SIAM J. DISCRETE MATH.
Vol. 27, No. 4, pp. 1844–1861

SEQUENCE COVERING ARRAYS*

YEOW MENG CHEE†, CHARLES J. COLBOURN‡, DANIEL HORSLEY§, AND JUNLING ZHOU¶

Abstract. Sequential processes can encounter faults as a result of improper ordering of subsets of the events. In order to reveal faults caused by the relative ordering of t or fewer of v events, for some fixed t, a test suite must provide tests so that every ordering of every set of t or fewer events is exercised. Such a test suite is equivalent to a sequence covering array, a set of permutations on v events for which every subsequence of t or fewer events arises in at least one of the permutations. Equivalently it is a (different) set of permutations, a completely t-scrambling set of permutations, in which the images of every set of t chosen events include each of the $t!$ possible "patterns." In event sequence testing, minimizing the number of permutations used is the principal objective. By developing a connection with covering arrays, lower bounds on this minimum in terms of the minimum number of rows in covering arrays are obtained. An existing bound on the largest v for which the minimum can equal $t!$ is improved. A conditional expectation algorithm is developed to generate sequence covering arrays whose number of permutations never exceeds a specified logarithmic function of v when t is fixed, and this method is shown to operate in polynomial time. A recursive product construction is established when $t = 3$ to construct sequence covering arrays on vw events from ones on v and w events. Finally computational results are given for $t \in \{3, 4, 5\}$ to demonstrate the utility of the conditional expectation algorithm and the product construction.

Key words. sequence covering array, completely scrambling set of permutations, covering array, directed t-design

AMS subject classifications. 05B40, 05B15, 05B30, 05A05

DOI. 10.1137/120894099

1. Introduction.

A set of permutations $\{\pi_1, \ldots, \pi_N\}$ of a v-element set X is *completely t-scrambling* if for every ordered t-set $(x_1, \ldots, x_t)$ with $x_i \in X$ for $1 \le i \le t$, there is some ρ $(1 \le \rho \le N)$ for which $\pi_\rho(x_i) < \pi_\rho(x_j)$ if and only if $i < j$. Spencer [32] first explored the existence of completely t-scrambling sets of permutations in generalizing a question of Dushnik [15] on linear extensions. Recently Kuhn et al. [23, 24] examined an equivalent combinatorial object, the *sequence covering array*. For parameters N, t, and v, such an array is a set of n permutations of v letters so that every permutation of every t of the v letters appears—in the specified order—in at least one of the n permutations. The motivation for finding sequence covering arrays with small values of n arises in event sequence testing. Suppose that a process involves a sequence of v tasks or events. The operator may, unfortunately, fail to do the tasks in the correct sequence. When this happens, errors may occur. But we anticipate that errors can be attributed to the (improper) ordering of a small

*Received by the editors October 8, 2012; accepted for publication (in revised form) September 4, 2013; published electronically October 28, 2013.
http://www.siam.org/journals/sidma/27-4/89409.html

†School of Physical and Mathematical Sciences, Nanyang Technological University 637371, Singapore (YMChee@ntu.edu.sg).

‡School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85257, and State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China (colbourn@asu.edu). This author was supported by Australian Research Council grant DP120103067.

§School of Mathematical Sciences, Monash University, Melbourne, Australia (danhorsley@gmail.com). This author was supported by Australian Research Council grants DP120103067 and DE120100040.

¶Department of Mathematics, Beijing Jiaotong University, Beijing, China (jlzhou@bjtu.edu.cn).

subset of tasks. When each permutation of a sequence covering array is used in turn to specify a task order, every potential ordering of t or fewer tasks will be tried and hence all errors found that result solely from the improper ordering of t or fewer tasks. Applications are discussed further in [19, 23, 39, 40]; related event sequence testing problems in which tasks can be repeated are discussed in [42, 43, 44]. While the application of these combinatorial structures is of much practical concern, our interest is in bounds on the size of sequence covering arrays and their explicit construction.

We first state the problem formally. Let $\Sigma = \{0, \ldots, v-1\}$ be *symbols* that represent the v tasks or events. A *t-subsequence* of Σ is a t-tuple $(x_1, \ldots, x_t)$ with $x_i \in \Sigma$ for $1 \le i \le t$, and $x_i \ne x_j$ when $i \ne j$. A permutation π of Σ *covers* the t-subsequence $(x_1, \ldots, x_t)$ if $\pi^{-1}(x_i) < \pi^{-1}(x_j)$ whenever $i < j$. For example, with $v = 5$ and $t = 3$, $(4, 0, 3)$ is a 3-subsequence that is covered by the permutation 4 2 0 3 1. A *sequence covering array* of *order* v and *strength* t, or $\mathsf{SeqCA}(N; t, v)$, is a set $\Pi = \{\pi_1, \ldots, \pi_N\}$, where π_i is a permutation of Σ, and every t-subsequence of Σ is covered by at least one of the permutations $\{\pi_1, \ldots, \pi_N\}$. Often the permutations are written as an $N \times v$ array.

We use an array representation for completely t-scrambling sets of permutations as well. An $N \times v$ array is a *completely t-scrambling set of permutations* of strength t on v symbols, or $\mathsf{CSSP}(N; t, v)$, when the columns are indexed by Σ and the symbols by Σ, and for every way $c_1, \ldots, c_t$ to choose t distinct columns and every permutation ϕ of $\{1, \ldots, t\}$, there is a row ρ for which, for every $1 \le a < b \le t$, the entry in cell $(\rho, c_{\phi(a)})$ is less than the entry in cell $(\rho, c_{\phi(b)})$.

LEMMA 1.1. *A* $\mathsf{CSSP}(N; t, v)$ *is equivalent to a* $\mathsf{SeqCA}(N; t, v)$.

Proof. If $\pi_1, \ldots, \pi_N$ are the N permutations of a $\mathsf{SeqCA}(N; t, v)$, form an $N \times v$ array A in which cell (i, j) contains $\pi_i^{-1}(j)$. Then A is a $\mathsf{CSSP}(N; t, v)$.

In the opposite direction, if A is a $\mathsf{CSSP}(N; t, v)$, define permutation π_i by setting $\pi_i(a_{ij}) = j$ for $1 \le i \le N$ and $0 \le j < v$. Then $\pi_1, \ldots, \pi_N$ form the N permutations of a $\mathsf{SeqCA}(N; t, v)$. ∎

In the $\mathsf{CSSP}(8;3,5)$ of Table 1.1, the symbols $\{1, 2, 3\}$ appear as 123 once, 132 once, 213 once, 231 zero times, 312 four times, and 321 once. Hence the rows of a completely t-scrambling set of permutations do not necessarily produce a sequence covering array; nevertheless they are conjugates, obtained by interchanging the roles of columns and symbols.

Permutation problems concerning the avoidance of specified patterns of subsequences have been extensively studied in algebraic and probabilistic combinatorics; see [33] for an excellent survey. (Here two subsequences $(x_1, \ldots, x_t)$ and $(y_1, \ldots, y_t)$ have the same *pattern* when, for $1 \le i < t$, $x_i < x_{i+1}$ if and only if $y_i < y_{i+1}$.)

TABLE 1.1
Example: $\mathsf{SeqCA}(8;3,5)$ – $t = 3$, $v = 5$, $N = 8$.

SeqCA	CSSP
4 2 0 3 1	2 4 1 3 0
1 4 3 0 2	3 0 4 2 1
3 1 2 0 4	3 1 2 0 4
0 2 4 1 3	0 3 1 4 2
2 1 3 4 0	4 1 0 2 3
0 3 4 1 2	0 3 4 1 2
3 0 2 1 4	1 3 2 0 4
4 1 2 0 3	3 1 2 4 0

While cosmetically similar to pattern avoidance problems, the existence problem for sequence covering arrays requires coverage rather than avoidance and requires that all subsequences be covered and not simply every pattern.

The question of principal concern in this paper is as follows: Given t and v, what is the smallest N for which a $\mathsf{CSSP}(N;t,v)$ (equivalently, a $\mathsf{SeqCA}(N;t,v)$) exists? Call this number $\mathsf{SeqCAN}(t,v)$. In the vernacular of completely t-scrambling sets of permutations, Spencer [32] did the foundational work, and Füredi [17], Ishigami [20, 21], Radhakrishnan [31], and Tarui [36] made improvements.

In a sequence covering array, every t symbols must appear in each of the $t!$ possible orderings, and there are $\binom{v}{t}t!$ t-subsequences in total, so

$$t! \le \mathsf{SeqCAN}(t,v) \le \binom{v}{t} t!$$

Both bounds are trivial, but the lower bound is the correct one when $t = 2$.

LEMMA 1.2. $\mathsf{SeqCAN}(2,v) = 2$ *for all* $v \ge 2$.

Proof. Any permutation on v symbols and its reversal form a $\mathsf{SeqCA}(2;2,v)$. ∎

When $t \ge 3$, neither bound is correct as v increases. Indeed the growth as a function of v for fixed t is logarithmic.

THEOREM 1.3 (see [31, 32]). *For* $t \ge 3$,

$$1 + \left(\frac{2}{\log_2(e)}\right)(t-1)!\left(\frac{v}{2v-t+1}\right)\log_2(v-t+2) \le \mathsf{SeqCAN}(t,v) \le \frac{t\log(v)}{\log\left(\frac{t!}{t!-1}\right)}.$$

The question of when $\mathsf{SeqCAN}(t,v) = t!$ is of independent interest in yet another setting. Let V be a finite set; an element of V is a *vertex*. A *transitive tournament* on V is a directed graph in which (1) for all $x \in V$, (x,x) is not an arc; (2) for distinct $x, y \in V$, (x,y) is an arc if and only if (y,x) is not an arc; and (3) whenever (x,y) and (y,z) are arcs, so is (x,z). A transitive tournament $T = (V,A)$ has transitive tournament $T' = (W,B)$ as a *subdigraph*, denoted $T' \prec T$, whenever $W \subseteq V$ and $B \subseteq A$. Let $(V,\mathcal{T})$ be a finite set V of cardinality v and a collection $\mathcal{T}$ with every $T \in \mathcal{T}$ being a transitive tournament on k of the vertices in V; members of $\mathcal{T}$ are *blocks*. Then $(V,\mathcal{T})$ is a (t,λ)-*directed packing of blocksize* k *and order* v, or $\mathrm{DP}_\lambda(t,k,v)$, if for every $X \subseteq V$ with $|X| = t$ and every transitive tournament T' on vertex set X,

$$|\{T \in \mathcal{T} : T' \prec T\}| \le \lambda.$$

On the other hand, $(V,\mathcal{T})$ is a (t,λ)-*directed covering of blocksize* k *and order* v, or $\mathrm{DC}_\lambda(t,k,v)$), if for every $X \subseteq V$ with $|X| = t$ and every transitive tournament T' on vertex set X,

$$|\{T \in \mathcal{T} : T' \prec T\}| \ge \lambda.$$

When $(V,\mathcal{T})$ is a $\mathrm{DP}_\lambda(t,k,v)$ and also a $\mathrm{DC}_\lambda(t,k,v)$, it is a (t,λ)-*directed design of blocksize* k *and order* v, or $\mathrm{DD}_\lambda(t,k,v)$. In this notation, the subscript is often omitted when $\lambda = 1$.

Directed designs with $t = 2$ have been extensively studied as generalizations of balanced incomplete block designs. The study of $(t,1)$-directed packings has also been extensive as a result of their equivalence to "deletion-correcting codes" (see Levenshtein [25]). The connection with our investigation follows.

LEMMA 1.4. *A* $\mathsf{CSSP}(N;t,v)$ *is equivalent to a* $DC(t,v,v)$ *with* N *blocks. Moreover, a* $DD(t,v,v)$ *exists if and only if* $\mathsf{SeqCAN}(t,v)=t!$.

Proof. For each row $(a_0,\ldots,a_{v-1})$ of the CSSP, form a transitive tournament on vertex set $\{0,\ldots,v-1\}$ by including, for $0\le i<j<v$, arc (i,j) if $a_i<a_j$ and arc (j,i) otherwise. Each transitive tournament on t of these vertices is a subdigraph of at least one of these N tournaments. The other direction is similar. When $N=t!$, every t-subsequence is covered exactly once, and every transitive tournament of order t arises as a subdigraph exactly once. ∎

THEOREM 1.5. *Sufficient conditions for a* $DD(t,v,v)$ *to exist include*

1. $t\ge 3$ *and* $t\le v\le t+1$ [25] *and*
2. $t=4$ *and* $v=6$ [28].

Necessary conditions for a $DD(t,v,v)$ *to exist include*

1. $v\le t+1$ *for* $t\in\{3,5,6\}$ [28],
2. $v\le t+2$ *for* $t=4$ [28], *and*
3. $v\le\binom{t+1}{2}-1$ *for* $t\ge 7$ [28].

Levenshtein [25] had conjectured that $v\le t+1$ whenever a $\mathrm{DD}(t,v,v)$ exists for $t\ge 3$. As stated in Theorem 1.5, this does not hold for $t=4$, but this is the only known exception to Levenshtein's conjecture. In the next section we significantly reduce the upper bound on the largest v for which $\mathsf{SeqCAN}(t,v)$ can equal $t!$.

2. Lower bounds. Here we extend a technique used in [17, Theorem 5.1], improving on a method of Ishigami [21]. We require a number of previous results on covering arrays, introduced next. See [8] for a more thorough introduction to them. Let N, k, t, and v be positive integers. Let C be an $N\times k$ array with entries from an alphabet Σ of size v; we typically take $\Sigma=\{0,\ldots,v-1\}$. When $(\nu_1,\ldots,\nu_t)$ is a t-tuple with $\nu_i\in\Sigma$ for $1\le i\le t$, $(c_1,\ldots,c_t)$ is a tuple of t column indices $(c_i\in\{1,\ldots,k\})$, and $c_i\ne c_j$ whenever $\nu_i\ne\nu_j$, the t-tuple $\{(c_i,\nu_i):1\le i\le t\}$ is a *t-way interaction*. The array *covers* the t-way interaction $\{(c_i,\nu_i):1\le i\le t\}$ if, in at least one row ρ of C, the entry in row ρ and column c_i is ν_i for $1\le i\le t$. Array C is a *covering array* $\mathsf{CA}(N;t,k,v)$ of *strength* t if it covers every t-way interaction. $\mathsf{CAN}(t,k,v)$ is the minimum N for which a $\mathsf{CA}(N;t,k,v)$ exists. The basic goal is to minimize the number of rows (tests) required and hence to determine $\mathsf{CAN}(t,k,v)$. When $t\ge 2$ and $v\ge 2$ are both fixed, $\mathsf{CAN}(t,k,v)$ is $\Theta(\log k)$ (see, for example, [8]).

We strengthen this standard definition somewhat. For a t-way interaction $T=\{(c_i,\nu_i):1\le i\le t\}$ with symbols chosen from $\Sigma=\{0,\ldots,v-1\}$, let $\tau_\sigma(T)=|\{i:\nu_i=\sigma\}|$. Then define $\mu(T)=\prod_{\sigma=0}^{v-1}\tau_\sigma(T)!$. The natural interpretation is that $\mu(T)$ is the number of ways to permute the columns $(c_1,\ldots,c_t)$ so that the symbols appear in the same order. A covering array provides *excess coverage* when every t-way interaction T is covered by at least $\mu(T)$ rows; such a covering array is denoted by $\mathsf{CA}_X(N;t,k,v)$. More generally, the subscript X is used to extend notation from covering arrays to those having excess coverage.

THEOREM 2.1. *Let* v, t, *and* a *be integers satisfying* $v\ge t\ge 3$ *and* $t\ge a\ge 0$. *Then* $\mathsf{SeqCAN}(t,v)\ge a!\mathsf{CAN}_X(t-a,v-a,a+1)$.

Proof. Let S be a $\mathsf{CSSP}(N;t,v)$. Choose any a columns of S, $\{e_1,\ldots,e_a\}$. For each ordering π of these columns, form a matrix C_π that contains all rows of S in which the entry in column $\pi(e_i)$ is less than that in column $\pi(e_{i+1})$ for $1\le i<a$. Because there are $a!$ orderings and every row of S appears in exactly one of the $\{\mathsf{C}_\pi\}$, it suffices to show that for every choice of π the number n of rows in C_π is at least $\mathsf{CAN}_X(t-a,v-a,a+1)$. To do this, form an $n\times(v-a)$ array A_π whose columns are the columns of C_π that are not among the a selected. To determine the content of

cell (r, c) of A_π, examine the symbol σ in column c and row r of C_π. If σ is less than the symbol in row r and column $\pi(e_1)$, the entry is set to 0. If σ is greater than the symbol in row r and column $\pi(e_a)$, the entry is set to a. Otherwise find the unique j for which σ is greater than the symbol in row r and column $\pi(e_j)$ but less than the symbol in row r and column $\pi(e_{j+1})$, and set the entry to j.

Now we claim that A_π is a $\mathsf{CA}_X(n; t-a, v-a, a+1)$. The verification requires demonstrating that every $(t-a)$-way interaction T is covered at least $\mu(T)$ times. So let $T = \{(f_i, \nu_i) : 1 \le i \le t-a\}$, noting that $\nu_i \in \{0, \ldots, a\}$ and f_i indexes a column of A. We form permutations of $\{e_1, \ldots, e_a\} \cup \{f_1, \ldots, f_{t-a}\}$ that are consistent with π on $\{e_1, \ldots, e_a\}$ in that these columns appear in the order prescribed by π. To do this, there are $\tau_0(T)$ columns with entry 0; place one of the $\tau_0(T)!$ orderings of these columns so that all appear before $\pi(e_1)$. There are $\tau_a(T)$ columns with entry a; place one of the $\tau_a(T)!$ orderings of these columns so that all appear after $\pi(e_a)$. For $1 \le j < a$, there are $\tau_j(T)$ columns with entry j; place one of the $\tau_j(T)!$ orderings of these columns so that all appear before $\pi(e_j)$ and after $\pi(e_{j+1})$. In this way we can form $\mu(T)$ permutations of $\{e_1, \ldots, e_a\} \cup \{f_1, \ldots, f_{t-a}\}$, each consistent with π. Because each is consistent with π, it appears in C_π. But each such appearance in C_π results in a different row of A that covers T, and hence T is indeed covered at least $\mu(T)$ times. ∎

The easiest applications of Theorem 2.1 result from using $\mathsf{CAN}(t, k, v)$ as a lower bound for $\mathsf{CAN}_X(t, k, v)$. Apply it with $a = t-1$, noting that $\mathsf{CAN}(1, k, t) = t$ for all $k \ge 1$, to recover the trivial lower bound that $\mathsf{SeqCAN}(t, v) \ge t!$. Apply it with $a = t-2$ to establish that $\mathsf{SeqCAN}(t, v) \ge (t-2)!\mathsf{CAN}(2, v-t+2, t-1)$.

Now we return to the question of when $\mathsf{SeqCAN}(t, v)$ can equal $t!$, or equivalently when a $\mathrm{DD}(t, v, v)$ can exist. Theorem 2.1 ensures that $\mathsf{SeqCAN}(t, v) \ge (t-2)!\mathsf{CAN}_X(2, v-t+2, t-1)$, so $\mathsf{SeqCAN}(t, v) = t!$ can hold only when $\mathsf{CAN}_X(2, v-t+2, t-1) \le t(t-1)$. The 2-way interaction $T = \{(c_1, \nu_1), (c_2, \nu_2)\}$ has $\mu(T) = 2$ exactly when $\nu_1 = \nu_2$ (called a *constant pair*) and has $\mu(T) = 1$ otherwise (a *nonconstant pair*). Because, for each pair of columns, $t-1$ constant pairs must be covered twice each, and $(t-1)(t-2)$ nonconstant pairs must be covered at least once each, $\mathsf{CAN}_X(2, v-t+2, t-1) \ge t(t-1)$. So we are concerned with when equality can hold.

LEMMA 2.2. *For $v \ge 4$, $\mathsf{CAN}_X(2, k, v) = v(v+1)$ only if $k \le v+2$.*

Proof. Suppose that a $\mathsf{CA}_X(v(v+1); 2, k, v)$ exists with columns indexed by $\{1, \ldots, k\}$ and symbols by $\{0, \ldots, v-1\}$. We form sets on symbols $V = (\{1, \ldots, k\} \times \{0, \ldots, v-1\}) \cup \{\infty\}$. The system of sets (*blocks*) $\mathcal{B}$ is formed as follows. For every row $(x_1, \ldots, x_k)$ of the covering array, a set $\{(i, x_i) : 1 \le i \le k\}$ is placed in $\mathcal{B}$. Then for every $1 \le i \le k$, a set $\{(i, j) : 0 \le j < v\} \cup \{\infty\}$ is placed in $\mathcal{B}$. The set system $(V, \mathcal{B})$ has $kv+1$ symbols and $v(v+1)+k$ blocks. By construction, every two different symbols appear together in exactly one block, unless the pair is of the form $\{(i, j), (i', j)\}$ corresponding to a constant pair and therefore occurring in exactly two blocks.

Now form a $(kv+1) \times (v(v+1)+k)$ matrix A, which is the *symbol-block incidence matrix*, as follows. Rows are indexed by symbols, columns by blocks. The matrix contains the entry 1 in row r and column c when symbol r appears in block c and 0 otherwise. Now examine $\mathsf{B} = \mathsf{A}\mathsf{A}^T$, which has rows and columns indexed by V. Its diagonal entries are k in entry (∞, ∞) and $v+2$ elsewhere. Its off-diagonal entries are 2 in cells indexed by $((i, j), (i', j))$ with $i \ne i'$ and 1 otherwise.

The rank of B cannot exceed the number of columns in A, namely, $v(v+1)+k$. So in order to bound k, we bound the rank of B.

Write $\mathsf{D} = \mathsf{B} - \mathsf{J}$. The rows indexed by $V \setminus \{\infty\}$ can be partitioned into v parts; a part is formed by including all rows and columns with indices $\{(i,j) : 1 \le i \le k\}$ for some j with $0 \le j < v$. Then D can be written as a block diagonal matrix with v $k \times k$ block matrices each equal to $\mathsf{X} = v\mathsf{I} + \mathsf{J}$ and a single 1×1 block matrix with entry $k-1$. Now $\det(\mathsf{D}) = (k-1)(\det(\mathsf{X}))^v$, and $\det(\mathsf{X}) = v^k(1+\frac{k}{v})$ by the matrix determinant lemma. Hence $\operatorname{rank}(\mathsf{D}) = kv+1$. Because B is obtained from D by a rank one update, $\operatorname{rank}(\mathsf{B}) \ge kv$.

Consequently, $kv \le v(v+1)+k$, or $k \le \frac{v(v+1)}{v-1}$. Thus $k \le v+2$ when $v \ge 4$, because k is an integer. ∎

This enables us to establish a substantial improvement on the bound of Mathon and Tran Van Trung [28] stated in Theorem 1.5.

THEOREM 2.3. *If $t \ge 3$ and $\mathsf{SeqCAN}(t,v) = t!$ (or equivalently, a $DD(t,v,v)$ exists), then $v \le 2t-1$.*

Proof. If $3 \le t \le 6$, this follows from Theorem 1.5. By Theorem 2.1, $t! = \mathsf{SeqCAN}(t,v) \ge (t-2)!\mathsf{CAN}_X(2, v-t+2, t-1)$ and thus $\mathsf{CAN}_X(2, v-t+2, t-1) = t(t-1)$. By Lemma 2.2, $v-t-2 \le t-1+2$ and hence $v \le 2t-1$ as required. ∎

It appears plausible that the bound should be $t+2$ rather than $2t-1$; nevertheless, the method here gives the first bound that is linear in t.

3. Upper bounds from probabilistic methods. Spencer [32] analyzed a method that selects a set of N permutations on v symbols uniformly at random; we explore this first.

LEMMA 3.1. *For fixed $t \ge 3$, $\mathsf{SeqCAN}(t,v) \le 1 + (\log(\frac{v!}{(v-t)!}))/(\log(\frac{t!}{t!-1}))$.*

Proof. A permutation of $\{0,\dots,v-1\}$ chosen uniformly at random covers any specific t-subsequence with probability $\frac{1}{t!}$ and so fails to cover it with probability $\frac{t!-1}{t!}$. Then N permutations of $\{0,\dots,v-1\}$ chosen uniformly at random and independently together fail to cover a specific t-subsequence with probability $\left(\frac{t!-1}{t!}\right)^N$. There are $\frac{v!}{(v-t)!}$ t-subsequences. When N permutations are chosen, each subsequence is *not* covered with probability $\left(\frac{t!-1}{t!}\right)^N$. Thus the expected number of uncovered t-subsequences is $\frac{v!}{(v-t)!}\left(\frac{t!-1}{t!}\right)^N$. When $\frac{v!}{(v-t)!}\left(\frac{t!-1}{t!}\right)^N < 1$, a $\mathsf{SeqCA}(N;t,v)$ must exist. This holds whenever $N > (\log(\frac{v!}{(v-t)!}))/(\log(\frac{t!}{t!-1}))$. ∎

3.1. One permutation at a time. Lemma 3.1 provides a useful upper bound on the size of completely t-scrambling sets of permutations but does not provide an effective method to find such arrays. Stein [34], Lovász [26], and Johnson [22] develop a general strategy for finding solutions to covering problems; this algorithm has been shown to lead to polynomial time methods in many combinatorial covering problems [3, 4, 7, 9, 10]. We extend that strategy here to treat sequence covering arrays.

The basic approach is greedy. Repeatedly select one permutation to add that covers a large number of as-yet-uncovered t-subsequences, until all are covered. Stein [34], Lovász [26], and Johnson [22] each suggest selecting to maximize the number of newly covered elements, but their analyses require only that the next selection cover at least the average. If after i permutations are selected there remain U_i uncovered t-subsequences, then a permutation selected uniformly at random is expected to cover $U_i\frac{1}{t!}$ t-subsequences for the first time. Provided that we select the $(i+1)$st permutation to cover at least $U_i\frac{1}{t!}$ t-subsequences for the first time, we have that $U_{i+1} \le U_i\frac{t!-1}{t!}$. Because $U_0 = \frac{v!}{(v-t)!}$, we have that $U_i \le \frac{v!}{(v-t)!}(\frac{t!-1}{t!})^i$. Choose N to be the smallest

value for which $U_N < 1$; then there must be a sequence covering array with N permutations.

This simply restates the argument of Lemma 3.1, but with two important improvements. It derandomizes the method by ensuring that appropriate selection of each permutation guarantees that the bound is met, rather than asserting the existence of a set of permutations that meets it. More importantly, the time to construct the sequence covering array is polynomial in the number of permutations and the time to select a permutation that covers at least the average.

For fixed t the number of permutations is logarithmic in v, and so an algorithm with running time polynomial in v will result if we can select the next permutation in time polynomial in v. Because the details are quite similar to earlier approaches, we merely outline how this can be done.

Suppose that $\mathcal{U}$ consists of the as-yet-uncovered t-subsequences. For $U \in \mathcal{U}$, let $\mathrm{cov}(\pi, U) = 1$ when π covers U, and $\mathrm{cov}(\pi, U) = 0$ otherwise. Let R be an r-subsequence; the r symbols in R are *fixed*, and the remaining $v - r$ are *free*. There is a set $\mathcal{P}_R$ of $\frac{v!}{r!}$ permutations that cover R. We focus on the expected number of members of $\mathcal{U}$ that are covered by a member of $\mathcal{P}_R$ chosen uniformly at random; this is

$$\mathrm{ec}(R) = \frac{r!}{v!} \sum_{\pi \in \mathcal{P}_R} \sum_{U \in \mathcal{U}} \mathrm{cov}(\pi, U).$$

The strategy is to find a sequence of subsequences $P_0, \ldots, P_v$, so that P_i is an i-subsequence, P_{i+1} covers P_i for $0 \le i < v$, symbol i is free in P_i but fixed in P_{i+1}, and $\mathrm{ec}(P_{i+1}) \ge \mathrm{ec}(P_i)$ for $0 \le i < v$. Because $\mathrm{ec}(P_0) = \frac{1}{t!}|\mathcal{U}|$, it follows that P_v is a permutation that covers at least $\frac{1}{t!}|\mathcal{U}|$ of the as-yet-uncovered t-subsequences. Given a selection of P_i, there are precisely $i+1$ candidates $\{C_1, \ldots, C_{i+1}\}$ for P_{i+1} obtained by placing symbol i in one of the $i+1$ positions of P_i. Our task is to choose one for which $\mathrm{ec}(C_j) \ge \mathrm{ec}(P_i)$, in order to set $P_{i+1} = C_j$.

A naive computation of $\mathrm{ec}(C_j)$ would enumerate members of $\mathcal{U}$ and of $\mathcal{P}_{C_j}$, but the latter may have size exponential in v. Instead, for $U \in \mathcal{U}$ let $\mathrm{ec}(U, R) = \frac{r!}{v!} \sum_{\pi \in \mathcal{P}_R} \mathrm{cov}(\pi, U)$, and observe that

$$\mathrm{ec}(R) = \sum_{U \in \mathcal{U}} \mathrm{ec}(U, R).$$

When t is fixed, $\mathcal{U}$ contains fewer than v^t subsequences, which is polynomial in v. Therefore it suffices to compute $\mathrm{ec}(U, R)$ efficiently, given a t-subsequence U and an r-subsequence R. Let τ be the number of symbols appearing in both U and R. When the τ symbols in common do not appear in the same order in U and R, $\mathrm{ec}(U, R) = 0$. Otherwise let T be the τ-subsequence that they have in common. Then $\mathrm{ec}(U, R) = \mathrm{ec}(U, T) = \frac{\tau!}{t!}$.

The key observation in selecting P_{i+1} is that $\mathrm{ec}(P_i) = \frac{1}{i+1} \sum_{j=1}^{i+1} \mathrm{ec}(C_j)$. Computing $\mathrm{ec}(C_j)$ for $1 \le j \le i+1$, and selecting P_{i+1} to be the one that maximizes $\mathrm{ec}(C_j)$, we are then sure that $\mathrm{ec}(P_{i+1}) \ge \mathrm{ec}(P_i)$.

Combining all of these arguments, we have established the next theorem.

THEOREM 3.2. *For fixed t and input v, there is an algorithm to construct a* $\mathsf{SeqCA}(N; t, v)$ *having* $N \le 1 + (\log(\frac{v!}{(v-t)!}))/(\log(\frac{t!}{t!-1}))$ *permutations in time that is polynomial in v.*

This algorithm can be easily implemented, and we report results from it in section 5. One immediate improvement results from observing that the counts of

as-yet-uncovered t-subsequences $(R_0, R_1, \ldots, R_N)$ must be integers. Hence we have that $R_{i+1} \leq \lfloor R_i \frac{t!-1}{t!} \rfloor$. In specific cases this improves on the bound, without the need to construct the sequence covering array. If, however, the sequence covering array is explicitly constructed, at each selection of P_{i+1} from P_i, we can choose P_{i+1} to maximize ec() among the $i+1$ candidates.

3.2. Greedy methods with reversals. The methods developed are greedy in that they attempt to cover the largest number of as-yet-uncovered t-subsequences. The very first permutation chosen is arbitrary; all are equally effective at coverage. Once one is selected, however, there is a genuine choice for the second. Greedy selection indicates that we should choose one that covers no t-subsequence already covered by the first. Indeed when $t = 2$, choosing the reversal of this first covers all remaining t-subsequences.

For $t \geq 3$, suppose that we have chosen $2s$ permutations $\pi_1, \ldots, \pi_{2s}$, and suppose further that π_{2i} is the reverse of π_{2i-1} for $1 \leq i \leq s$. It follows that the number of as-yet-uncovered t-subsequences covered by a permutation π is precisely the same as the number covered by the reverse of π. Yet π and its reverse never cover the same t-subsequence. Hence if the algorithm were to select π next, the reverse of π remains an equally beneficial choice immediately thereafter. Therefore a useful variant of the algorithm developed, after adding a permutation π to the array, always adds the reverse of π as well. Pursuing this, we obtain the following.

THEOREM 3.3. *For fixed t and input v, there is an algorithm to construct a* $\mathsf{SeqCA}(N; t, v)$ *having* $N \leq 2(\log(\frac{v!}{(v-t)!}))/(\log(\frac{t!}{t!-2}))$ *permutations in time that is polynomial in v.*

Naturally, we could again obtain small improvements in practice because every count of as-yet-uncovered t-subsequences is an integer.

In principle, always including reversals improves slightly on the bound (that is, Theorem 3.3 improves on Theorem 3.2). Whether this is a practical improvement remains to be seen; we return to this point.

4. Product constructions. Product (or "cut-and-paste" or "Roux-type") constructions are well studied for covering arrays; see, for example, [11, 6]. We develop a product construction for completely 3-scrambling sets of permutations. To do this, we first introduce an auxiliary property. A *signing* of a $\mathsf{CSSP}(N; t, v)$ $\mathsf{A} = (a_{ij})$ is an $N \times v$ matrix $\mathsf{S} = (s_{ij})$ with entries $\{\uparrow, \downarrow\}$. A $\mathsf{CSSP}(N; t, v)$ A is *properly signed* by an $N \times v$ matrix S with entries $\{\uparrow, \downarrow\}$. When for every set of $t-1$ distinct columns $c_1, \ldots, c_{t-1}$, each sign $s \in \{\uparrow, \downarrow\}$, and every permutation π of $1, \ldots, t-1$, there exists a row ρ of A for which, for every $1 \leq a < b < t$, the entry in cell $(\rho, c_{\pi(a)})$ is less than the entry in row $(\rho, c_{\pi(b)})$, *and* the sign $s_{\rho,c_1} = s$. A properly signed $\mathsf{CSSP}(7;3,5)$ is shown in Table 4.1.

We defer for the moment the question of how to sign a completely t-scrambling set of permutations.

4.1. Products for strength three.

THEOREM 4.1. *If a properly signed* $\mathsf{CSSP}(N; 3, v)$ *and a properly signed* $\mathsf{CSSP}(M; 3, w)$ *both exist, so does a properly signed* $\mathsf{CSSP}(N+M; 3, vw)$.

Proof. Let $\mathsf{A} = (a_{ij})$ be a $\mathsf{CSSP}(N; 3, v)$ having sign matrix $\mathsf{S} = (s_{ij})$ with columns indexed by $\{0, \ldots, v-1\}$. Let $\mathsf{B} = (b_{ij})$ be a $\mathsf{CSSP}(M; 3, w)$ having sign matrix $\mathsf{T} = (t_{ij})$ with columns indexed by $\{0, \ldots, w-1\}$. Form an array $\mathsf{C} = (c_{\rho,(i,j)})$ on $N+M$ rows and vw columns with columns indexed by $\{0, \ldots, v-1\} \times \{0, \ldots, w-1\}$. In row ρ for $1 \leq \rho \leq N$, in column (i, j), place the entry $a_{\rho i}w + j$ if $s_{\rho i} = \uparrow$, $a_{\rho i}w + (w-1-j)$

TABLE 4.1
Properly signed CSSP(7; 3, 5) – $t = 3$, $v = 5$, $N = 7$. *All signs not specified can be selected arbitrarily.*

0↑	4↑	2↓	3↓	1↓
1↓	2↓	3↓	0↑	4↑
2↓	0↑	4↑	3↓	1↓
2↓	1↓	0↑	3	4↓
2	3↓	0↓	4↑	1
4↑	1	2	3	0↑
4	3	2	0↓	1

(7;3,5)

if $s_{\rho i} = \downarrow$. In row $N + \rho$ for $1 \le \rho \le M$, in column (i, j), place the entry $b_{\rho j}v + i$ if $t_{\rho j} = \uparrow$, $b_{\rho j}v + (v - 1 - i)$ if $t_{\rho j} = \downarrow$.

To show that C is a CSSP$(N + M; 3, vw)$, we must establish that every 3-subsequence is covered. Consider three columns $(i_1, j_1), (i_2, j_2), (i_3, j_3)$, in this order. If i_1, i_2, and i_3 are all distinct, there is a row ρ of A in which $a_{\rho i_1} < a_{\rho i_2} < a_{\rho i_3}$. Then in C row ρ has the three specified columns in the chosen order. By the same token, if j_1, j_2, and j_3 are all distinct, there is a row ρ of B in which $b_{\rho j_1} < b_{\rho j_2} < b_{\rho j_3}$. Then in C row $\rho + N$ has the three specified columns in the chosen order. If $\{i_1, i_2, i_3\}$ contains only one element, $\{j_1, j_2, j_3\}$ contains three distinct elements; symmetrically, if $\{j_1, j_2, j_3\}$ contains only one element, $\{i_1, i_2, i_3\}$ contains three distinct elements. So it remains only to treat cases in which both $\{i_1, i_2, i_3\}$ and $\{j_1, j_2, j_3\}$ contain two distinct elements.

Now suppose that the three columns are $\{(i_1, j_1), (i_1, j_2), (i_2, j_1)\}$; we are concerned with the six orderings of the elements $\{c_{(i_1,j_1)}, c_{(i_1,j_2)}, c_{(i_2,j_1)}\}$, which we represent by giving the indices in the sorted order for $\{c_{(i_1,j_1)}, c_{(i_1,j_2)}, c_{(i_2,j_1)}\}$, as shown next.

	j_1	j_2
i_1	$c_{(i_1,j_1)}$	$c_{(i_1,j_2)}$
i_2	$c_{(i_2,j_1)}$	

1	2	1	3	2	1	2	3	3	1	3	2
3		2		3		1		2		1	

Table 4.2 gives the rows in which each of the six orderings is covered; we use $sgn(x - y)$ to be $\downarrow$ if $x < y$, $\uparrow$ if $x > y$. Two of the orderings are covered at least twice.

To sign C properly, assign $\uparrow$ to each entry in each of the first N rows and $\downarrow$ to each entry in each of the last M rows. □

A small example, combining two CSSP(6;3,4)s to form a CSSP(12;3,16), is shown in Table 4.3.

Use the strategy in the proof of Theorem 4.1, taking B and T from

$$\begin{pmatrix} 0\uparrow & 1\uparrow \\ 0\downarrow & 1\downarrow \\ 1\uparrow & 0\uparrow \\ 1\downarrow & 0\downarrow \end{pmatrix},$$

to establish the next theorem.

THEOREM 4.2. *If a properly signed* CSSP$(N; 3, v)$ *exists, so does a properly signed* CSSP$(N + 4; 3, 2v)$.

4.2. Signing a completely *t*-scrambling set of permutations. We first give one technique for signing that applies for all strengths.

TABLE 4.2
Verification for six orderings.

Row in C	Condition on ρ	Sign condition	Ordering
ρ	$a_{\rho i_1} < a_{\rho i_2}$	$s_{\rho i_1} = sgn(j_2 - j_1)$	$\begin{matrix}1 & 2\\ 3\end{matrix}$
ρ	$a_{\rho i_1} < a_{\rho i_2}$	$s_{\rho i_1} = sgn(j_1 - j_2)$	$\begin{matrix}2 & 1\\ 3\end{matrix}$
ρ	$a_{\rho i_1} > a_{\rho i_2}$	$s_{\rho i_1} = sgn(j_2 - j_1)$	$\begin{matrix}2 & 3\\ 1\end{matrix}$
ρ	$a_{\rho i_1} > a_{\rho i_2}$	$s_{\rho i_1} = sgn(j_1 - j_2)$	$\begin{matrix}3 & 2\\ 1\end{matrix}$
$\rho + N$	$b_{\rho j_1} < b_{\rho j_2}$	$t_{\rho j_1} = sgn(i_2 - i_1)$	$\begin{matrix}1 & 3\\ 2\end{matrix}$
$\rho + N$	$b_{\rho j_1} < b_{\rho j_2}$	$t_{\rho j_1} = sgn(i_1 - i_2)$	$\begin{matrix}2 & 3\\ 1\end{matrix}$
$\rho + N$	$b_{\rho j_1} > b_{\rho j_2}$	$t_{\rho j_1} = sgn(i_2 - i_1)$	$\begin{matrix}2 & 1\\ 3\end{matrix}$
$\rho + N$	$b_{\rho j_1} > b_{\rho j_2}$	$t_{\rho j_1} = sgn(i_1 - i_2)$	$\begin{matrix}3 & 1\\ 2\end{matrix}$

TABLE 4.3
Example: Properly signed $\mathsf{CSSP}(6;3,4)$ *and its product with itself, a* $\mathsf{CSSP}(12;3,16)$.

				3	2	1	0	4	5	6	7	8	9	10	11	15	14	13	12
				4	5	6	7	3	2	1	0	15	14	13	12	8	9	10	11
				4	5	6	7	15	14	13	12	3	2	1	0	8	9	10	11
$0\downarrow$	$1\uparrow$	$2\uparrow$	$3\downarrow$	12	13	14	15	4	5	6	7	0	1	2	3	8	9	10	11
$1\uparrow$	$0\downarrow$	$3\downarrow$	$2\uparrow$	4	5	6	7	12	13	14	15	8	9	10	11	0	1	2	3
$1\uparrow$	$3\downarrow$	$0\downarrow$	$2\uparrow$	15	14	13	12	4	5	6	7	8	9	10	11	3	2	1	0
$3\uparrow$	$1\uparrow$	$0\uparrow$	$2\uparrow$	3	4	8	15	2	5	9	14	1	6	10	13	0	7	11	12
$1\uparrow$	$3\uparrow$	$2\uparrow$	$0\uparrow$	4	3	15	8	5	2	14	9	6	1	13	10	7	0	12	11
$3\downarrow$	$1\uparrow$	$2\uparrow$	$0\downarrow$	4	15	3	8	5	14	2	9	6	13	1	10	7	12	0	11
				12	4	0	8	13	5	1	9	14	6	2	10	15	7	3	11
				4	12	8	0	5	13	9	1	6	14	10	2	7	15	11	3
				15	4	8	3	14	5	9	2	13	6	10	1	12	7	11	0

LEMMA 4.3. *Whenever a* $\mathsf{CSSP}(N;t,v)$ *exists, a properly signed* $\mathsf{CSSP}(N;t,v-1)$ *exists.*

Proof. Let $\mathsf{A} = (a_{ij})$ be a $\mathsf{CSSP}(N;t,v)$. Form an $N \times (v-1)$ array $\mathsf{S} = (s_{ij})$ with $s_{ij} = sgn(a_{ij} - a_{i,v-1})$. Form an $N \times (v-1)$ array $\mathsf{B} = (b_{ij})$ with $b_{ij} = a_{ij}$ if $a_{ij} < a_{i,v-1}$ and $b_{ij} = a_{ij} - 1$ otherwise. Then B is a $\mathsf{CSSP}(N;t,v-1)$ that is properly signed by S. ∎

Let A be a $\mathsf{CSSP}(N;t,v)$ and $\mathsf{A}_1, \mathsf{A}_2$ be arrays that partition the rows of A. When, for $i = 1, 2$, A_i is a $\mathsf{CSSP}(N_i;t-1,v)$, A is a *partitionable* CSSP.

LEMMA 4.4. *Whenever a partitionable* $\mathsf{CSSP}(N;t,v)$ *exists, a properly signed* $\mathsf{CSSP}(N;t,v)$ *exists.*

Proof. Let A be a $\mathsf{CSSP}(N;t,v)$ with partition $\mathsf{A}_1, \mathsf{A}_2$. Assign sign $\uparrow$ to every entry of A_1 and $\downarrow$ to every entry of A_2. ∎

COROLLARY 4.5. *Whenever a* $\mathsf{CSSP}(N;3,v)$ *contains a row and its reverse, it is partitionable and hence can be properly signed.*

Proof. Place the row and its reverse in A_1 and all other rows in A_2. Then A_1 is a $\mathsf{CSSP}(N;2,v)$. Moreover, A_2 is a $\mathsf{CSSP}(N;2,v)$ because for every $i, j \in \{0, \ldots, v-1\}$ with $i \neq j$, in A there are at least three rows in which the entry in column i is less than that in column j. Then A is partitionable, so apply Lemma 4.4. ∎

Lemma 4.4 provides a sufficient condition for a $\mathsf{CSSP}(N;3,v)$ to have a proper signing but considers only signings in which all entries in each row receive the same sign. Column c is *properly signed* when, for every set of $t-1$ distinct columns $c_1, c_2, \ldots, c_{t-1}$ with $c_1 = c$, each sign $s \in \{+,-\}$, and every permutation π of $1,\ldots,t-1$, there exists a row ρ of A for which, for every $1 \le a < b < t$, the entry in cell $(\rho, c_{\pi(a)})$ is less than the entry in row $(\rho, c_{\pi(b)})$, *and* the sign $s_{\rho,c} = s$. Properly signing the $N \times v$ array A is equivalent to properly signing each column of A; the important fact is that signs assigned in one column are unrelated to signs in any other, and so one can (hope to) proceed by signing each column separately.

Consider the case of strength $t = 3$. What does it mean to properly sign a specific column c? For every column i other than c we form two sets: A_i contains the row indices in which the entry in column i is larger than that in column c, and $B_i (= \{1,\ldots,N\} \setminus A_i)$ contains the row indices in which the entry in column i is smaller than that in column c. We can consider these sets as the edges of a hypergraph H on vertex set $\{1,\ldots,N\}$. Then H has $2v-2$ edges each containing at least three vertices and a proper 2-coloring of H corresponds to a proper signing of c. Lemma 4.4 and Corollary 4.5 give proper 2-colorings. In all examples that we have examined, each column can be properly signed by finding a suitable 2-coloring. Hence it is plausible that every $\mathsf{CSSP}(N;3,v)$ can be properly signed, but if this is true the proof is elusive at the moment.

5. Computational results. In [23], a simple greedy method is used to compute upper bounds on $\mathsf{SeqCAN}(t,v)$ for $t \in \{3,4\}$ and small values of v. These are reported in column `K` in Tables 5.1 and 5.2. Results from a more sophisticated greedy method by Erdem et al. [16] are reported in column `ER`. Using techniques from constraint satisfaction, in particular answer set programming, much more sophisticated search methods have been applied to strengths three and four [1, 2, 16]. Banbara, Tamura, and Inoue [1] implement an answer set programming method and show bounds for $\mathsf{SeqCAN}(3,v)$ for $v \le 80$ and for $\mathsf{SeqCAN}(4,v)$ for $v \le 23$. These bounds appear in column `BTI` in Tables 5.1 and 5.2. Results by Brain et al. [2] are reported in column `BR` in Tables 5.1 and 5.2. In [18], bounds for $t \in \{3,4,5\}$ and $v \le 10$ are reported from a method called the "bees algorithm"; these offer modest improvements on the greedy method in [23]. We do not report them for $t \in \{3,4\}$ because they are not competitive with the results in [1]; we do report them for $t = 5$ in column `BA`, because they are the only published computational results. When $t = 3$, Tarui [36] establishes by direct construction that $\mathsf{SeqCAN}(3, \binom{\lfloor q/2 \rfloor}{\lfloor q/4 \rfloor}) \le q$ for all $q \ge 4$; these are reported in column `TA`.

The bound `U` is obtained by computing the number U_i of as-yet-uncovered t-subsequences using $U_{i+1} = \lfloor \frac{t!-1}{t!} U_i \rfloor$ and terminating with the first value N for which $U_N = 0$. (This does not explicitly construct the array but rather yields a number of rows for which it can surely be produced.) In the same manner, bound $\mathtt{U_R}$ is obtained by including reversals, so that $U_{i+2} = U_i - 2\lceil \frac{1}{t!} U_i \rceil$ when i is even. The bound `D` is obtained by applying the algorithm that establishes Theorem 3.2 and that of $\mathtt{D_R}$ by applying the algorithm that establishes Theorem 3.3.

Table 5.1 reports results for $t = 3$. The theoretical results indicate that including reversals accelerates coverage, and so the bound $\mathtt{U_R}$ improves on the bound `U`. Neither is competitive with greedy bounds from [23], given by `K`. In turn these are improved upon by the greedy method from [16], given by `ER`. Implementing our greedy approach nevertheless results in useful improvement to the two earlier greedy bounds, whether reversals are included or not. It comes as no surprise that the answer set programming

TABLE 5.1
Upper bounds on $\mathsf{SeqCAN}(3, v)$.

Events	$t = 3$								
v	TA	U	$\mathtt{U}_\mathtt{R}$	K	ER	$\mathtt{D}_\mathtt{R}$	D	BTI	BR
4	8	12	12	-		8	6		
5	8	17	16	8		10	8	7	7
6	8	20	18	10		10	8	8	8
7	10	23	22	12		12	9	8	8
8	10	26	24	12		12	10	8	8
9	10	28	26	14		12	11	9	9
10	10	30	28	14	11	14	12	9	9
11	12	32	30	14		14	12	10	10
12	12	33	30	16		14	13	10	10
13	12	35	32	16		16	13	10	10
14	12	36	34	16		16	14	10	10
15	12	37	34	18		16	14	10	10
16	12	39	36	18		16	15	11	10
17	12	40	36	20		18	15	11	11
18	12	41	38	20		18	16	12	12
19	12	42	38	22		18	16	12	12
20	12	42	38	22	19	18	16	12	12
21	14	43	40	22		18	17	12	12
22	14	44	40	22		20	17	13	12
23	14	45	40	24		20	17	14	13
24	14	46	42	24		20	17	14	13
25	14	46	42	24		20	18	14	14
26	14	47	42	24		20	18	14	14
27	14	48	44	26		20	18	14	14
28	14	48	44	26		20	18	14	14
29	14	49	44	26		22	19	15	14
30	14	49	46	26	23	22	19	15	15
40	16	54	50	32	27	24	21	17	17
50	16	58	52	34	31	26	23	19	18
60	16	61	56	38	34	26	24	21	20
70	16	64	58	40	36	28	25	22	22
80	18	66	60	42	38	30	26	24	23
90	18	68	62	44		30	27		

methods from [1, 2] (BTI, BR) yield consistent improvements on all the greedy methods for strength three. However, the direct construction of Tarui [36] provides better results at this time whenever $v \geq 30$.

Perhaps the most perplexing pattern is the regularity with which D yields a better bound than does $\mathtt{D}_\mathtt{R}$. Remarkably, we consistently produce smaller sequence covering arrays when we do *not* automatically include reversals! The reasons for this are quite unclear at the present time.

Table 5.2 gives results for strengths four and five. For strength four, our improvements on the method from [23] are more dramatic than for strength three. Surprisingly, the answer set programming technique from [1] obtains a better result than our greedy methods only when $v \leq 8$. For $9 \leq v \leq 23$, our greedy method yields much smaller arrays. (For $v = 23$, we employ 98 permutations as opposed to the 112 in BTI.) A similar comparison applies with the results from [2, 16] reported in column BR. Of course, we expect that given enough time, the answer set programming techniques would improve upon our greedy bounds.

However, our methods require polynomial time in theory and are effective in practice for larger problems than those considered in [2, 1]; despite these "limitations," our methods appear to yield better results within the time available.

TABLE 5.2
Upper bounds on $\mathsf{SeqCAN}(t, v)$ *for* $t \in \{4, 5\}$.

Events	$t = 4$							$t = 5$				
v	U	$\mathtt{U}_\mathtt{R}$	K	$\mathtt{D}_\mathtt{R}$	D	BTI	BR	U	$\mathtt{U}_\mathtt{R}$	$\mathtt{D}_\mathtt{R}$	D	BA
4	24	24	24	24	24							
5	54	54	29	24	26	24		120	120	120	120	
6	79	78	38	32	34	24		294	294	148	149	159
7	98	96	50	40	41	38		437	436	198	200	212
8	114	112	56	44	47	44		552	550	242	243	271
9	128	126	68	50	52	52		648	646	282	284	329
10	140	138	72	56	57	58	55	731	728	318	322	383
11	151	148	78	60	61	65		803	800	354	356	
12	160	158	86	64	66	69		868	864	384	386	
13	169	166	92	70	71	77		926	922	416	419	
14	177	174	100	74	73	81		978	976	446	448	
15	184	180	108	78	78	84		1027	1024	470	475	
16	191	188	112	80	81	89		1072	1068	496	501	
17	197	194	118	84	84	91		1113	1110	518	521	
18	203	200	122	86	86	97		1152	1148	540	547	
19	209	204	128	90	91	100		1189	1184	560	570	
20	214	210	134	92	92	105	104	1223	1218	582	590	
21	219	214	134	96	95	104		1256	1252	600	610	
22	224	220	140	98	97	111		1286	1282	622	629	
23	228	224	146	98	99	112		1316	1310	636	646	
24	232	228	146	102	101			1344	1338	654	665	
25	236	232	152	104	104			1370	1366	674	682	
26	240	236	158	106	105			1396	1390	688	698	
27	244	240	160	108	107			1420	1416	706	715	
28	248	242	162	110	110			1444	1438	718	732	
29	251	246	166	112	111			1466	1460	734	746	
30	255	250	166	114	113		149	1488	1482	748	760	
40	283	278	198	132	128		181	1671	1664			
50	305	298	214	146	141			1811	1804			
60	322	316	238	154	151			1924	1916			
70	337	330	250	166	160			2019	2012			
80	350	342	264	174	168			2101	2092			
90	361	354	-	180	176			2173	2164			

A somewhat different pattern with respect to reversals is evident for strength four: The theoretical bound profits by including reversals throughout, but the implemented construction method appears first to benefit from reversals (for $v \leq 20$) but later no longer benefit (for $40 \leq v \leq 90$). Again the reasons for this are unclear. When $v = 90$, our methods track the coverage of 61,324,560 4-subsequences; thus, while the methods scale polynomially with v, the computations are nonetheless quite extensive. There is a $\mathsf{CSSP}(24;4,6)$ [28], but our methods do not yield fewer than 32 permutations.

For strength five, none of the published methods in [1, 16, 23] report computational results, so it is difficult to make any comments about relative accuracy. However, the answer set programming methods do appear to require substantially more storage, which limits to a degree their effective range. To apply our methods would require tracking the coverage of 78,960,960 5-subsequences for $v = 40$; despite the efficiency of our methods, a straightforward implementation encounters both storage and time limitations. The method BA [18] is not competitive with our greedy methods.

Within the range computed, including reversals improves our results. The pattern thereafter is unknown. Again, there is a $\mathsf{CSSP}(120;5,6)$ [25], but our methods do not yield fewer than 148 permutations.

6. Using the product construction. For strength three, Theorem 4.1 provides substantial improvements on the computational results from the greedy methods. We properly signed a $\mathsf{CSSP}(6;3,4)$ in Table 4.3 and a $\mathsf{CSSP}(7;3,5)$ in Table 4.1. Table 6.1 shows proper signings for further arrays from [1].

We obtain $\mathsf{CSSP}(N;3,v)$ for $(v,N) \in \{(40,15),\ (80,17),\ (128,18),\ (160,19),\ (256,20),(288,21)\}$ by using these in Theorem 4.1. These improve upon all the computational results! For example, while in [1] it is shown that $\mathsf{SeqCAN}(3,80) \le 24$ and in [2] that $\mathsf{SeqCAN}(3,80) \le 23$, here it is shown that $\mathsf{SeqCAN}(3,80) \le 17$. The examples given also provide better bounds than those of Tarui [36], but Theorem 4.1 does not outperform the direct construction asymptotically.

7. Constraints. In the testing application, it may happen that not every permutation of the events can in fact be executed; see [2, 23, 24]. It is therefore reasonable to ask how constraints on the execution order affect the number of permutations needed and how they affect the difficulty of finding a sequence covering array. We briefly consider the latter, in order to examine connections with further problems.

Let $\Sigma = \{0,\dots,v-1\}$. Let $\mathcal{C}$ be a set of subpermutations of Σ, called *constraints*. A *constrained sequence covering array* $\mathsf{SeqCA}(N;t,v,\mathcal{C})$ is a set $\Pi = \{\pi_1,\dots,\pi_N\}$ where π_i is a permutation of Σ that does not cover any subpermutation in $\mathcal{C}$, and every t-subsequence of Σ that does not cover any subpermutation in $\mathcal{C}$ is covered by at least one of the permutations $\{\pi_1,\dots,\pi_N\}$.

Even in the easiest case, when $t = 2$ and all constraints are 2-subpermutations, the nature of the problem changes dramatically. Imposing the subpermutation constraint that b cannot precede a is the same as enforcing the precedence constraint that a precede b. When the precedence constraints contain a cycle, it is impossible to meet all constraints. This can be easily checked. When the constraints are acyclic, there is a permutation that covers no constraint. However, covering *all* 2-subpermutations not in $\mathcal{C}$ requires more. Let $\mathcal{C}^r$ be the set of 2-subpermutations obtained by reversing each 2-subpermutation in $\mathcal{C}$. Suppose that a $\mathsf{SeqCA}(N;2,v,\mathcal{C})$ exists. Every permutation in the sequence covering array covers every 2-subpermutation in $\mathcal{C}^r$. Equivalently, treating $\mathcal{C}^r$ as a partial order, every permutation gives a linear extension of the partial order. When $(a,b) \in \mathcal{C}$, (b,a) must be covered by *every* permutation in the sequence covering array. When $\{(a,b),(b,a)\} \cap \mathcal{C} = \emptyset$, some but not all permutations in the sequence covering array cover (a,b)—and the rest cover (b,a). Hence the set of 2-subpermutations covered by *every* permutation in the sequence covering array is exactly $\mathcal{C}^r$. This establishes a connection with the theory of partial orders. The *dimension* of a partial order is the smallest number of linear extensions whose intersection is the partial order [37, 38]. Our discussion establishes that a $\mathsf{SeqCA}(N;2,v,\mathcal{C})$ exists if and only if the dimension of the partial order induced by $\mathcal{C}^r$ is at most N. Hence we have the next lemma.

LEMMA 7.1. *Deciding whether a* $\mathsf{SeqCA}(N;2,v,\mathcal{C})$ *exists is NP-complete, even when* $\mathcal{C}$ *is an acyclic set of* 2*-subpermutations.*

Proof. Yannakakis [41] shows that determining whether a partial order has dimension at most 3 is NP-complete. □

Brain et al. [2] establish the NP-completeness of a related problem in which the subsequences to be covered, the constraints, and the permutations allowed are all specified. Lemma 7.1 is in stark contrast with the existence of sequence covering arrays of strength two without constraints. Nevertheless, the complexity arises in determining whether a small sequence covering array exists in these cases, not in

TABLE 6.1
Small properly signed $\mathsf{CSSP}(N;3,v)$*s. Signs not shown can be chosen arbitrarily.*

8↑	5↑	6↓	2↑	3↓	7↑	1↑	4↓
1↑	2↑	8↑	4↑	5↓	7↑	6↓	3↓
6↓	5↓	7	8↑	3	2↓	4↓	1↑
5↓	7↑	4↓	3↓	8↑	1↑	6	2
3↓	7↓	4	2↓	1↑	5↓	8↑	6↓
6	2↓	1↑	3	8	7↓	4↓	5
2	7	3↓	6	4	5	1	8↑
5	1	3	8↓	4↓	2	6	7

(8;3,8)

10↑	4↓	5↑	6↑	8↓	1↑	3↑	2↓	9↑	7↑
2↑	10↑	6↑	5↑	4↓	8↑	7	1↑	9↑	3↑
8	6	1↓	5↑	10↑	3	9↑	7↓	4↓	2↓
5↑	3↓	9↓	10↓	1↑	8↑	4↑	6↓	7↓	2↓
2↓	5	7↑	4↓	8	9↓	3↓	10↑	1↑	6
6	8	1	10	7	5↓	2↓	4	3	9
9↓	10↓	8↓	3	5↓	2	4	7	1	6↑
5↓	1↑	7	3↓	2	4↓	9	8	6↓	10↓
3	5	1	2	9	7	10↓	4	6	8

(9;3,10)

2↑	14↑	15↑	5↑	8↑	3↓	1↑	11↑	9↑	7↑	13↑	12↑	16↑	6↑	4↑	10↑
14↑	8↓	3↑	6↓	7↓	11	1↓	16↑	12↓	13↓	2↑	10↓	15↓	5↓	9	4↓
12↓	16↓	4	6↑	3↑	5	7	2↓	1↑	9	15↓	11↓	8	14↓	10	13↓
12↓	2↓	13↓	4↓	3↓	14↓	9	6	11	1↑	5↓	15	10↓	7↓	16↑	8↓
16	4	10↑	5	12	9↓	11	14↓	1↓	8	13	2↑	7	15↑	6↑	3↑
6	4↑	12	15	3	1↑	11↓	13	8	14↑	10	5	2↓	9	16↓	7
2↓	15	12↓	11	6	8	13↓	5	14↓	7	3↓	4↓	9	16	10	1
5	9↑	7	14↓	16↑	13	6	10↓	12	1↓	11	2	8	3↑	4↓	15
5	6	1↓	15↑	12	16↑	10	2↑	3	13↓	4	11	9	8	7↓	14
11↑	7	8	4	15↓	5	16↑	6	14↑	12	9↑	13↑	1↑	2	3	10↑

(10;3,16)

1↑	2↑	3↑	4↑	5↑	6↑	7↑	8↑	9↑	10↓	11↑	12↓	13	14↑	15↑	16↓	17↑	18↑
4↓	14↑	11↑	10↑	6↑	9↑	13↓	7↓	17↑	1↑	18↑	3↓	2↑	12↓	16↑	5↓	8↓	15↓
4↓	17↓	3	2	18↓	16↓	14↑	7	9	13	11↓	12	10↑	8↓	1↑	6	15↓	5↑
6	13	7↓	8↓	12	10	14	4↑	16↓	1	2↓	18↑	17↓	15↑	9	11	3↑	5↑
7	18	6↓	16↓	4↓	5	1↓	17↑	13	11	10	12	14	15	8↓	2	9↑	3
8	3↓	15↓	4↓	9	17	2	16	11↓	12	14	13	10↓	7	6	1↑	5	18
8	11	3	14	6↓	15	13	5↓	2↓	16↓	17↓	1↑	18↑	7	10	9	4	12
10	7↓	16	4	5	9↓	13	15↓	1	18↑	3	17↓	2	12	14	8	6	11
13	9↑	16	14	15	11↑	7	12	5↑	4↓	2↑	1	3↓	10↓	6↓	18↑	17	8↓
18↑	4	15↑	17↑	16↑	2↓	8↑	1	11	13	9	12	10	3	6	14	7	5↓
18↓	7	2	5	4	3	17↓	10	15	11	14	12	13	1↑	16↓	9↓	8↓	6

(11;3,18)

19↑	3↑	5↑	2↑	6↑	18↑	9	10	14↑	21↑	20↑	1↑	15↓	4↑	12	22↑	16↑	7	13	8	11↑	23↑	17↑
2↓	22↑	13↓	21↑	7↑	4↓	1↑	6↓	18↓	23↓	15↓	12	8	10↑	19↑	16↑	17	11↑	20↑	3↑	5↓	14↑	9↓
1↑	21↓	6↑	7	4↓	20	16↓	18	10↓	23	3↑	13	17	9↓	11↑	14	2↑	19↓	8↑	22↑	12	15↓	5
10	12↑	5↓	15	8↓	13	23↑	17↓	22↑	11	18	14	9	3↓	4↓	1↑	16	2↑	20↓	21	19↓	6	7↓
3	17↓	22	4↓	14	2↓	7	13	1↑	10↑	9	16↑	11↓	12	23↓	8	20↑	19	18↓	15	21	5	6↑
10	1↓	19↓	15↑	3	14↑	23↓	4	12	11	8	18↓	2	21↑	16	6	17	20↑	7	5↓	9	13	22↓
20↑	13	3	17↓	18	7	14	10	22	9↓	1↓	16	11↓	21	15	8↓	4	6	2	5↓	23↑	19↓	12↑
8	18	12	3	22↓	11	14	1↑	15	6	21↓	5	10	19↓	9	13	4↓	2↓	20	17	16	7	23
21↓	2	12	15	22↑	23↓	3↓	19↓	8↓	5↓	16	17↑	1↑	20	6↑	18	13↓	7	9↑	11	14↓	4↓	10
18	10	9	14↓	7	17	8	6	5	15	16↑	13	23↑	11	4↓	12	20↓	21	1↓	22↓	3↑	2↑	19
16	10	14	9	21	12	7	23↑	13	2↑	4	19↓	20	1	22	3↓	8	17↓	18	6↑	5	15	11
15↓	16	23↑	21	12	3↑	19	17	4	9	13	1↓	18	14	5	22↓	7	11	10	8	6	20	2

(12;3,23)

determining whether a sequence covering array exists. The situation is worse when constraints have strength three.

Consider a collection $\mathcal{T}$ of ordered triples of distinct elements of Σ, and associate with (a,b,c) the constraints $\{(b,a,c),(b,c,a),(a,c,b),(c,a,b)\}$. Meeting these constraints requires that b lie between a and c, and a collection of constraints of this type forms an instance of the *betweenness problem* [5] in which one is required to

order all items so that for every triple $(a, b, c) \in \mathcal{T}$, b lies between a and c. Forming $\mathcal{C} = \{(b, a, c), (b, c, a), (a, c, b), (c, a, b) : (a, b, c) \in \mathcal{T}\}$, even finding a single permutation that covers no 3-subpermutation in $\mathcal{C}$ appears hard:

LEMMA 7.2 (see [30]). *Determining whether an instance of the betweenness problem has a solution is NP-complete.*

These complexity results suggest that constraints pose severe additional challenges in the construction of sequence covering arrays. Checking feasibility can become difficult; even when feasibility is easily checked, the minimization problem is substantially more complicated.

8. Conclusions. The close connection between sequence covering arrays and covering arrays has proved useful in establishing bounds on the sizes of sequence covering arrays. The efficient conditional expectation algorithm for generating sequence covering arrays and the product construction for strength three parallel analogous results for covering arrays. Unfortunately, while sequence covering arrays lead to covering arrays with excess coverage, additional conditions on such a covering array would be required in order to recover a sequence covering array. Hence the parallels between the extensive literature on covering arrays and the existence problem for sequence covering arrays are primarily by analogy.

We have examined numerous formulations for sequence covering arrays. In closing, we indicate one more (see also [27]). A *perfect hash family* $\mathsf{PHF}(N; k, w, t)$ is an $N \times k$ array on w symbols in which in every $N \times t$ subarray, at least one row consists of distinct symbols. Mehlhorn [29] introduced perfect hash families as an efficient tool for compact storage and fast retrieval of frequently used information; see also [14]. Stinson et al. [35] establish that perfect hash families can be used to construct separating systems, key distribution patterns, group testing algorithms, cover-free families, and secure frameproof codes. They are also used extensively in product constructions for covering arrays [8, 12, 13]. Completely t-scrambling sets of permutations can be viewed as an ordered analogue of perfect hash families in which $k = w$, no element appears twice in a row, and for every way to select t distinct columns *in order* there is a row in which the elements in these columns are in *increasing* order. In particular, a completely t-scrambling set of permutations provides a perfect hash family in which, for every set of t columns, there are at least $t!$ rows containing distinct symbols in the chosen columns, and at least one for each of the $t!$ symbol orderings. For this reason, it appears reasonable to expect that constructions for perfect hash families may also prove to be useful for sequence covering arrays.

Acknowledgments. Thanks to two anonymous referees for pointing out relevant references. Special thanks to Mutsunori Banbara and Johannes Oetsch for providing explicit solutions for small sequence covering arrays with $t = 3$.

REFERENCES

[1] M. BANBARA, N. TAMURA, AND K. INOUE, *Generating event-sequence test cases by answer set programming with the incidence matrix*, in Technical Communications of the 28th International Conference on Logic Programming (ICLP12), 2012, pp. 86–97.

[2] M. BRAIN, E. ERDEM, K. INOUE, J. OETSCH, J. PÜHRER, H. TOMPITS, AND C. YILMAZ, *Event-sequence testing using answer-set programming*, Internat. J. Advances Software, 5 (2012), pp. 237–251.

[3] R. C. BRYCE AND C. J. COLBOURN, *The density algorithm for pairwise interaction testing*, Software Testing, Verification, and Reliability, 17 (2007), pp. 159–182.

[4] R. C. Bryce and C. J. Colbourn, *A density-based greedy algorithm for higher strength covering arrays*, Software Testing Verification Reliability, 19 (2009), pp. 37–53.
[5] B. Chor and M. Sudan, *A geometric approach to betweenness*, SIAM J. Discrete Math., 11 (1998), pp. 511–523.
[6] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, *Constructing strength three covering arrays with augmented annealing*, Discrete Math., 308 (2008), pp. 2709–2722.
[7] C. J. Colbourn, *Constructing perfect hash families using a greedy algorithm*, in Coding and Cryptology, Y. Li, S. Zhang, S. Ling, H. Wang, C. Xing, and H. Niederreiter, eds., World Scientific, Singapore, 2008, pp. 109–118.
[8] C. J. Colbourn, *Covering arrays and hash families*, in Information Security and Related Combinatorics, NATO Peace and Information Security, IOS Press, Amsterdam, 2011, pp. 99–136.
[9] C. J. Colbourn, *Efficient conditional expectation algorithms for constructing hash families*, in Combinatorial Algorithms, Lecture Notes in Comput. Sci., 7056, Springer-Verlag, Berlin, 2011, pp. 144–155.
[10] C. J. Colbourn, D. Horsley, and V. R. Syrotiuk, *Strengthening hash families and compressive sensing*, J. Discrete Algorithms, 16 (2012), pp. 170–186.
[11] 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), pp. 33–57.
[12] C. J. Colbourn and J. Torres-Jiménez, *Heterogeneous hash families and covering arrays*, Contemp. Math., 523 (2010), pp. 3–15.
[13] C. J. Colbourn and J. Zhou, *Improving two recursive constructions for covering arrays*, J. Statist. Theory Practice, 6 (2012), pp. 30–47.
[14] Z. J. Czech, G. Havas, and B. S. Majewski, *Perfect hashing*, Theoret. Comput. Sci., 182 (1997), pp. 1–143.
[15] B. Dushnik, *Concerning a certain set of arrangements*, Proc. Amer. Math. Soc., 1 (1950), pp. 788–796.
[16] E. Erdem, K. Inoue, J. Oetsch, J. Pührer, H. Tompits, and C. Yilmaz, *Answer-set programming as a new approach to event-sequence testing*, in Proceedings of the 2nd International Conference on Advances in System Testing and Validation Lifecycle, Xpert Publishing Services, 2011, pp. 25–34.
[17] Z. Füredi, *Scrambling permutations and entropy of hypergraphs*, Random Structures Algorithms, 8 (1996), pp. 97–104.
[18] M. M. Z. Hazli, K. Z. Zamli, and R. R. Othman, *Sequence-based interaction testing implementation using bees algorithm*, in Proceedings of the IEEE Symposium on Computers and Informatics, 2012, pp. 81–85.
[19] S. Huang, M. B. Cohen, and A. M. Memon, *Repairing GUI test suites using a genetic algorithm*, in Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST), 2010, pp. 245–254.
[20] Y. Ishigami, *Containment problems in high-dimensional spaces*, Graphs Combin., 11 (1995), pp. 327–335.
[21] Y. Ishigami, *An extremal problem of d permutations containing every permutation of every t elements*, Discrete Math., 159 (1996), pp. 279–283.
[22] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.
[23] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, *Combinatorial methods for event sequence testing*, in Proceedings of the IEEE 5th International Conference on Software Testing, Verification and Validation (ICST), 2012, pp. 601–609.
[24] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, *Combinatorial methods for event sequence testing*, CrossTalk J. Defense Software Engineering, 25 (2012), pp. 15–18.
[25] V. I. Levenshtein, *Perfect codes in the metric of deletions and insertions*, Diskret. Mat., 3 (1991), pp. 3–20.
[26] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[27] O. Margalit, *Better bounds for event sequence testing*, in Proceedings of the 2nd International Workshop on Combinatorial Testing, 2013.
[28] R. Mathon and Tran Van Trung, *Directed t-packings and directed t-Steiner systems*, Des. Codes Cryptogr., 18 (1999), pp. 187–198.
[29] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, Berlin, 1984.

[30] J. Opatrný, *Total ordering problem*, SIAM J. Comput., 8 (1979), pp. 111–114.
[31] J. Radhakrishnan, *A note on scrambling permutations*, Random Structures Algorithms, 22 (2003), pp. 435–439.
[32] J. Spencer, *Minimal scrambling sets of simple orders*, Acta Math. Acad. Sci. Hungar., 22 (1971/72), pp. 349–353.
[33] R. P. Stanley, *Increasing and decreasing subsequences and their variants*, in Proceedings of the International Congress of Mathematicians, vol. I, Madrid, 2007, pp. 545–579.
[34] S. K. Stein, *Two combinatorial covering theorems*, J. Combin. Theory Ser. A, 16 (1974), pp. 391–397.
[35] D. R. Stinson, Tran Van Trung, and R. Wei, *Secure frameproof codes, key distribution patterns, group testing algorithms and related structures*, J. Statist. Plann. Inference, 86 (2000), pp. 595–617.
[36] J. Tarui, *On the minimum number of completely* 3*-scrambling permutations*, Discrete Math., 308 (2008), pp. 1350–1354.
[37] W. T. Trotter, Jr., *Some combinatorial problems for permutations*, in Proceedings of the 8th Southeastern Conference on Combinatorics, Graph Theory and Computing, Baton Rouge, La., 1977, Utilitas Mathematica, Winnipeg, pp. 619–632.
[38] W. T. Trotter, Jr., *Combinatorics and partially ordered sets*, in Dimension Theory, Johns Hopkins Ser. Math. Sci., Johns Hopkins University Press, Baltimore, MD, 1992.
[39] W. Wang, Y. Lei, S. Sampath, R. Kacker, D. Kuhn, and J. Lawrence, *A combinatorial approach to building navigation graphs for dynamic web applications*, in Proceedings of the 25th International Conference on Software Maintenance, 2009, pp. 211–220.
[40] W. Wang, S. Sampath, Y. Lei, and R. Kacker, *An interaction-based test sequence generation approach for testing web applications*, in 11th IEEE High Assurance Systems Engineering Symposium, 2008, pp. 209–218.
[41] M. Yannakakis, *The complexity of the partial order dimension problem*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 351–358.
[42] X. Yuan, M. B. Cohen, and A. M. Memon, *Towards dynamic adaptive automated test generation for graphical user interfaces*, in International Conference on Software Testing, Verification and Validation Workshops, 2009, pp. 263–266.
[43] X. Yuan, M. B. Cohen, and A. M. Memon, *GUI interaction testing: Incorporating event context*, IEEE Trans. Software Engrg., 37 (2011), pp. 559–574.
[44] X. Yuan and A. M. Memon, *Generating event sequence-based test cases using GUI runtime state feedback*, IEEE Trans. Software Engrg., 36 (2010), pp. 81–95.