilting arrays applied to all of the covering arrays with the same number of columns. Then rows labelled 'CDs' and 'FDs' report results from AVERAGE_COVERING_ARRAY applied to cover the interactions whose species family is in $\mathbb{S}_s$, and rows labelled '*CDs' and '*FDs' report results after post-optimization.

When direct constructions are known, neither simulated annealing nor the methods here outperform them. Simulated annealing yields the best results when k is small, but as k increases, AVERAGE_COVERING_ARRAY is the winner. Again, for covering arrays results for the Frobenius group generally beat those for cyclic groups, which always beat those for the trivial group.

AVERAGE_COVERING_ARRAY provides an effective method for the construction of quilting arrays. In these cases, however, the relative performance using the cyclic and Frobenius groups is not as predictable. This

	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Direct	**243**	**351**	**405**		483			**723**			**963**			1197						
SA			**405**	**405**	**405**	**550**	**600**	880	**890**	**944**	1025	1117	1165	**1190**	1257	1310	1319	1382	1417	1440
Dens	304	435	522	595	683	751	823	898	919	972	1072	1115	1162	1213	1256	1297	1345	1384	1422	1461
CD	318	432	525	609	684	768	831	897	954	1008	1071	1113	1167	1215	1257	1299	1338	1380	1416	1452
FD	261	429	513	563	669	753	825	879	951	1005	1065	1101	1155	1203	1251	1287	1317	1365	1401	1437
*Dens	268	397	486	564	644	725	792	872	901	960	1055	1107	1157	1206	1245	1290	1336	1378	1410	1454
*CD	261	384	480	571	648	734	809	876	932	995	1055	1097	1157	1208	1245	1291	1334	1373	1410	1448
*FD	**243**	390	478	**405**	639	718	795	852	916	981	1043	**1083**	**1143**	1191	**1239**	**1276**	**1311**	**1360**	**1394**	**1435**
Postop	**225**	345	**402**	**402**	**403**	**537**	**590**	**720**	**873**	937	**958**	1068	1120	1169	1229	1286	1313	1358	1403	1437
CD6	279	393	489	564	639	714	789	852	918	966	1008	1071	1116	1158	1203	1245	1284	1323	1362	1395
FD6	252	390	492	576	648	720	780	846	906	972	1020	1068	1116	1158	1206	1248	1290	1332	1362	1404
*CD6	242	**342**	438	515	591	676	760	834	905	**929**	994	1050	**1097**	**1147**	1197	**1240**	**1277**	**1313**	1356	**1390**
*FD6	240	344	438	518	596	682	750	821	881	941	1000	**1049**	1103	**1147**	**1196**	1242	1280	1324	**1355**	1398
Postop	**180**	**288**	**348**	**361**	**365**	**491**	**539**	714	772	826	893	940	994	1052	1123	1172	1217	1264	1316	1371
CD7	**180**	321	420	492	564	630	684	741	795	852	897	954	993	1038	1083	1125	1164	1200	1242	1269
FD7	**180**	318	402	492	558	624	684	744	798	846	894	948	996	1032	1080	1116	1158	1194	1230	1260
*CD7	**180**	290	367	451	525	585	658	**706**	**765**	824	883	939	**979**	1029	1077	1117	1155	1193	1235	1264
*FD7	**180**	291	370	445	515	577	655	703	768	**820**	**878**	**931**	983	**1022**	**1072**	**1108**	**1151**	**1185**	**1225**	**1255**
Postop	**90**	**162**	243	298	363	430	491	551	610	669	728	781	833	887	964	1009	1051	1099	1151	1189
CD8	**90**	195	267	309	402	468	516	582	642	690	738	795	834	888	930	975	1014	1053	1089	1125
FD8	**90**	186	246	318	390	462	516	576	636	690	738	792	840	882	930	972	1008	1056	1092	1128
*CD8	**90**	**162**	**168**	**168**	363	431	**489**	555	611	**662**	721	778	**823**	880	**920**	**962**	1006	**1045**	**1082**	**1118**
*FD8	**90**	**162**	**168**	289	**357**	**428**	**489**	**550**	**607**	667	**716**	**772**	826	**870**	921	**962**	**1001**	1048	1085	1121

TABLE 3. Quilting Arrays: $t = 5$, $v = 3$

k	Den	SA	CD	FD	k	Den	SA	CD	FD	k	Den	SA	CD	FD
26	1504	1488	1491	1479	27	1534	1527	1527	1515	28	1569	1552	1545	1545
29	1606	1585	1578	1575	30	1635	1601	1611	1605	31	1673	1642	1638	1635
32	1691	1666	1662	1659	33	1733	1697	1698	1695	34	1756	1719	1719	1707
35	1780	1748	1746	1743	36	1813	1775	1776	1767	37	1845	1799	1797	1785
38	1872	1829	1827	1821	39	1895	1851	1846	1839	40	1920	1866		1863
41	1944	1890		1887	42	1966	1923		1899	43	2001	1940		1935
44	2009	2089		1941	45	2044	2111			46	2066	2129		
47	2083	2149		2007	48	2106	2168		2025	49	2129	2189		
50	2149	2211			51				2085	52	2187			2103
53				2115	54				2133	55	2246			2157
56				2169	57				2187	58				2205
59				2211	60	2334			2229					

TABLE 4. CA$(N; 5, k, 3)$ for $26 \leq k \leq 60$

serves as a warning, perhaps, that simply making the group larger does not ensure a better (or even equal) result.

Table 4 reports further results for CA$(N; 5, k, 3)$.

Now we proceed to higher strength, bounds on CAN$(6, k, 3)$. In Table 5 we report results for direct constructions: [22] for $k = 7$, [15] for $k = 8$, [37] for $k \in \{11, 12, 14\}$, and [37] with post-optimization for $k = 9$. We report results for a variant of the `In-Parameter-Order` algorithm IPO [18], which has until this time produced the most extensive set of computational results for a variety of parameters. We report results from simulated annealing ('SA') [1,35,36]. Finally we report results for AVERAGE_COVERING_ARRAY with the trivial and Frobenius groups, and the sizes that result after post-optimization. Once again, when available the direct constructions provide the best known results. In the remaining cases, while IPO is typically much faster to compute, it is not competitive in terms of accuracy. The simulated annealing results typically beat AVERAGE_COVERING_ARRAY with the trivial group, but AVERAGE_COVERING_ARRAY with the Frobenius group beats simulated annealing when $k \geq 16$. With post-optimization, the results from the Frobenius group are useful improvements on the previously best known bounds. Regarding the remarkably good results for '*Dens' when $15 \leq k \leq 19$, a much larger number of covering arrays was produced by AVERAGE_COVERING_ARRAY in these cases, and the best post-optimized. To make the comparison of the different group assumptions fair, however, we report the sizes prior to post-optimization when the numbers of repetitions and candidates used are similar.

In Table 6, we again examine quilting arrays with strength five, but with four symbols. Here there are additional families of species: $S_9 =$

k	Dir	IPO	SA	Dens	*Dens	FD	*FD
7	729	990				927	729
8	1152	1490		1391	1259	1395	1259
9	1431	1847	1452	1774		1713	1585
10	1449	2190	1849	2103		2031	1921
11	1449	2512	2136			2361	2211
12	2181	2815	2206	2670		2625	2516
13		3106	2721	3019		2865	2781
14	2907	3358	2920	3255		3117	3050
15		3623	3338	3504	3223	3351	3289
16		3863	3647	3598	3435	3585	3553
17		4095	3873	3884	3654	3777	3755
18		4310	4098	4096	3846	3993	3974
19		4509	4299	4308	4051	4179	4162
20		4701	4373	4508	4486	4371	4363
21		4890	4571	4698	4678	4545	4535
22		5073	4732	4874	4853	4707	4700
23		5239	4941			4857	4855
24		5409	5100	5199	5193	5037	5035
25		5564	5238			5181	5180
26		5709	5380			5355	
27		5853	5667			5481	
28		6003	5827			5631	
29		6150	5969			5757	
30		6281	6103			5883	

TABLE 5. CA(N; 6, k, 3)

$\{\{a,a,b,c,d\}\}$, $\mathbb{S}_8 = \mathbb{S}_9 \cup \{\{a,a,b,b,c\}\}$, and $\mathbb{S}_7 = \mathbb{S}_8 \cup \{\{a,a,a,b,c\}\}$, and $\mathbb{S}_6 = \mathbb{S}_7 \cup \{\{a,a,a,b,b\}\}$. The results reported are as for Table 3, except that in this case only some of the relevant computations have been undertaken. The direct constructions are from [22] when $k = 6$, [23] when $k = 7$, [38] when $k = 8$, [37] when $k \in \{11, 15\}$, [32] when $k = 16$, and [37] with post-optimization when $k = 9$. The results for the trivial group are reported only after post-optimization, and only when no suitable direct construction is available. Again, for covering arrays the Frobenius group is consistently better, but for quilting arrays the pattern is not clear.

Further results for CAN$(5, k, 4)$ are given in Table 7, comparing the results for the trivial and the Frobenius groups. In Table 8, results for the Frobenius group for CAN$(6, k, 4)$ are compared with those from IPO.

We also report results when $v = 5$. In Table 9, we report results from direct constructions from [22] when $k = 6$, [23] when $k = 7$, and [38]

	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Dir	**1024**	**1536**	**1792**	**2032**	**2044**	**2044**	**3064**	**3064**	**3064**	**3064**	4548							
*Dens												**4672**	5081	5298	5501	5698	5883	5992
CD	1380	1856	2252	2620	2952	3280	3576	3848	4124	4372	4604	4836	5056	5264	5452	5656	5832	6008
FD	1288	1804	2236	2584	2932	3220	3556	3808	4096	4360	4576	4804	5032	5224	5416	5632	5788	5968
*CD	1159	1686	2100	2464	2851	3184												
*FD	**1024**	1675	2064	2451	2824	3138					**4546**	4784	**5015**	**5210**	**5402**	**5626**	**5779**	**5959**
P-o	**1020**	**1535**	**1791**	**2028**	**2040**	**2040**	**3049**	**3060**	**3060**	**3060**	4544	4668	5070	5290	5497	5694	5879	5988
CD6	1288	1772	2168	2532	2860	3172	3472	3772	4024	4284	4516	4740	4944	5172	5356	5552	5728	5908
FD6	1248	1776	2172	2532	2880	3192	3480	3780	4020	4284	4536	4752	4992	5172	5376	5556	5736	5904
*CD6	1110	1613	2000	2368	2726						**4494**		**4932**	**5157**	**5344**	**5541**	**5719**	5901
*FD6	1029	1597	2022	2378	2737													**5898**
P-o	**840**	1535	**1791**	**2016**	**2034**	**2034**	**2990**	**3048**	**3060**	**3060**	4512	4648	5014	5255	5455	5661	5858	5970
CD7	1128	1644	2024	2396	2736	3048	3332	3632	3884	4132	4356	4592	4808	5000	5200	5392	5556	5732
FD7	1056	1620	2016	2388	2736	3060	3336	3636	3900	4152	4392	4596	4788	5016	5196	5400	5556	5736
*CD7	848	**1475**	1863	2240	2594						**4334**	**4576**	4786	**4986**	5192	**5380**	**5546**	**5727**
*FD7	864	1481	1882	2237	2598								**4780**		**5186**			
P-o	**600**	1028	1358	1757	2026	**2034**	2695	3003	**3048**	**3048**	3807	4081	4339	4596	4853	5123	5295	5562
CD8	752	1148	1388	1812	2124	2420	2676	2964	3216	3456	3672	3900	4120	4312	4500	4692	4868	5032
FD8	636	1068	1164	1752	2064	2376	2640	2916	3180	3420	3672	3912	4128	4320	4500	4704	4896	5064
*CD8	**600**	**912**	**912**	1658	2008	2331	2591	2891			3648	**3875**	**4100**	**4295**	**4486**	**4680**	**4861**	**5024**
*FD8	**600**	**912**	**912**	**1646**	**1991**	2288	**2556**	**2855**	3156		**3635**				**4486**			
P-o	**360**	610	805	1036	1296	1555	1800	2069	2290	2519	2761	2979	3204	3438	3695	3902	4107	4305
CD9	420	676	860	1128	1384	1604	1816	2056	2292	2492	2688	2908	3108	3292	3448	3640	3812	3972
FD9	420	672	732	1104	1368	1632	1824	2064	2292	2496	2688	2904	3084	3276	3444	3624	3804	3948
*CD9	362	613	689	1024	**1273**	**1509**	**1745**	**1994**	**2230**	**2449**	**2649**	2874	3080	3272	3435	3632	3804	3961
*FD9	384	**608**	**624**	**1011**	1286	1531	1746	1995	2240	2452	**2649**	**2872**	**3066**	**3255**	**3428**	**3610**	**3795**	**3945**

TABLE 6. Quilting Arrays: $t = 5$, $v = 4$

k	Dens	FD	k	Dens	FD	k	Dens	FD	k	Dens	FD	k	Dens	FD
24	6249	6148	25	6427	6316	26	6578	6472	27	6862	6628	28	6948	6760
29	7040	6892	30	7156	7024	31	7324	7156	32	7446	7276	33	7571	7396
34	7698	7516	35	7820	7648	36		7732	37		7864	38		7972
39		8068	40	8364	8188	41		8260	42		8368	43		8476
44		8560	45		8632									

TABLE 7. CA$(N; 5, k, 4)$ for $24 \leq k \leq 45$

k	IPO	FD	k	IPO	FD	k	IPO	FD
16	22608	20848	17	23947	22096	18	25212	23236
19	26392	24364	20	27534	25420			

TABLE 8. CA$(N; 6, k, 4)$ for $16 \leq k \leq 20$

k	Dir	IPO	MIPOG	*MIPOG	*Dens	FD	*FD
6	3125	4195				4205	3660
7	4375	5942			5744	5625	5100
8	5000	7349			6911	6865	6374
9		8629	6996	6634	7647	7965	7642
10		9796	8169	7666	8700	9045	8783
11		10862	9067	8554	9975	10025	9858
12		11889		9475		10925	10790
13		12851	11004	10598	12014	11785	11715
14		13748	11924	11592	12825	12605	12550
15		14578	12704	12534		13405	13365
16		15379	13469	13282	14348	14205	14180
17		16128			15797	14845	14824
18		16843			16479	15545	15526
19		17516			16937	16185	16170
20		18171			16958	16845	16836
21		18779			17596	17385	17376
22		19387				17925	17923
23		19941			18793	18485	
24		20482				19045	
25		21004			19840	19525	
26		21518				20025	
27		21999				20485	
28		22488				20945	
29		22929				21385	
30		23369				21785	
31		23789				22205	
32		24205				22585	

TABLE 9. CA$(N; 5, k, 5)$

k	Dir	IPO	Dens	FD	*FD
7	15625	22100		22485	19339
8	25000	32822		31005	27776
9		41210	40963	38365	35696
10		49111	48347	45445	43461
11		56615	53314	52045	50872
12		63620		58465	57872
13		70190		64485	64152
14	72681	76390		70205	69944
15		82139		75605	75526
16		87559		80545	80537

TABLE 10. CA(N; 6, k, 5)

when $k = 8$. We report results from IPO [18], and from a massively parallel implementation of the IPO strategy, MIPOG [39]; for the latter we also report results after post-optimization. Then we report results for AVERAGE_COVERING_ARRAY with the trivial group and with the Frobenius group, and also post-optimization results for the latter. While AVERAGE_COVERING_ARRAY with the trivial group always beats IPO, the substantial computational effort of MIPOG yields much better results. Nevertheless, AVERAGE_COVERING_ARRAY with the Frobenius group yields a clear improvement. While this still does not match the results from MIPOG, it extends the range for which results can be obtained in a reasonable time. Again we emphasize that none of these methods produce arrays that are best possible; even the better results obtained by MIPOG admit substantial improvement via post-optimization.

Results for CAN(6, k, 5) are given in Table 10. The direct results are from [22] when $k = 7$, [38] when $k = 8$, and a randomized method called Paintball [25] when $k = 14$. While AVERAGE_COVERING_ARRAY with the Frobenius group yields useful improvements here, the scale of the computation still effectively restricts the application to relatively small values of k.

Finally we report results for $v = 6$. In these cases, the largest group employed is the cyclic group, and comparisons are made with IPO. Table 11 gives results for CAN(4, k, 6), and Table 12 gives results for CAN(5, k, 6).

We have only one useful result when $v \geq 7$, a CA(87661;5,19,7) using the Frobenius group, at this time.

4. CONCLUDING REMARKS

AVERAGE_COVERING_ARRAY provides a general method for finding covering arrays and quilting arrays that is efficient when t and v are fixed, and

k	IPO	CD	k	IPO	CD	k	IPO	CD	k	IPO	CD
9	3374	3168	10	3713	3492	11	4011	3750	12	4295	4020
13	4553	4254	14	4800	4458	15	5024	4674	16	5248	4884
17	5449	5052	18	5650	5250	19	5841	5430	20	6015	5604
21	6186	5748	22	5352	5904	23	6508	6066	24	6662	6180
25	6809	6330	26	6953	6456	27	7087	6606	28	7226	6714
29	7348	6852	30	7473	6966	31	7598	7092	32	7711	7200
33	7825	7320	34	7931	7410	35	8044	7506	36	8154	7614
37	8255	7716	38	8357	7812	39	8455	7902	40	8550	7992
41	8646	8094	42	8740	8184	43	8823	8256	44	8907	8352
45	8994	8436	46	9079	8508	47	9161	8586	48	9240	8670
49	9323	8760	50	9393	8826	51	9466	8910	52	9550	8982
53	9623	9054	54	9696	9120	55	9762	9186			

TABLE 11. $\mathsf{CA}(N; 4, k, 6)$

k	IPO	CD	k	IPO	CD	k	IPO	CD
17	40334	37206	18	42102	39996	19	43833	40548
20	45425	42048	21	46970	43536	22	49479	45018

TABLE 12. $\mathsf{CA}(N; 5, k, 6)$

that always yields an array whose size is bounded by a constant multiple of $\log k$ (the constant multiple being determined by the fixed values of t and v). Thus we can realize the promise of the Stein-Lovász-Johnson paradigm when the parameters t, v, and k permit us to store and record coverage information for all t-way interactions. More importantly, the same can be achieved when a group acts on the symbols of the array. Not only is there a concomitant reduction in the time and space required, there is also an improvement on the bounds obtained both in theory and in practice. The implementation of the approach provides many improvements in best known covering array numbers, typically yielding much better bounds than the method would achieve if it indeed only obtained the average new coverage in each row selected.

The surprising conclusion that imposing the action of a larger group leads to a better bound, at the same time reducing time and storage, has a lot of promise for developing methods in which covering arrays with substantial symmetries are present. We expect this to yield covering array generation algorithms that are both more practical *and* more accurate.

References

[1] H. Avila-George, J. Torres-Jimenez, and V. Hernandez. New bounds for ternary covering arrays using parallel simulated annealing. *Mathematical Problems in Engineering*, 2012, 2012. Article ID 897027, 19 pages.

[2] R. C. Bryce and C. J. Colbourn. The density algorithm for pairwise interaction testing. *Software Testing, Verification, and Reliability*, 17:159–182, 2007.

[3] R. C. Bryce and C. J. Colbourn. A density-based greedy algorithm for higher strength covering arrays. *Software Testing, Verification, and Reliability*, 19:37–53, 2009.

[4] R. C. Bryce, C. J. Colbourn, and M. B. Cohen. A framework of greedy methods for constructing interaction tests. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 146–155, Los Alamitos, CA, 2005. IEEE.

[5] M. A. Chateauneuf and D. L. Kreher. On the state of strength-three covering arrays. *J. Combin. Des.*, 10:217–238, 2002.

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

[7] G. Cohen, S. Litsyn, and G. Zémor. On greedy algorithms in coding theory. *IEEE Trans. Inform. Theory*, 42:2053–2057, 1996.

[8] C. J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche (Catania)*, 58:121–167, 2004.

[9] C. J. Colbourn. Covering array tables, 2005-2013. http://www.public.asu.edu/~ccolbou/src/tabby.

[10] C. J. Colbourn. Covering arrays from cyclotomy. *Des. Codes Cryptogr.*, 55:201–219, 2010.

[11] C. J. Colbourn. Covering arrays and hash families. In *Information Security and Related Combinatorics*, NATO Peace and Information Security, pages 99–136. IOS Press, 2011.

[12] C. J. Colbourn. Efficient conditional expectation algorithms for constructing hash families. *Lecture Notes in Computer Science*, 7056:144–155, 2011.

[13] C. J. Colbourn, D. Horsley, and V. R. Syrotiuk. Strengthening hash families and compressive sensing. *Journal of Discrete Algorithms*, 16:170–186, 2012.

[14] C. J. Colbourn and G. Kéri. Covering arrays and existentially closed graphs. *Lecture Notes in Computer Science*, 5557:22–33, 2009.

[15] C. J. Colbourn, G. Kéri, P. P. Rivas Soriano, and J.-C. Schlage-Puchta. Covering and radius-covering arrays: Constructions and classification. *Discrete Applied Mathematics*, 158:1158–1190, 2010.

[16] C. J. Colbourn and J. Torres-Jiménez. Heterogeneous hash families and covering arrays. *Contemporary Mathematics*, 523:3–15, 2010.

[17] C. J. Colbourn and J. Zhou. Improving two recursive constructions for covering arrays. *Journal of Statistical Theory and Practice*, 6:30–47, 2012.

[18] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn. Refining the in-parameter-order strategy for constructing covering arrays. *J. Res. Nat. Inst. Stand. Tech.*, 113:287–297, 2008.

[19] M. Grindal, J. Offutt, and S. F. Andler. Combination testing strategies – a survey. *Software Testing, Verification, and Reliability*, 5:167–199, 2005.

[20] A. Hartman. Software and hardware testing using combinatorial covering suites. In M. C. Golumbic and I. B.-A. Hartman, editors, *Interdisciplinary Applications of Graph Theory, Combinatorics, and Algorithms*, pages 237–266. Springer, Norwell, MA, 2005.

[21] A. Hartman and L. Raskin. Problems and algorithms for covering arrays. *Discrete Math.*, 284:149–156, 2004.
[22] A. S. Hedayat, N. J. A. Sloane, and J. Stufken. *Orthogonal Arrays.* Springer-Verlag, New York, 1999.
[23] L. Ji, Y. Li, and J. Yin. Constructions of covering arrays of strength five. *Des. Codes Cryptography*, 62(2):199–208, 2012.
[24] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.
[25] D. R. Kuhn, Y. Lei, R. Kacker, V. Okun, and J. Lawrence. Paintball: A fast algorithm for covering arrays of high strength. *Internal Tech. Report, NISTIR 7308*, 2007.
[26] V. V. Kuliamin and A. Petukhov. A survey of methods for constructing covering arrays. *Programming and Computer Software*, 37(3):121–146, 2011.
[27] D. Linnemann and M. Frewer. Computations with the density algorithm (private communication by e-mail), October 2008.
[28] J. R. Lobb, C. J. Colbourn, P. Danziger, B. Stevens, and J. Torres-Jimenez. Cover starters for strength two covering arrays. *Discrete Mathematics*, 312:943–956, 2012.
[29] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math.*, 13(4):383–390, 1975.
[30] K. Meagher and B. Stevens. Group construction of covering arrays. *J. Combin. Des.*, 13:70–77, 2005.
[31] P. Nayeri, C. J. Colbourn, and G. Konjevod. Randomized postoptimization of covering arrays. *Lecture Notes in Computer Science*, 5874:408–419, 2009.
[32] P. Nayeri, C. J. Colbourn, and G. Konjevod. Randomized postoptimization of covering arrays. *European Journal of Combinatorics*, 34:91–103, 2013.
[33] G. B. Sherwood, S. S. Martirosyan, and C. J. Colbourn. Covering arrays of higher strength from permutation vectors. *J. Combin. Des.*, 14:202–213, 2006.
[34] S. K. Stein. Two combinatorial covering theorems. *J. Combinatorial Theory Ser. A*, 16:391–397, 1974.
[35] J. Torres-Jimenez. Covering array tables, 2010-2013. http://www.tamps.cinvestav.mx/~jtj/.
[36] J. Torres-Jimenez and E. Rodriguez-Tello. New upper bounds for binary covering arrays using simulated annealing. *Information Sciences*, 185(1):137–152, 2012.
[37] R. A. Walker II and C. J. Colbourn. Tabu search for covering arrays using permutation vectors. *J. Stat. Plann. Infer.*, 139:69–80, 2009.
[38] F. Xu, L. Ji, and Z. Dong. Constructions of covering arrays with strength from four to eight. *preprint*, 2009.
[39] M. I. Younis and K. Z. Zamli. MC-MIPOG: A parallel t-way test generation strategy for multicore systems. *ETRI Journal*, 32(1):73–83, 2010.

SCHOOL OF COMPUTING, INFORMATICS, AND DECISION SYSTEMS ENGINEERING, ARIZONA STATE UNIVERSITY, TEMPE AZ 85287-8809, U.S.A. *and* STATE KEY LABORATORY OF SOFTWARE DEVELOPMENT ENVIRONMENT, BEIHANG UNIVERSITY, BEIJING 100191, CHINA

E-mail address: Charles.Colbourn@asu.edu