IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 12, 2019, accepted March 12, 2019, date of publication March 22, 2019, date of current version April 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2907057

Methods to Construct Uniform Covering Arrays

JOSE TORRES-JIMENEZ[1], (Senior Member, IEEE), IDELFONSO IZQUIERDO-MARQUEZ[1], AND HIMER AVILA-GEORGE[2], (Senior Member, IEEE)
[1]Cinvestav Tamaulipas, Ciudad Victoria 87130, Mexico
[2]Department of Computer Science and Engineering, University of Guadalajara, Ameca 46600, Mexico

Corresponding author: Jose Torres-Jimenez (jtj@cinvestav.mx)

This work was supported in part by the ABACUS-CINVESTAV, CONACYT, under Grant EDOMEX-2011-COI-165873, and in part by the project: CONACyT 238469–Métodos Exactos para Construir Covering Arrays.

ABSTRACT Uniform covering arrays are covering arrays in which every column has the same alphabet. In recent years, a number of methods to construct such arrays have been developed. Here, we review several of these methods organizing them into six classes: algebraic, recursive, exact, greedy, metaheuristic, and transformations. The objective of this paper is to highlight the strategy of some representative algorithms of each class. Most of the reviewed methods are accompanied by examples and/or pseudocodes. This paper ends with a discussion about the general strengths and weaknesses of each class of methods.

INDEX TERMS Covering arrays, uniform covering arrays, methods to construct covering arrays.

I. INTRODUCTION

One option to test the functionality of a software or hardware component is to test all possible configurations of its input parameters. However, testing all configurations may require a large number of test cases. Consider for example a component with 12 parameters, each of which can take 4 different values; then, there are $4^{12} = 16,777,216$ different test cases. One option to reduce the number of test cases is to use the combinatorial designs called *covering arrays* (CAs). These designs are used in the *combinatorial testing* technique, CAs allow to test all configurations among any subset of t parameters. Combinatorial testing has proven to be an effective testing strategy [1]; the main idea in this technique is that most failures occur due to interactions among a small subset of input parameters, that is, failures occur when certain parameters take specific values.

The value of t modulates the coverage of interactions tested. If t is equal to the number of parameters then we have full coverage. However, in a series of studies conducted at the National Institute of Standards and Technology (NIST) in a wide range of domains determined that, all failures in the software products under study were due to interactions involving at most six parameters [2]–[5]. These results indicate that a failure is triggered by the interaction of a relatively small number of parameters, and therefore testing all configurations of size t is an effective way to detect failures in software products. Formally, a covering array $\mathrm{CA}(N; t, k, v)$ with *strength* t and *order* v is an array of size $N \times k$ over the symbol set $\mathbb{Z}_v = \{0, 1, \ldots, v-1\}$, such that every subarray of size $N \times t$ contains as a row each t-tuple over $\mathbb{Z}_v$ at least once. A CA of strength t ensures the coverage of all possible combinations of values among any t columns. As an example of a CA, Fig 1 shows a $\mathrm{CA}(12; 2, 7, 3)$. In this CA every subarray of $t = 2$ columns covers at least once every possible t-tuple over $\mathbb{Z}_3 = \{0, 1, 2\}$, which are the tuples (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), and (2, 2). In the subarray formed by the first two columns the first occurrence of these nine tuples is colored. Some tuples may occur more than once, but the requirement is that all of them occur at least once for every t distinct columns.

$$\mathrm{CA}(12; 2, 7, 3) = \begin{pmatrix} 0 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 2 & 2 & 1 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 & 0 & 1 & 0 \\ 2 & 2 & 0 & 1 & 0 & 1 & 1 \\ 2 & 0 & 0 & 0 & 2 & 2 & 0 \\ 1 & 2 & 1 & 2 & 2 & 2 & 1 \\ 2 & 1 & 2 & 2 & 1 & 0 & 1 \\ 1 & 0 & 2 & 1 & 2 & 0 & 2 \\ 1 & 2 & 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 & 2 & 1 & 2 \\ 2 & 1 & 1 & 1 & 0 & 2 & 2 \end{pmatrix}$$

FIGURE 1. Example of a CA.

The associate editor coordinating the review of this manuscript and approving it for publication was Roberto Pietrantuono.

CAs can be viewed as test suites where columns represent parameters of the software or hardware component under test; the order of a CA is the number of distinct values for every parameter; and the strength of a CA is the degree of coverage of interactions. CAs like the one in Fig 1 are sometimes called *uniform covering arrays* because every column has the same set of symbols. On the other hand, there exist *mixed covering arrays* (MCAs) which are denoted as $\mathrm{MCA}(N; t, k, (v_0, \ldots, v_{k-1}))$, where symbols in column $j \in \{0, \ldots, k-1\}$ come from an alphabet of cardinality v_j. For any t distinct columns $j_0, \ldots, j_{t-1}$, every t-tuple in the Cartesian product of the alphabets associated to these columns occurs at least once in the $N \times t$ subarray formed by the columns $j_0, \ldots, j_{t-1}$. Usually the exponential notation $s_0^{u_0}, \ldots, s_l^{u_l}$ is used to indicate that an MCA has u_i columns whose alphabets have cardinality s_i. In practice MCAs are used more often than uniform CAs as test suites because it is very unlikely that every parameter of the software or hardware component has the same number of distinct values. CAs have also been applied in other areas including GUI testing [6], fire accident reconstruction [7], testing effects of multiple inputs in regulating a biological system [8], and clustering business process models [9].

The covering array number $\mathrm{CAN}(t, k, v)$ is the minimum number of rows N for which exists a CA with k columns, strength t, and order v:

$$\mathrm{CAN}(t, k, v) = \min\{N : \exists\, \mathrm{CA}(N; t, k, v)\}.$$

It is very difficult to find the exact value of $\mathrm{CAN}(t, k, v)$ for general values of t, k, and v; but there are particular cases for which the exact value of $\mathrm{CAN}(t, k, v)$ is known:

- $\mathrm{CAN}(t, t+1, 2) = 2^t$ for each $t \geq 1$.
- $\mathrm{CAN}(2, k, 2) = N$, where N is the least positive integer that satisfies $\binom{N-1}{\lceil \frac{N}{2} \rceil} \geq k$ [10], [11].
- $\mathrm{CAN}(t, v+1, v) = v^t$ for v prime-power and $v > t$ [12].
- $\mathrm{CAN}(t, t+1, v) = v^t$ for v prime-power and $v \leq t$ [13].
- $\mathrm{CAN}(3, v+2, v) = v^3$ for $v = 2^n$ [12].
- $\mathrm{CAN}(v-1, v+2, v) = v^{v-1}$ when $v = 2^n$ [14] (Corollary 3.8).
- $\mathrm{CAN}(t, t+2, 2) = \lfloor \frac{4}{3} 2^t \rfloor$ for each $t \geq 1$ [15].

A CA with strength t and order v must have at least v^t rows; so a trivial *lower bound* for the covering array number is $\mathrm{CAN}(t, k, v) \geq v^t$. Some works where lower bounds of CAN are studied are [16], [17]. Similarly, a trivial *upper bound* for $\mathrm{CAN}(t, k, v)$ is v^k, which is the number of vectors of length k over $\mathbb{Z}_v$. The study of theoretical upper bounds on the size of CAs focuses on determining the value of $\mathrm{CAN}(t, k, v)$ as function of k for fixed t and v. Very recent works addressing this topic are [18], [19].

The improvement of upper bounds for $\mathrm{CAN}(t, k, v)$ is an active research topic, motivated in part by the reduction of the number of test cases when CAs are used as test suites. In the last years, the Covering Array Tables [20] have been used as the main source to report improvements in the current upper bounds of covering array numbers.

The construction of CAs is a large and active field, and several works have reviewed the different strategies to construct CAs. The works of Hartman [21] and Colbourn [22] review combinatorial and searching techniques. Lawrence *et al.* [23] presented a survey for binary CAs (CAs of order two). Kuliamin and Petukhov [24] summarized a large number of methods to construct CAs, and a special characteristic of their work is that the complexity of the algorithms is studied. Torres-Jimenez and Izquierdo-Marquez [25] described briefly some construction methods. Nie and Leung [26] studied construction methods in the context of test suite generation for combinatorial testing; also Khalsa and Labiche [27] presented the methods in the context of combinatorial testing, and reviewed methods to construct CAs of various types like CAs with variable strength and CAs with constraints. Finally, the work of Zhang *et al.* [28] explains in great detail several techniques for constructing CAs.

This paper focuses only on methods to construct uniform CAs. MCAs are barely mentioned, and also the important topics of CAs with constraints and CAs with variable strength are not addressed. We do not touch in this paper methods to construct related objects such as orthogonal arrays, sequence covering arrays, and ordered designs are not mentioned, except for orthogonal arrays of index unity.

It is difficult to classify all existing construction methods into general classes because some methods combine different strategies. However, we can classify the existing methods into two classes according to the main strategy of the method: combinatorial methods, and searching methods. In the first class are methods whose main strategy is based on combinatorial arguments; this class includes the works cited in the above list of known values of CAN. In the second class are methods that construct the CAs using computational search; these methods construct the CAs basically cell by cell.

Combinatorial methods are fast and in most cases the size of the output CA is known in advance; in addition, they provide an infinite number of CAs and currently they are the best methods to construct CAs with large values of t, k, and/or v. However, they are not completely general because for some combinations of t, k, v no combinatorial construction is known; for example there is no combinatorial method that given t can generate in all cases a CA with strength t, $t + 3$ columns, and order 2. On the other hand, searching methods can handle any values of t, k, v, but the range of these parameters is limited by execution time constraints.

To better organize the presentation of the methods covered in this survey the two high level classes are broken into more specific classes. Combinatorial methods are divided into algebraic (Section II) and recursive (Section III) techniques; and searching methods are divided into exact (Section IV), greedy (Section V), and metaheuristic (Section VI) strategies. In addition, a class of transformation methods is presented in Section VII. For each class of methods we have tried to include methods whose strategy is significantly different from the strategy of other methods in the same class. A particular characteristic of this survey is the level of detail at which algebraic and recursive methods are described; they are illustrated with examples in order to expose clearly the strategy of the method. Another valuable characteristic of this work is that very recent and powerful methods based on Linear Feedback Shift Register sequences (LFSR sequences) and covering perfect hash families (CPHF) are covered. Section VIII provides a discussion about general strengths and weaknesses of every class of methods. A discussion about which methods might improve the current upper bounds of CAN is presented in Section IX. Finally, conclusions are given in Section X.

II. ALGEBRAIC METHODS

Algebraic methods construct CAs using formulas, algebraic structures such as groups and finite fields, or operations with 1-dimensional objects like polynomials, vectors, and sequences. Methods under this class do not employ 2-dimensional designs like other CAs, orthogonal arrays, or difference matrices; this is the difference with the recursive methods presented in Section III.

Some algebraic methods yield optimal CAs; for examples, the method to construct a CA(N; 2, k, 2) by Rényi [29], Kleitman and Spencer [11], and Katona [10]; the Bush's construction [12] to generate an OA(v^t; t, $v+1$, v) where v is prime-power and $v > t$; the construction of Johnson and Entringer [15] that produces a CA($\lfloor \frac{4}{3} 2^t \rfloor$; t, $t+2$, 2) for each $t \geq 1$; and the Zero-Sum construction [13] for OA(v^t; t, $t+1$, v).

Other algebraic approaches construct CAs by concatenating a set of vectors with specific characteristics. For examples, Tang and Woo [30] constructed test cases for logic circuits using vectors of a particular set of weights; and Martinez-Pena and Torres-Jimenez [31] constructed ternary CAs ($v = 3$) using sets of rows represented by trinomial coefficients.

A. CASE T = V = 2

Rényi [29] determined the value of CAN(2, k, 2) for N even; and Kleitman and Spencer [11], and Katona [10] determined the value of CAN(2, k, 2) for every N. Given N, a CA(N; 2, k, 2) with $k = \binom{N-1}{\lceil \frac{N}{2} \rceil}$ columns is constructed by placing as columns the distinct binary vectors of length N that begin with 0 and have $\lceil \frac{N}{2} \rceil$ 1's. Theorem 1 gives the value of CAN(2, k, 2).

Theorem 1 (Kleitman and Spencer [11], Katona [10]): Let k be a positive integer, then

$$\mathrm{CAN}(2, k, 2) = \min \left\{ N : \binom{N-1}{\lceil \frac{N}{2} \rceil} \geq k \right\}.$$

Fig 2 shows an example of the construction for $k = 9$. Theorem 1 gives $N = 6$, because $N = 6$ is the smallest N such that $\binom{N-1}{\lceil \frac{N}{2} \rceil} = \binom{5}{3} = 10$ is greater than or equal to $k = 9$. The nine columns of the CA(6; 2, 9, 2) are 9 of the 10 binary vectors of length 6 and weight $\lceil \frac{6}{2} \rceil = 3$ that begin with zero.

$$\mathrm{CA}(6;2,9,2) = \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 1&1&1&0&1&1&0&1&0 \\ 1&1&0&1&1&0&1&0&1 \\ 1&0&1&1&0&1&1&0&0 \\ 0&1&1&1&0&0&0&1&1 \\ 0&0&0&0&1&1&1&1&1 \end{pmatrix}$$

FIGURE 2. Example of the construction for $t = v = 2$.

B. BUSH'S CONSTRUCTION

An orthogonal array OA$_\lambda$(N; t, k, v) is an array of size $N \times k$ where every $N \times t$ submatrix covers exactly λ times each t-tuple over $\mathbb{Z}_v$. If $\lambda = 1$ then the OA has index unity and it is omitted in the notation.

The Bush's construction [12] generates OAs of index unity for given values of v and t, when v is a prime-power and $v > t$. Let e_j, $0 \leq j \leq v-1$, be the v elements of $\mathbb{F}_v$ (the finite field with v elements) and consider the distinct polynomials $y_i(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1 x + a_0$ with coefficients in $\mathbb{F}_v$; there are v^t of such polynomials since every coefficient can take any of the v values. Then, we construct an array M of size $v^t \times v$ and label the rows from 0 to $v^t - 1$ and label the columns from 0 to $v - 1$. The value u that is assigned to the cell $M_{i,j}$ is $y_i(e_j) = e_u$; then M is an OA(v^t; t, v, v). One extra column is added to M by assigning the value u to those rows associated with a polynomial whose leading coefficient is e_u.

	e_0	e_1	e_2	e_3	
$y_0(x) = e_0x + e_0$	0	0	0	0	0
$y_1(x) = e_0x + e_1$	1	1	1	1	0
$y_2(x) = e_0x + e_2$	2	2	2	2	0
$y_3(x) = e_0x + e_3$	3	3	3	3	0
$y_4(x) = e_1x + e_0$	0	1	2	3	1
$y_5(x) = e_1x + e_1$	1	0	3	2	1
$y_6(x) = e_1x + e_2$	2	3	0	1	1
$y_7(x) = e_1x + e_3$	3	2	1	0	1
$y_8(x) = e_2x + e_0$	0	2	3	1	2
$y_9(x) = e_2x + e_1$	1	3	2	0	2
$y_{10}(x) = e_2x + e_2$	2	0	1	3	2
$y_{11}(x) = e_2x + e_3$	3	1	0	2	2
$y_{12}(x) = e_3x + e_0$	0	3	1	2	3
$y_{13}(x) = e_3x + e_1$	1	2	0	3	3
$y_{14}(x) = e_3x + e_2$	2	1	3	0	3
$y_{15}(x) = e_3x + e_3$	3	0	2	1	3

FIGURE 3. Bush's construction for $v = 4$ and $t = 2$.

As an example of the construction consider $v = 4$ and $t = 2$. The elements of $\mathbb{F}_4$ are $0x+0$, $0x+1$, $x+0$, $x+1$, or in simplified form 0, 1, x, $x+1$. Denote them by e_0, e_1, e_2, e_3 respectively. Because $t = 2$, polynomials $y_i(x)$ are of the form $y_i(x) = a_1x + a_0$ where $a_0, a_1 \in \mathbb{F}_4$. Fig 3 shows the construction of OA(16; 2, 5, 4). The value of the entry (11, 3) is given by $y_{11}(e_3) = e_2e_3 + e_3$. The product e_2e_3 is equal to $(x)(x+1) = x^2 + x$, which using the primitive element

$x+1$ and taking the modulo 2 of the coefficients is reduced to $x^2+x=(x+1)+x=2x+1=0x+1=1$. The final result is given by $1+e_3$, which is equal to $1+(x+1)=x+2=x+0=x$. The element x is the element e_2 of $\mathbb{F}_4$, so the value of the entry $(11,3)$ is 2.

The construction of OAs of index unity can be done using the sum and product tables of $\mathbb{F}_v$. When $v \le t$ an OA$(v^t; t, t+1, v)$ can be constructed using the *Zero-Sum* construction [13].

C. JOHNSON-ENTRINGER CONSTRUCTION

The construction of Johnson and Entringer [15] gives an optimal CA$(\lfloor \frac{4}{3}2^t \rfloor; t, t+2, 2)$ for each strength $t \ge 1$. Let $|u|$ be the weight of the binary vector u, and let $k=t+2$. The 2^k binary vectors are partitioned into three sets V_k^0, V_k^1, V_k^2, where for $j=0,1,2$ the set V_k^j contains the binary vectors u of length k such that $|u| \equiv j \pmod 3$. The three sets V_k^0, V_k^1, V_k^2 are CAs of strength t and $k=t+2$ columns, and at least one of them has $\lfloor \frac{4}{3}2^t \rfloor$ rows.

$$\begin{pmatrix}
0&0&0&0&0&1\\
0&0&0&0&1&0\\
0&0&0&1&0&0\\
0&0&1&0&0&0\\
0&0&1&1&1&1\\
0&1&0&0&0&0\\
0&1&0&1&1&1\\
0&1&1&0&1&1\\
0&1&1&1&0&1\\
0&1&1&1&1&0\\
1&0&0&0&0&0\\
1&0&0&1&1&1\\
1&0&1&0&1&1\\
1&0&1&1&0&1\\
1&0&1&1&1&0\\
1&1&0&0&1&1\\
1&1&0&1&0&1\\
1&1&0&1&1&0\\
1&1&1&0&0&1\\
1&1&1&0&1&0\\
1&1&1&1&0&0
\end{pmatrix}
\qquad
\begin{pmatrix}
0&0&0&0&1&1\\
0&0&0&1&0&1\\
0&0&0&1&1&0\\
0&0&1&0&0&1\\
0&0&1&0&1&0\\
0&0&1&1&0&0\\
0&1&0&0&0&1\\
0&1&0&0&1&0\\
0&1&0&1&0&0\\
0&1&1&0&0&0\\
0&1&1&1&1&1\\
1&0&0&0&0&1\\
1&0&0&0&1&0\\
1&0&0&1&0&0\\
1&0&1&0&0&0\\
1&0&1&1&1&1\\
1&1&0&0&0&0\\
1&1&0&1&1&1\\
1&1&1&0&1&1\\
1&1&1&1&0&1\\
1&1&1&1&1&0
\end{pmatrix}$$

FIGURE 4. CAs of size $\lfloor \frac{4}{3}2^4 \rfloor$ produced by the Johnson-Entringer construction for $t=4$.

For example, for $t=4$ we have $k=t+2=6$, and the CAs given by the sets V_6^1 and V_6^2 are shown in Fig 4. In this case there are 64 binary vectors of length 6, and the sets V_6^0, V_6^1, V_6^2 contains respectively 22, 21, and 21 vectors. The value $\lfloor \frac{4}{3}2^t \rfloor$ is equal to $\lfloor \frac{4}{3}2^4 \rfloor = \lfloor 21.3333 \rfloor = 21$, which is the number of rows of the CA$(21; 4, 6, 2)$ given by the sets V_6^1 and V_6^2.

D. BINOMIAL COEFFICIENTS

The method of Torres-Jimenez *et al.* [32] construct binary CAs by juxtaposing subsets of binary rows represented by binomial coefficients. In this method a binomial coefficient $\binom{n}{r}$ is used to represent the set of vectors of length n that are formed by r ones and $n-r$ zeros. For $k \ge 2$ the vertical juxtaposition of the $k+1$ subsets of rows given by the binomial coefficients $\binom{k}{0}, \binom{k}{1}, \ldots, \binom{k}{k}$ generates an OA$(2^k; k, k, 2)$. However, for $t < k$ it is not necessary to juxtapose all $k+1$ binomial coefficients, only a proper subset is required. The problem is to determine the set of binomial coefficients that produces the smaller CA for the given values of k and t.

By means of a Branch & Bound algorithm the authors derived the following equation, which for given values of k and t provides the binomial coefficients to construct the smaller CA that can be constructed with binomial coefficients:

$$\sum_{j=0}^{\left\lfloor \frac{k-(\lfloor \frac{t}{2} \rfloor \bmod (k-t+1))}{k-t+1} \right\rfloor} \binom{k}{(k-t+1)j + (\lfloor \frac{t}{2} \rfloor \bmod (k-t+1))}$$

The summation of binomial coefficients means to juxtapose vertically the sets of rows represented by the binomial coefficients. As an example consider $k=10$ and $t=6$. In this case $k-t+1=5$; thus, the expression $(\lfloor \frac{t}{2} \rfloor \bmod (k-t+1))$ evaluates to $(\lfloor \frac{6}{2} \rfloor \bmod 5) = (3 \bmod 5) = 3$, and so $\left\lfloor \frac{k-(\lfloor \frac{t}{2} \rfloor \bmod (k-t+1))}{k-t+1} \right\rfloor = \lfloor \frac{10-3}{5} \rfloor = 1$. Therefore, the CA for $k=10$ and $t=6$ is given by $\sum_{j=0}^{1} \binom{10}{5j+3} = \binom{10}{3} + \binom{10}{8}$. The rows of the resulting CA$(165; 6, 10, 2)$ are the $\binom{10}{3} = 120$ rows of weight 3 and the $\binom{10}{8} = 45$ rows of weight 8.

E. GROUP CONSTRUCTION AND STARTER VECTORS

The method of Chateauneuf and Kreher [33] constructs a CA$(N; 3, k, v)$ from an $l \times k$ matrix M over a set Ω of v symbols and from a group G acting on Ω. The number of rows of the CA is $N = l|G| + v$. Denote by M^g the matrix whose (i,j) element is the image under g of the (i,j) element of M. The method constructs a matrix $M^G = [M^g : g \in G]$ by juxtaposing vertically the $|G|$ matrices M^g. Now, let C be the $|\Omega| \times k$ matrix that has a constant row for every element of Ω. With appropriate M and G the juxtaposition of M^G and C is a CA$(N; 3, k, v)$.

Meagher and Stevens [34] extended the idea of Chateauneuf and Kreher [33]. The new construction is based on selecting a subgroup of the symmetric group of a set of v symbols, $G < Sym_v$, and in finding a starter vector $\alpha \in \mathbb{Z}_v^k$. From the circular rotations of the starter vector a cyclic matrix M is constructed; after that, G acting on M produces $|G|$ matrices that are concatenated to form an array Z. A small array C may be added to Z to complete the covering conditions.

Fig 5 shows an example of how this construction works. Let $G = \{e, (12)\} < Sym_3$ be a subgroup of the symmetric group of the set $\mathbb{Z}_3 = \{0, 1, 2\}$, and let $\alpha = (0, 1, 1, 1, 2) \in \mathbb{Z}_3^5$. Construct the cyclic matrix M (Fig 5(a)) from the rotations of α. The elements of G acting on M produce the arrays shown in Fig 5(b). Element e is the identity permutation, so $M_e = M$, and element (12) is the permutation that changes 1's by 2's and 2's by 1's. The small vector

(a)

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

(b)

$$M_e = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix}$$

$$M_{(12)} = \begin{pmatrix} 0 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 2 & 2 \\ 2 & 1 & 0 & 2 & 2 \\ 2 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 1 & 0 \end{pmatrix}$$

(c)

$$Z = \left(\begin{array}{ccccc} 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \\ \hline 0 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 2 & 2 \\ 2 & 1 & 0 & 2 & 2 \\ 2 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

FIGURE 5. **Example of the group construction. (a) Circular matrix M constructed from the starter vector $\alpha = (0, 1, 1, 1, 2)$. (b) The elements of G acting on M produce M_e and $M_{(12)}$. (c) CA(11; 2, 5, 3) is constructed by juxtaposing M_e, $M_{(12)}$, and $C = (0\ 0\ 0\ 0\ 0)$.**

$C = (0\ 0\ 0\ 0\ 0)$ is needed to ensure the coverage of all pairs. The CA(11; 2, 5, 3) is constructed by juxtaposing the arrays M_e, $M_{(1,2)}$, and C, as shown in Fig 5(c).

Lobb *et al.* [35] developed a generalization of this method for strength-two CAs that allows any number of fixed points. Another construction using groups was developed by Akhtar *et al.* [36]; in this case the constructed CAs have strength four and order three.

F. CYCLOTOMY

Colbourn [37] introduced several constructions using cyclotomic vectors. For a given order v, let q be a prime-power such that $q \equiv 1 \pmod{v}$, and let ω be a primitive element of $\mathbb{F}_q$. In this construction q is the number of columns of the CA. Form a cyclotomic vector $\mathbf{x}_{q,v} = (x_i : i \in \mathbb{F}_q) \in \mathbb{F}_q^q$ by making $x_0 = 0$ and $x_i = j \bmod v$ when $i = \omega^j$. From $\mathbf{x}_{q,v}$ construct a $q \times q$ cyclotomic matrix $A_{q,v} = (a_{ij})$ by setting $a_{ij} = x_{j-i}$. The cyclotomic matrix $A_{q,v}$ is a CA of strength t when $q > t^2 v^{4t}$. For $v = 2$, $A_{q,2}$ is a CA of strength t when $q > t^2 2^{2t-2}$.

Given a matrix $M = (m_{ij})$ over $\mathbb{Z}_v$, let $M + s$ be the matrix whose (i, j) entry is $m_{ij}+s$. From $A_{q,v}$ construct a $vq \times q$ matrix $B_{q,v}$ by juxtaposing vertically the v matrices $\{A_{q,v} + c \,|\, 0 \leq c < v\}$. For $t \geq 3$ this matrix $B_{q,v}$ is a CA of strength t when $q > (t-1)^2 v^{2t-2}$.

Sometimes the matrices $A_{q,v}$ and $B_{q,v}$ are CAs of strength t for values of q smaller than the above bounds. As an example of this, Fig 6 shows a cyclotomic matrix with $v = 2$, $q = 11$, and $t = 2$; this value of q is smaller than $t^2 2^{2t-2} = (4)(2^2) = 16$.

Cyclotomic vector

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Cyclotomic matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

FIGURE 6. **A cyclotomic vector and the cyclotomic matrix constructed from it. The matrix is a CA(11; 2, 11, 2).**

G. LFSR SEQUENCES

For any order v that is prime or prime-power the method developed by Raaphorst *et al.* [38] constructs a CA of strength $t = 3$ and $k = v^2 + v + 1$ columns using linear feedback shift register sequences (LFSR sequences) over $\mathbb{F}_v$. Let $f(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} + x^n$ be a polynomial of degree n and coefficients in $\mathbb{F}_v$, and let $I = (b_0, b_1, \ldots, b_{n-1})$ be an n-tuple over $\mathbb{F}_v$. An LFSR sequence with characteristic polynomial f and initial values I is a sequence $S(f, I) = (s_0, s_1, s_2, \ldots)$ defined as:

- $s_i = b_i$ if $0 \leq i < n$.
- $s_i = c_{n-1} s_{i-1} - c_{n-2} s_{i-2} - \cdots - c_1 s_{i-(n-1)} - c_0 s_{i-n}$ if $i \geq n$.

If f is primitive and I is nonzero, then $S(f, I)$ is a sequence with period $v^n - 1$ (this is the maximum period). Let $C_i^r(S)$ denote the subinterval of S of length r beginning at position i. Construct an array M of size $(v^3 - 1) \times k$ such that the $v^3 - 1$ rows of M are respectively $C_0^k(S), C_1^k(S), \ldots, C_{v^3-2}^k(S)$. Next, construct M' from M by reversing the rows of M. The vertical juxtaposition of M, M', and a row of k zeros gives a CA$(2v^3 - 1; 3, v^2 + v + 1, v)$. This method was extended to strengths $t \geq 4$ by Tzanakis *et al.* [39].

As an example consider $v = 2$. A primitive polynomial of degree 3 with coefficients in $\mathbb{F}_2$ is $f(x) = x^3 + x + 1$, and a nonzero tuple is $I = (0, 1, 0)$. In this case $s_0 = 0$, $s_1 = 1$, $s_2 = 0$, and $s_i = -(0)(s_{i-1}) - (1)(s_{i-2}) - (1)(s_{i-3})$ for $i \geq 3$. In this last expression the first factors of the terms are respectively the coefficient of x^2, the coefficient of x, and the independent term in $f(x) = x^3 + x + 1$. The evaluation of s_i can be transformed to $-[(0)(s_{i-1}) + (1)(s_{i-2}) + (1)(s_{i-3})]$, where the minus sign indicates to take the additive inverse of the result. In $\mathbb{F}_2$ the additive inverse of 0 is 0, and of 1 is 1, so we can ignore the minus sign. Fig 7(a) shows the

(a)

i	s_i	LFSR
0	0	0
1	1	01
2	0	010
3	$0(0)+1(1)+1(0)=0+1+0=1$	0101
4	$0(1)+1(0)+1(1)=0+0+1=1$	01011
5	$0(1)+1(1)+1(0)=0+1+0=1$	010111
6	$0(1)+1(1)+1(1)=0+1+1=0$	0101110

(b)

$$\begin{pmatrix} 0&1&0&1&1&1&0\\ 1&0&1&1&1&0&0\\ 0&1&1&1&0&0&1\\ 1&1&1&0&0&1&0\\ 1&1&0&0&1&0&1\\ 1&0&0&1&0&1&1\\ 0&0&1&0&1&1&1\\ \hline 0&1&1&1&0&1&0\\ 0&0&1&1&1&0&1\\ 1&0&0&1&1&1&0\\ 0&1&0&0&1&1&1\\ 1&0&1&0&0&1&1\\ 1&1&0&1&0&0&1\\ 1&1&1&0&1&0&0\\ \hline 0&0&0&0&0&0&0 \end{pmatrix}$$

FIGURE 7. Construction of CA(15; 3, 7, 2) from the LFSR sequence $S = (0, 1, 0, 1, 1, 1, 0, \ldots)$. (a) $S(x^3 + x + 1, (0, 1, 0))$. (b) CA (15;3,7,2).

evaluation of s_3, s_4, s_5, and s_6. Because the period of the sequence is $2^3 - 1 = 7$ we have $s_i = s_{i-7}$ for $i \geq 7$; thus, $S(f, I) = (0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, \ldots)$. The matrix M is constructed as follows: the first row of M are elements 0 to 6 of S, the second row of M are elements 1 to 7 of S, and so on until the seventh row of M are elements 6 to 12 of S. Fig 7(b) shows the CA(15; 3, 7, 2) given by S; the first part of the CA is the matrix M, the second part M' is obtained by reversing the rows of M, and the third part is a constant row of zeros.

H. DIRECT CONSTRUCTION OF CPHFs

The method of Sherwood *et al.* [40] constructs CAs of order v prime-power and strength t using column vectors that are formed by concatenating permutations of $(0, 1, \ldots, v-1)$. Let $(\beta_0^{(i)}, \beta_1^{(i)}, \ldots, \beta_{t-1}^{(i)})$ be the base v representation of $i \in \{0, 1, \ldots, v^t - 1\}$; that is, $i = \beta_0^{(i)} + v^1\beta_1^{(i)} + \cdots + v^{t-1}\beta_{t-1}^{(i)}$ where every $\beta_j^{(i)} \in \{0, 1, \ldots, v-1\}$. For each $(t-1)$-tuple $(h_1, h_2, \ldots, h_{t-1})$ over $\mathbb{F}_v$, a *permutation vector* $\overrightarrow{(h_1, h_2, \ldots, h_{t-1})}$ of length v^t is the vector that has the symbol $\beta_0^{(i)} + (h_1 \times \beta_1^{(i)}) + (h_2 \times \beta_2^{(i)}) + \cdots + (h_{t-1} \times \beta_{t-1}^{(i)})$ in position i for $0 \leq i \leq v^t - 1$.

A tuple of t permutation vectors is called *covering* if the array of size $v^t \times t$ having those permutation vectors as columns is an OA$(v^t; t, t, v)$; if the array is not an OA then the tuple is *noncovering*. A *perfect hash family* PHF$(n; k, q, t)$ is an array of size $n \times k$ with entries from $\mathbb{Z}_q$ such that every subset of t distinct columns has at least one row where all entries are distinct. To produce CAs, the method requires a PHF$(n; k, q, t)$ with $q = v^{t-1}$ such that every set of t columns has at least one row that is a covering tuple of t permutation vectors. A PHF with this additional property is called a *covering perfect hash family* and it is denoted by CPHF$(n; k, v^{t-1}, t)$. Replacing the symbols of a CPHF$(n; k, v^{t-1}, t)$ by their corresponding permutation vectors produces a CA$(n \cdot v^t; t, k, v)$, from which a CA$(n \cdot (v^t - v) + v; t, k, v)$ is obtained by deleting the repeated constant rows.

In the work of Torres-Jimenez and Izquierdo-Marquez [41] the concept of permutation vectors was extended to vectors generated by t-tuples $(h_0, h_1, \ldots, h_{t-1})$ over $\mathbb{F}_v$. An *extended permutation vector (EPV)* $\overrightarrow{(h_0, h_1, \ldots, h_{t-1})}$ of length v^t is the vector that has the symbol $(h_0 \times \beta_0^{(i)}) + (h_1 \times \beta_1^{(i)}) + (h_2 \times \beta_2^{(i)}) + \cdots + (h_{t-1} \times \beta_{t-1}^{(i)})$ in position i for $0 \leq i \leq v^t - 1$. For every prime-power order v and for strength $t = 3$ there exists a CPHF$(2; v^2 - v + 3, v^3, 3)$ formed by EPVs that are length-3 subintervals of a LFSR sequence. This CPHF generates a CA$(2v^3 - v; 3, v^2 - v + 3, v)$. The LFSR sequence is constructed with a primitive polynomial of degree 3 over $\mathbb{F}_v$ and with initial values $I = (0, 0, 1)$.

For $v = 3$ the LFSR sequence $S(f, I)$ constructed with $f(x) = x^3 + 2x + 1$ and $I = (0, 0, 1)$ is $S = (0, 0, 1, 0, 1, 2, 1, 1, 2, 0, 1, 1, 1, 0, 0, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 2, \ldots)$. By taking the length-3 subintervals from the second element of S we obtain the CPHF$(2; 13, 3^3, 3)$ shown at the bottom of this page.

Here the EPVs $\overrightarrow{(h_0, h_1, h_2)}$ over $\mathbb{F}_3$ are written as $h_0h_1h_2$. By removing the columns with exactly one EPV whose first element is 0 and by replacing the EPVs whose first nonzero element is not 1 by its isomorphic EPV whose first nonzero element is 1, the result is the following CPHF$(2; 9, 3^3, 3)$:

$$\begin{pmatrix} 010 & 101 & 121 & 112 & 120 & 102 & 111 & 100 & 001 \\ 001 & 100 & 111 & 102 & 120 & 112 & 121 & 101 & 010 \end{pmatrix}$$

Replacing each EPV of CPHF$(2; 9, 3^3, 3)$ by its corresponding column vector of length $v^t = 3^3 = 27$ results in a CA$(54; 3, 9, 3)$. In this CA there are $v = 3$ repeated rows, which are deleted to obtain the final result CA$(51; 3, 9, 3)$.

III. RECURSIVE METHODS

Recursive methods construct new CAs from smaller ones. The proof of the theorem CAN$(3, 2k, 2) \leq$ CAN$(3, k, 2) +$ CAN$(2, k, 2)$ taken from the doctoral thesis of Roux [42]

$$\begin{pmatrix} 010 & 101 & 012 & 121 & 211 & 112 & 120 & 201 & 011 & 111 & 110 & 100 & 002 \\ 002 & 100 & 110 & 111 & 011 & 201 & 120 & 112 & 211 & 121 & 012 & 101 & 010 \end{pmatrix}$$

gives a CA with $2k$ columns using two CAs with k columns as ingredients. Some generalizations of the Roux theorem appear in Chateauneuf and Kreher [33]; and more generalizations appear in Cohen *et al.* [43], Colbourn *et al.* [44], and Martirosyan and Trung [45]. These constructions are known as *Roux-type Constructions*. Other approaches take smaller CAs as inputs as in [46], [47]; or employ other combinatorial designs such as ordered designs [43] and difference matrices [48]. Another class of recursive methods construct a base array whose elements will be replaced by columns of another array [21].

$$Z = \left(\begin{array}{ccc|ccc|c|ccc}
x_{11} & \dots & x_{1k} & x_{11} & \dots & x_{1k} & \dots & x_{11} & \dots & x_{1k} \\
x_{21} & \dots & x_{2k} & x_{21} & \dots & x_{2k} & \dots & x_{21} & \dots & x_{2k} \\
\vdots & & & \vdots & & & \dots & \vdots & & \\
x_{N_11} & \dots & x_{N_1k} & x_{N_11} & \dots & x_{N_1k} & \dots & x_{N_11} & \dots & x_{N_1k} \\
\hline
y_{11} & \dots & y_{11} & y_{12} & \dots & y_{12} & \dots & y_{1l} & \dots & y_{1l} \\
y_{21} & \dots & y_{21} & y_{22} & \dots & y_{22} & \dots & y_{2l} & \dots & y_{2l} \\
\vdots & & & \vdots & & & \dots & \vdots & & \\
y_{N_21} & \dots & y_{N_21} & y_{N_22} & \dots & y_{N_22} & \dots & y_{N_2l} & \dots & y_{N_2l}
\end{array}\right)$$

FIGURE 8. Structure of the product of two covering arrays CA(N_1; 2, k, v) and CA(N_2; 2, l, v).

A. PRODUCT OF CAs

Given two covering arrays $X = \mathrm{CA}(N_1; 2, k, v)$ and $Y = \mathrm{CA}(N_2; 2, l, v)$ the product construction reported by Colbourn *et al.* [46] generates a covering array $Z = \mathrm{CA}(N_1 + N_2; 2, kl, v)$ as follows: place l copies of X side by side to form an array with N_1 rows and kl columns; next, for $1 \le i \le l$ place below the i-th copy of X an array of size $N_2 \times k$ formed by replicating k times the i-th column of Y. As shown in Fig 8 the first N_1 rows of Z are l copies of X, and below each copy of X is the same column of Y. Any two distinct columns of Z contain two distinct columns of X or two distinct columns of Y, and therefore Z is a $\mathrm{CA}(N_1 + N_2; 2, kl, v)$. Fig 9 shows the construction of CA(9; 2, 12, 2) using as ingredients CA(5; 2, 4, 2) and CA(4; 2, 3, 2).

Colbourn *et al.* [46] introduced a special type of CA called *partitioned covering array* (*PCA*), which is denoted as PCA(N; 2, (k_1, k_2), v). Fig 10 shows the PCA structure. Arrays A_1 and A_2 have size $(N - v) \times k_1$ and $(N - v) \times k_2$ respectively; X is an array of size $v \times k_2$; and P is an array of size $v \times k_1$ where every column is a permutation of the symbols $\{0, 1, \dots, v - 1\}$.

Given that every column of P is a permutation of the symbols $\{0, 1, \dots, v - 1\}$, every column of P can be transformed to the identity permutation, that is, to the column vector $(0, 1, \dots, v - 1)$. In addition, if all elements of X are zero then the CA is an SCA(N; 2, (k_1, k_2), v).

Theorem 2 (Colbourn et al. [46]): If a PCA(N; 2, (k_1, k_2), v) and an SCA(M; 2, (l_1, l_2), v) both exist, then a PCA($N + M - v$; 2, (k_1l_1, $k_1l_2 + k_2l_1$), v) also exists.

Let A_1, A_2, D, and X be the partitions of the PCA; and let B_1, B_2, D, and O be the partitions of the SCA; the PCA given

$$\begin{array}{cc}
\mathrm{CA}(5;2,4,2) & \mathrm{CA}(4;2,3,2) \\
\begin{pmatrix} 1&1&1&0\\ 1&1&0&1\\ 1&0&1&1\\ 0&1&1&1\\ 0&0&0&0 \end{pmatrix} &
\begin{pmatrix} 1&1&0\\ 1&0&1\\ 0&1&1\\ 0&0&0 \end{pmatrix}
\end{array}$$

$$\mathrm{CA}(9;2,12,2)$$
$$\left(\begin{array}{cccc|cccc|cccc}
1&1&1&0&1&1&1&0&1&1&1&0\\
1&1&0&1&1&1&0&1&1&1&0&1\\
1&0&1&1&1&0&1&1&1&0&1&1\\
0&1&1&1&0&1&1&1&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{array}\right)$$

FIGURE 9. Example of the product of two CAs.

$$\left(\begin{array}{c|c} A_1 & A_2 \\ \hline P & X \end{array}\right)$$

FIGURE 10. Structure of a partitioned covering array (PCA).

$$\left(\begin{array}{c|c|c} A_1 \otimes B_1 & A_2 \otimes B_1 & A_1 \otimes B_2 \\ \hline D & k_1\,X & O \end{array}\right)$$

FIGURE 11. Structure of the product of a PCA and an SCA.

by Theorem 2 has the structure shown in Fig 11. Products $A_i \otimes B_j$ are computed as explained at the beginning of the section; D is an array of identity permutations; the array $k_1\,X$ is formed by k_1 copies of X placed side by side; and O is an array of all zeros.

A generalization of the product construction using the *profile* of a CA (the number of positions in every column that can be modified without affecting the coverage of the CA) was developed by Colbourn and Torres-Jimenez [49].

B. POWER (K^n)

Hartman [21] describes a recursive construction to square the number of columns of a CA(N; t, k, v). The number of rows of the resulting CA is computed based on the Turan number $T(t, v)$, which is defined as the number of edges in the Turan graph with t vertices partitioned into v subsets.

Theorem 3 (Squaring Covering Arrays [21]): If $\mathrm{CAN}(t, k, v) = N$ and there exist $T(t, v) - 1$ mutually orthogonal Latin squares of side k (or equivalently $\mathrm{CAN}(2, T(t, v) + 1, k) = k^2$) then $\mathrm{CAN}(t, k^2, v) \le N(T(t, v) + 1)$.

$$\underset{\mathrm{CA}(N;t,k,v)}{\left(\begin{array}{c|c|c|c} X^0 & X^1 & \dots & X^{k-1} \end{array}\right)} \qquad \underset{\mathrm{OA}(k^2;2,T(t,v)+1,k)}{\left(\begin{array}{c} y_{ij} \end{array}\right)}$$

$$\underset{\mathrm{CA}(N(T(t,v)+1);t,k^2,v)}{\left(\begin{array}{c} z_{ij} = X^{y_{ij}} \end{array}\right)}$$

FIGURE 12. **The construction of Theorem 3.**

$$\underset{\mathrm{CA}(4;2,3,2)}{\left(\begin{array}{c|c|c} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array}\right)} \qquad \underset{\mathrm{OA}(9;2,2,3)^T}{\left(\begin{array}{ccccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{array}\right)}$$

$$\underset{\mathrm{CA}(8;2,9,2)}{\left(\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{array}\right)}$$

FIGURE 13. **Example of squaring the number of columns.**

This construction generates a CA($N(T(t, v) + 1); t, k^2, v$) from the following two ingredients:

- A CA($N; t, k, v$).
- An OA($k^2; 2, T(t, v) + 1, k$).

The procedure is as follows: let X be a CA($N; t, k, v$) and let X^i be the i-th column of X. Let $Y = (y_{ij})$ be an strength-two OA with $T(t, v) + 1$ columns over $\{0, 1, \dots, k - 1\}$. Construct an array of blocks $Z = (z_{ij})$ with $T(t, v) + 1$ rows and k^2 columns; now replace element z_{ij} with column $X^{y_{ij}}$, as shown in Fig 12. The construction of CA(8; 2, 9, 2) using as ingredients CA(4; 2, 3, 2) and OA(9; 2, 2, 3) is illustrated in Fig 13.

To raise to the n-th power the number of columns k of CA($N; t, k, v$) it is required a CA($M; s, l, r$) with the following properties:

- $r = k$
- $M = k^n$
- $s = n$
- $l = (n - 1) \times T(v, t) + 1$

The procedure is similar to squaring. In this case the block array Z has k^n columns and Nl rows. Some other power constructions using *perfect hash families* as the object in which the elements are replaced by columns of another object have been developed in [37], [50], [51].

C. DUPLICATION (2K)

The method developed by Roux [42] allows the duplication of the number of columns of two input CAs.

Theorem 4 (Roux [42]): CAN(3, $2k$, 2) $\leq$ CAN(3, k, 2)+ CAN(2, k, 2).

This method constructs a CA($N_2 + N_3$; 3, $2k$, 2) using X = CA(N_3; 3, k, 2) and Y = CA(N_2; 2, k, 2). The first step is to concatenate horizontally two copies of X, which gives an $N_3 \times 2k$ array. Below the first copy of X is placed a copy of Y, and below the second copy of X is placed the bit-complement of Y (0's are changed to 1's and 1's are changed to 0's). Fig 14 illustrates this construction, and Fig 15 shows the construction of CA(13; 3, 8, 2) using as ingredients X = CA(8; 3, 4, 2) and Y = CA(5; 2, 4, 2).

$$Z = \left(\begin{array}{c|c} X & X \\ \hline Y & \overline{Y} \end{array}\right)$$

X is a strength-3 CA.
Y is a strength-2 CA.
$\overline{Y}$ is the bit-complement of Y.
Z is a strength-3 CA.

FIGURE 14. **Original Roux construction.**

The following Theorem 5 is a generalization for $v \geq 2$ of the Roux construction:

Theorem 5 (Chateauneuf and Kreher [33]): CAN(3, $2k$, v) $\leq$ CAN(3, k, v) +($v - 1$) CAN(2, k, v).

This construction requires two covering arrays X = CA(N_3; 3, k, v) and Y = CA(N_2; 2, k, v) with the same order v and number of columns k. Let π be a cyclic permutation of $\{0, 1, \dots, v - 1\}$. The construction first places two copies of X side by side, and then for $1 \leq i \leq v - 1$ the construction appends vertically N_2 rows formed by concatenating horizontally Y and Y^{π^i}, where Y^{π^i} is the array obtained by applying the permutation π^i to the symbols of Y. Fig 16 shows the structure of the CA resulting from the construction.

As an example of this construction, Fig 17 and Fig 18 show the construction of Z = CA(45; 3, 8, 3) using the ingredients X = CA(27; 3, 4, 3) and Y = CA(9; 2, 4, 3). In this case π = (0, 1, 2), and so π^1 = (0, 1, 2) and π^2 = (0, 1, 2)$\circ$(0, 1, 2) = (0, 2, 1). The array Y^{π^1} is obtained from Y using π^1 = (0, 1, 2), which indicates to change 0's by 1's, 1's by 2's, and 2's by 0's. Similarly, Y^{π^2} is obtained from Y replacing 0's by 2's, 2's by 1's, and 1's by 0's.

D. V-PLICATION (vk)

Colbourn *et al.* [44] proposed a number of recursive Roux-type constructions for strength three and four.

Theorem 6 (Colbourn et al. [44]): For any prime-power $v \geq 3$, CAN(3, vk, v) $\leq$ CAN(3, k, v)+($v-1$)CAN(2, k, v)+ $v^3 - v^2$.

This construction requires four ingredients: (1) a covering array C_3 = CA(N_3; 3, k, v), (2) a covering array C_2 = CA(N_2; 2, k, v), (3) a subarray B of the orthogonal array

$$\underset{}{\mathrm{CA}(8;3,4,2)}\quad \begin{pmatrix} 0&0&0&0\\ 1&1&1&1\\ 0&1&1&0\\ 1&1&0&0\\ 1&0&0&1\\ 0&0&1&1\\ 1&0&1&0\\ 0&1&0&1 \end{pmatrix} \qquad \mathrm{CA}(5;2,4,2)\quad \begin{pmatrix} 1&1&1&0\\ 1&1&0&1\\ 1&0&1&1\\ 0&1&1&1\\ 0&0&0&0 \end{pmatrix}$$

$$\mathrm{CA}(8+5;3,2\cdot 4,2)\quad \left(\begin{array}{cccc|cccc} 0&0&0&0&0&0&0&0\\ 1&1&1&1&1&1&1&1\\ 0&1&1&0&0&1&1&0\\ 1&1&0&0&1&1&0&0\\ 1&0&0&1&1&0&0&1\\ 0&0&1&1&0&0&1&1\\ 1&0&1&0&1&0&1&0\\ 0&1&0&1&0&1&0&1\\ \hline 1&1&1&0&0&0&0&1\\ 1&1&0&1&0&0&1&0\\ 1&0&1&1&0&1&0&0\\ 0&1&1&1&1&0&0&0\\ 0&0&0&0&1&1&1&1 \end{array}\right)$$

FIGURE 15. CA(13; 3, 8, 2) is constructed using the Roux construction with ingredients CA(8; 3, 4, 2) and CA(5; 2, 4, 2).

$$Z = \left(\begin{array}{c|c} X & X\\ \hline Y & Y^{\pi^1}\\ Y & Y^{\pi^2}\\ \vdots & \vdots\\ Y & Y^{\pi^{v-1}} \end{array}\right)$$

X is a $\mathrm{CA}(N_3;3,k,v)$.
Y is a $\mathrm{CA}(N_2;2,k,v)$.
π is a cyclic permutation of $\{0,1,\ldots,v-1\}$.
Z is a $\mathrm{CA}(N_3+(v-1)N_2;3,2k,v)$.

FIGURE 16. Structure of the construction of Theorem 5.

$A = \mathrm{OA}(v^3; 3, v, v)$, and (4) an array $D = \mathrm{DCA}(v-1; 2, v, v)$ constructed by deleting the first row of the multiplication table of $\mathbb{F}_v$.

The first step is to obtain the subarray B of $A = \mathrm{OA}(v^3; 3, v, v)$, where A is generated using the Bush's construction. The subarray B is formed by the rows of A labeled by the polynomials that have degree 2, that is, B is formed by the last $v^3 - v^2$ rows of A. Ingredients C_2, C_3, B, D are used to form three arrays G_1, G_2, G_3 that juxtaposed vertically generate the resulting $\mathrm{CA}(M; 3, vk, v)$, as shown in Fig 19.

Fig 20 and Fig 21 show an example of this Roux-type construction to generate a CA(63; 3, 12, 3). Fig 20 shows the ingredients needed, and Fig 21 illustrates the construction of the arrays G_1, G_2, and G_3. The array G_1 is formed by $v = 3$ copies of $C_3 = \mathrm{CA}(27; 3, 4, 3)$ placed side by side (see Fig 21, row 1). The array G_2 is a matrix with rows indexed

$$X = \mathrm{CA}(27;3,4,3)\quad \begin{pmatrix} 0&0&0&0\\ 1&1&1&0\\ 2&2&2&0\\ 0&1&2&0\\ 1&2&0&0\\ 2&0&1&0\\ 0&2&1&0\\ 1&0&2&0\\ 2&1&0&0\\ 0&1&1&1\\ 1&2&2&1\\ 2&0&0&1\\ 0&2&0&1\\ 1&0&1&1\\ 2&1&2&1\\ 0&0&2&1\\ 1&1&0&1\\ 2&2&1&1\\ 0&2&2&2\\ 1&0&0&2\\ 2&1&1&2\\ 0&0&1&2\\ 1&1&2&2\\ 2&2&0&2\\ 0&1&0&2\\ 1&2&1&2\\ 2&0&2&2 \end{pmatrix} \qquad Y = \mathrm{CA}(9;2,4,3)\quad \begin{pmatrix} 0&0&0&0\\ 1&1&1&0\\ 2&2&2&0\\ 0&1&2&1\\ 1&2&0&1\\ 2&0&1&1\\ 0&2&1&2\\ 1&0&2&2\\ 2&1&0&2 \end{pmatrix}$$

FIGURE 17. An example of Theorem 5. Ingredients $X = \mathrm{CA}(27; 3, 4, 3)$ and $Y = \mathrm{CA}(9; 2, 4, 3)$.

by (r, s) and columns indexed by (f, h) where each cell is the result of adding to cell (r, f) of $C_2 = \mathrm{CA}(9; 2, 4, 3)$ the value of cell (s, h) of $D = \mathrm{DCA}(2; 2, 3, 3)$, and taking modulo v (see Fig 21, row 2). The array G_3 is formed by k copies of each column of the array B (see Fig 21, row 3).

For strength four, one of the constructions is given in Theorem 7:

Theorem 7 (Colbourn et al. [44]): For any prime-power $v \geq 4$, $\mathrm{CAN}(4, vk, v) \leq \mathrm{CAN}(4, k, v) + (v-1)\mathrm{CAN}(3, k, v) + (v^3 - v^2)\mathrm{DCAN}(2, k, v) + \mathrm{CODN}(2, k, v^2) + v^4 - v^2$.

This construction generates a $\mathrm{CA}(L; 4, vk, v)$ by following a procedure similar to the construction of Theorem 6. In this case the following elements are required: (1) a $\mathrm{CA}(N_4; 4, k, v)$; (2) a $\mathrm{CA}(N_3; 3, k, v)$; (3) a $\mathrm{DCA}(S; 2, k, v)$; (4) a $\mathrm{DCA}(v - 1, 2, v, v)$; (5) a $\mathrm{COD}(N_2; 2, k, v^2)$; (6) an $\mathrm{OA}(v^2; 2, v, v)$; (7) an $\mathrm{OA}(v^3; 3, v, v)$; and (8) an $\mathrm{OA}(v^4; 4, v, v)$. These elements are combined to construct five matrices G_1, G_2, G_3, G_4, and G_5, which juxtaposed vertically form a $\mathrm{CA}(L; 4, vk, v)$, where L is the addition of the rows of each matrix G_i.

E. AUGMENTATION

Chateauneuf and Kreher [33] describe a construction called *Construction D* which takes two covering arrays

$$\left(\begin{array}{cccc|cccc}
0&0&0&0&0&0&0&0\\
1&1&1&0&1&1&1&0\\
2&2&2&0&2&2&2&0\\
0&1&2&0&0&1&2&0\\
1&2&0&0&1&2&0&0\\
2&0&1&0&2&0&1&0\\
0&2&1&0&0&2&1&0\\
1&0&2&0&1&0&2&0\\
2&1&0&0&2&1&0&0\\
0&1&1&1&0&1&1&1\\
1&2&2&1&1&2&2&1\\
2&0&0&1&2&0&0&1\\
0&2&0&1&0&2&0&1\\
1&0&1&1&1&0&1&1\\
2&1&2&1&2&1&2&1\\
0&0&2&1&0&0&2&1\\
1&1&0&1&1&1&0&1\\
2&2&1&1&2&2&1&1\\
0&2&2&2&0&2&2&2\\
1&0&0&2&1&0&0&2\\
2&1&1&2&2&1&1&2\\
0&0&1&2&0&0&1&2\\
1&1&2&2&1&1&2&2\\
2&2&0&2&2&2&0&2\\
0&1&0&2&0&1&0&2\\
1&2&1&2&1&2&1&2\\
2&0&2&2&2&0&2&2\\
\hline
0&0&0&0&1&1&1&1\\
1&1&1&0&2&2&2&1\\
2&2&2&0&0&0&0&1\\
0&1&2&1&1&2&0&2\\
1&2&0&1&2&0&1&2\\
2&0&1&1&0&1&2&2\\
0&2&1&2&1&0&2&0\\
1&0&2&2&2&1&0&0\\
2&1&0&2&0&2&1&0\\
\hline
0&0&0&0&2&2&2&2\\
1&1&1&0&0&0&0&2\\
2&2&2&0&1&1&1&2\\
0&1&2&1&2&0&1&0\\
1&2&0&1&0&1&2&0\\
2&0&1&1&1&2&0&0\\
0&2&1&2&2&1&0&1\\
1&0&2&2&0&2&1&1\\
2&1&0&2&1&0&2&1
\end{array}\right)$$

FIGURE 18. An example of Theorem 5. Result $Z = \mathrm{CA}(45; 3, 8, 3)$.

$A = \mathrm{CA}(N; 3, k, v-1)$ and $B = \mathrm{CA}(M; 2, k-1, v-1)$ to construct an array C as shown in Fig 22. The array C is a $\mathrm{CA}(N + kM + k(v-1); 3, k, v)$ over $V = V' \cup \{0\}$, and so $\mathrm{CAN}(3, k, v) \leq \mathrm{CAN}(3, k, v-1) + k \cdot \mathrm{CAN}(2, k-1, v-1) + k(v-1)$.

In the same work [33] it is described another construction to multiply the order to two CAs. Let $A = \mathrm{CA}(N; t, k, v)$ be a CA with entries $a_{ij} \in V = \{0, 1, \ldots, v-1\}$, and

$$G = \left(\begin{array}{c} G_1 \\ \hline G_2 \\ \hline G_3 \end{array}\right)$$

G_1 is an array formed by v copies of $C_3 = \mathrm{CA}(N_3; 3, k, v)$ placed side by side.

G_2 is a matrix of $(v-1)N_2$ rows indexed by ordered pairs from $\{1, \ldots, N_2\} \times \{1, \ldots, v-1\}$, and vk columns indexed by ordered pairs from $\{0, \ldots, k-1\} \times \{0, \ldots, v-1\}$. Denote the covering array C_2 by $C_2 = (c_{i,j})$, and denote the array D by $D = (d_{i,j})$. The element of G_2 at row (r, s) and column (f, h) is given by $c_{r,f} + d_{s,h}$.

G_3 consists of v arrays F_0, F_1, …, F_{v-1} placed side by side, where array F_i is formed by k copies of the i-th column of B.

FIGURE 19. The construction of $\mathrm{CA}(M; 3, vk, v)$ of Theorem 6.

let $B = \mathrm{CA}(M; t, k, w)$ be a CA with entries $b_{ij} \in W = \{0, 1, \ldots, w-1\}$. For $l = 1, 2, \ldots, M$ let C_l be the array of size $N \times k$ with entries (a_{ij}, b_{lj}), where $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, k$. Entries (a_{ij}, b_{lj}) are transformed to an integer c_{ij} between 0 and $vw-1$ by $c_{i,j} = (a_{ij})(w) + b_{lj}$. Then $[C_1\ C_2\ \cdots\ C_M]^T$ is a $\mathrm{CA}(NM; t, k, vw)$ over $V \times W = \{0, 1, \ldots, vw-1\}$, and thus $\mathrm{CAN}(t, k, vw) \leq \mathrm{CAN}(t, k, v) \cdot \mathrm{CAN}(t, k, w)$.

Another form of augmentation is given by Colbourn *et al.* [16]:

$$\mathrm{CAN}(t,k,v) \leq \underbrace{\left\lfloor \frac{v}{v-1} \left\lfloor \frac{v}{v-1} \cdots \left\lfloor \frac{v\,\mathrm{CAN}(t,k,v-1)}{v-1} \right\rfloor \cdots \right\rfloor \right\rfloor}_{k \text{ times}}$$

Given $A = \mathrm{CA}(N; t, k, v-1)$ the objective is to construct $B = \mathrm{CA}(M; t, k, v)$. The process consists in adding the new symbol to each column of A in a greedy manner. Let $\sigma' = v-1$ be the symbol to add to every column of A, and let $C_0 = A$. Index the columns of A starting from 1. For $j = 1, 2, \ldots, k$ construct C_j from C_{j-1} as follows: initially C_j is equal to C_{j-1}, then select in column j the symbol $\sigma \in \{0, 1, \ldots, v-2\}$ that appears the least number of times; next, for every row of C_{j-1} that contains symbol σ in column j add a new row to C_j that is identical to the row of C_{j-1} except that column j contains σ' instead of σ.

Finally, Colbourn [52] introduced some strategies to increase in one unit the order of CAs of strength two. The idea of one of these strategies is to construct a CA of order v by juxtaposing $v-1$ arrays on two symbols to a CA of order $v-1$.

F. AUGMENTED ANNEALING

Augmented annealing introduced by Cohen *et al.* [43] combines combinatorial constructions with simulated annealing. The problem is divided into smaller subproblems or ingredients using a combinatorial construction; next, the ingredients

(a)

$$\begin{pmatrix} 0&0&0&0\\1&1&1&0\\2&2&2&0\\0&1&2&0\\1&2&0&0\\2&0&1&0\\0&2&1&0\\1&0&2&0\\2&1&0&0\\0&1&1&1\\1&2&2&1\\2&0&0&1\\0&2&0&1\\1&0&1&1\\2&1&2&1\\0&0&2&1\\1&1&0&1\\2&2&1&1\\0&2&2&2\\1&0&0&2\\2&1&1&2\\0&0&1&2\\1&1&2&2\\2&2&0&2\\0&1&0&2\\1&2&1&2\\2&0&2&2 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 0&0&0\\1&1&1\\2&2&2\\0&1&2\\1&2&0\\2&0&1\\0&2&1\\1&0&2\\2&1&0\\0&1&1\\1&2&2\\2&0&0\\0&2&0\\1&0&1\\2&1&2\\0&0&2\\1&1&0\\2&2&1\\0&2&2\\1&0&0\\2&1&1\\0&0&1\\1&1&2\\2&2&0\\0&1&0\\1&2&1\\2&0&2 \end{pmatrix}$$

(c)

$$\begin{pmatrix} 0&0&0&0\\1&1&1&0\\2&2&2&0\\0&1&2&1\\1&2&0&1\\2&0&1&1\\0&2&1&2\\1&0&2&2\\2&1&0&2 \end{pmatrix}$$

(d)

$$\begin{pmatrix} 0&0&0\\0&1&2\\0&2&1 \end{pmatrix}$$

FIGURE 20. Ingredients to construct CA(63; 3, 12, 3) using Theorem 6. (a) C_3 = CA(27; 3, 4, 3). (b) C_2 = CA(9; 2, 4, 3). (c) A = OA(27; 3, 3, 3), and the shaded area corresponds to the submatrix B. (d) $\mathcal{D}$ is the multiplication table of $\mathbb{F}_3$, and the shaded area corresponds to D = DCA(2; 2, 3, 3).

are generated either by a combinatorial construction or by simulated annealing.

One of the constructions introduced for CAs of strength three is based on ordered designs (OD). An *ordered design* OD(t, k, v) is an array of size $\binom{v}{t}t! \times k$ over a set of v symbols such that every subarray of t columns covers each t-tuple of distinct symbols. For q prime-power an OD$(3, q+1, q+1)$ exists [13]. From this OD a CA with strength $t = 3, k = q+1$ columns, and order $v = q+1$ can be constructed by appending to the OD $\binom{q+1}{2}$ copies of a CA$(N; 3, q+1, 2)$.

For example, consider $q = 3$, so $k = q+1 = 4$ and $v = q+1 = 4$. Fig 23(a) shows an OD(3, 4, 4); this OD has $\binom{v}{t}t! = 4{\cdot}6 = 24$ rows, and it covers all triples (x, y, z), where $x \neq y \neq z \neq x$ and $x, y, z \in \{0, 1, 2, 3\}$, in every subset of 3 columns. The CA of order 2 needed to complete the triples missing in the OD is shown in Fig 23(b). The number of copies required of this CA is $\binom{4}{2} = 6$, and these copies should be relabeled using the symbols sets $\{0, 1\}$, $\{0, 2\}$, $\{0, 3\}$, $\{1, 2\}$, $\{1, 3\}$, and $\{2, 3\}$ respectively. The final result is a CA$(24 + 6(8); 3, 4, 4)$.

IV. EXACT METHODS

Exact methods construct CAs by exhaustive search. Therefore, they can only be used to construct small CAs. In this category we found Branch & Bound strategies, such as the work proposed by Bracho-Rios *et al.* [53]; and SAT encodings, like the ones developed by Banbara *et al.* [54].

Because more than one method exploit the symmetries in CAs it is convenient to introduce here the three symmetries of CAs. Given a covering array A = CA$(N; t, k, v)$ we can permute the rows and the columns of A, and the resulting array is a CA *isomorphic* to A, that is, the new array has the same coverage properties that A. In addition, in any column of the matrix A the v symbols can be permuted, and the resulting array is also isomorphic to A. Therefore, the CA A can have up to $N!k!(v!)^k$ isomorphic CAs. Symbol permutations are also called *column relabelings* or *relabelings*. Exact methods try to break some of these three symmetries to accelerate the search.

A. THE AUTOMATIC GENERATOR EXACT

Yan and Zhang [55] introduced an exhaustive search technique to construct CAs. The algorithm assigns values to each cell of an $N \times k$ matrix until all covering conditions are satisfied. After assigning a value, a constraint propagation function is executed to determine if this assignment implies a value for another cell or a contradiction. The search is accelerated by symmetry breaking techniques and by two heuristics called respectively LNH and SCEH. The heuristic LNH uses a variable *mdn* to store the largest value present in the assigned cells; when a new cell will be assigned the candidate values are $\{0, 1, \ldots, mdn + 1\}$, that is, values greater than $mdn + 1$ are not considered. The heuristic SCEH assumes that it is always possible to find a CA where each sub-combination (or sub-tuple) of size s occurs almost the same number of times in a subset of s columns. The authors integrated all these techniques in a program called EXACT (EXhaustive seArch of Combinatorial Test suites). The EXACT tool was further improved in [56].

B. NEW BACKTRACKING ALGORITHM

Bracho-Rios *et al.* [53] introduced a searching algorithm to construct binary CAs ($v = 2$) of strength t and dimensions $N \times k$. The algorithm constructs the CAs column by column imposing a lexicographic ordering of the columns to break the column and row symmetries. The columns to construct the CAs are balanced in symbols, so the candidate columns have $\lfloor \frac{N}{2} \rfloor$ zeros and $N - \lfloor \frac{N}{2} \rfloor$ ones. Before starting the search, a block of t columns is fixed, the first $N - 2^t$ rows of the

$$C_3 = \left(\begin{array}{cccc}
0&0&0&0\\
1&1&1&0\\
2&2&2&0\\
0&1&2&0\\
1&2&0&0\\
2&0&1&0\\
0&2&1&0\\
1&0&2&0\\
2&1&0&0\\
0&1&1&1\\
1&2&2&1\\
2&0&0&1\\
0&2&0&1\\
1&0&1&1\\
2&1&2&1\\
0&0&2&1\\
1&1&0&1\\
2&2&1&1\\
0&2&2&2\\
1&0&0&2\\
2&1&1&2\\
0&0&1&2\\
1&1&2&2\\
2&2&0&2\\
0&1&0&2\\
1&2&1&2\\
2&0&2&2
\end{array}\right)
\qquad
G_1 = \left(\begin{array}{cccc|cccc|cccc}
0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&1&1&1&0&1&1&1&0\\
2&2&2&0&2&2&2&0&2&2&2&0\\
0&1&2&0&0&1&2&0&0&1&2&0\\
1&2&0&0&1&2&0&0&1&2&0&0\\
2&0&1&0&2&0&1&0&2&0&1&0\\
0&2&1&0&0&2&1&0&0&2&1&0\\
1&0&2&0&1&0&2&0&1&0&2&0\\
2&1&0&0&2&1&0&0&2&1&0&0\\
0&1&1&1&0&1&1&1&0&1&1&1\\
1&2&2&1&1&2&2&1&1&2&2&1\\
2&0&0&1&2&0&0&1&2&0&0&1\\
0&2&0&1&0&2&0&1&0&2&0&1\\
1&0&1&1&1&0&1&1&1&0&1&1\\
2&1&2&1&2&1&2&1&2&1&2&1\\
0&0&2&1&0&0&2&1&0&0&2&1\\
1&1&0&1&1&1&0&1&1&1&0&1\\
2&2&1&1&2&2&1&1&2&2&1&1\\
0&2&2&2&0&2&2&2&0&2&2&2\\
1&0&0&2&1&0&0&2&1&0&0&2\\
2&1&1&2&2&1&1&2&2&1&1&2\\
0&0&1&2&0&0&1&2&0&0&1&2\\
1&1&2&2&1&1&2&2&1&1&2&2\\
2&2&0&2&2&2&0&2&2&2&0&2\\
0&1&0&2&0&1&0&2&0&1&0&2\\
1&2&1&2&1&2&1&2&1&2&1&2\\
2&0&2&2&2&0&2&2&2&0&2&2
\end{array}\right)$$

$$C_2 + D = \left(\begin{array}{cccc|cccc|cccc}
0+0&0+0&0+0&0+0&0+1&0+1&0+1&0+1&0+2&0+2&0+2&0+2\\
1+0&1+0&1+0&0+0&1+1&1+1&1+1&0+1&1+2&1+2&1+2&0+2\\
2+0&2+0&2+0&0+0&2+1&2+1&2+1&0+1&2+2&2+2&2+2&0+2\\
0+0&1+0&2+0&1+0&0+1&1+1&2+1&1+1&0+2&1+2&2+2&1+2\\
1+0&2+0&0+0&1+0&1+1&2+1&0+1&1+1&1+2&2+2&0+2&1+2\\
2+0&0+0&1+0&1+0&2+1&0+1&1+1&1+1&2+2&0+2&1+2&1+2\\
0+0&2+0&1+0&2+0&0+1&2+1&1+1&2+1&0+2&2+2&1+2&2+2\\
1+0&0+0&2+0&2+0&1+1&0+1&2+1&2+1&1+2&0+2&2+2&2+2\\
2+0&1+0&0+0&2+0&2+1&1+1&0+1&2+1&2+2&1+2&0+2&2+2\\
\hline
0+0&0+0&0+0&0+0&0+2&0+2&0+2&0+2&0+1&0+1&0+1&0+1\\
1+0&1+0&1+0&0+0&1+2&1+2&1+2&0+2&1+1&1+1&1+1&0+1\\
2+0&2+0&2+0&0+0&2+2&2+2&2+2&0+2&2+1&2+1&2+1&0+1\\
0+0&1+0&2+0&1+0&0+2&1+2&2+2&1+2&0+1&1+1&2+1&1+1\\
1+0&2+0&0+0&1+0&1+2&2+2&0+2&1+2&1+1&2+1&0+1&1+1\\
2+0&0+0&1+0&1+0&2+2&0+2&1+2&1+2&2+1&0+1&1+1&1+1\\
0+0&2+0&1+0&2+0&0+2&2+2&1+2&2+2&0+1&2+1&1+1&2+1\\
1+0&0+0&2+0&2+0&1+2&0+2&2+2&2+2&1+1&0+1&2+1&2+1\\
2+0&1+0&0+0&2+0&2+2&1+2&0+2&2+2&2+1&1+1&0+1&2+1
\end{array}\right)
\qquad
G_2 = \left(\begin{array}{cccc|cccc|cccc}
0&0&0&0&1&1&1&1&2&2&2&2\\
1&1&1&0&2&2&2&1&0&0&0&2\\
2&2&2&0&0&0&0&1&1&1&1&2\\
0&1&2&1&1&2&0&2&2&0&1&0\\
1&2&0&1&2&0&1&2&0&1&2&0\\
2&0&1&1&0&1&2&2&1&2&0&0\\
0&2&1&2&1&0&2&0&2&1&0&1\\
1&0&2&2&2&1&0&0&0&2&1&1\\
2&1&0&2&0&2&1&0&1&0&2&1\\
\hline
0&0&0&0&2&2&2&2&1&1&1&1\\
1&1&1&0&0&0&0&2&2&2&2&1\\
2&2&2&0&1&1&1&2&0&0&0&1\\
0&1&2&1&2&0&1&0&1&2&0&2\\
1&2&0&1&0&1&2&0&2&0&1&2\\
2&0&1&1&1&2&0&0&0&1&2&2\\
0&2&1&2&2&1&0&1&1&0&2&0\\
1&0&2&2&0&2&1&1&2&1&0&0\\
2&1&0&2&1&0&2&1&0&2&1&0
\end{array}\right)$$

$$B = \left(\begin{array}{ccc}
0&1&1\\
1&2&2\\
2&0&0\\
0&2&0\\
1&0&1\\
2&1&2\\
0&0&2\\
1&1&0\\
2&2&1\\
0&2&2\\
1&0&0\\
2&1&1\\
0&0&1\\
1&1&2\\
2&2&0\\
0&1&0\\
1&2&1\\
2&0&2
\end{array}\right)
\qquad
G_3 = \left(\begin{array}{cccc|cccc|cccc}
0&0&0&0&1&1&1&1&1&1&1&1\\
1&1&1&1&2&2&2&2&2&2&2&2\\
2&2&2&2&0&0&0&0&0&0&0&0\\
0&0&0&0&2&2&2&2&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1\\
2&2&2&2&1&1&1&1&2&2&2&2\\
0&0&0&0&0&0&0&0&2&2&2&2\\
1&1&1&1&1&1&1&1&0&0&0&0\\
2&2&2&2&2&2&2&2&1&1&1&1\\
0&0&0&0&2&2&2&2&2&2&2&2\\
1&1&1&1&0&0&0&0&0&0&0&0\\
2&2&2&2&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1\\
1&1&1&1&1&1&1&1&2&2&2&2\\
2&2&2&2&2&2&2&2&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0\\
1&1&1&1&2&2&2&2&1&1&1&1\\
2&2&2&2&0&0&0&0&2&2&2&2
\end{array}\right)$$

FIGURE 21. **Example of the construction of CA(63; 3, 12, 3) of Theorem 6. Rows 1, 2, 3 illustrate respectively the construction of the matrices G_1, G_2, and G_3. Column 2 corresponds to the resulting covering array G = CA(63; 3, 12, 3).**

block are filled with zeros and the last 2^t rows are filled with the 2^t tuples of size t over the symbol set {0, 1}. Suppose we have a partial solution with r columns ($t \leq r < k$), and let l be the last column of the partial solution. To construct a CA of strength t and $r + 1$ columns, the algorithm checks all columns l' greater than l in lexicographic order until finding

$$C = \left(\begin{array}{cccccc}
\multicolumn{6}{c}{A = \mathrm{CA}(N;3,k,v-1)} \\
\hline
\vec{0} & B_1 & B_2 & \cdots & B_{k-2} & B_{k-1} \\
B_1 & \vec{0} & B_2 & \cdots & B_{k-2} & B_{k-1} \\
B_1 & B_2 & \vec{0} & \cdots & B_{k-2} & B_{k-1} \\
 & & & \ddots & & \\
B_1 & B_2 & B_3 & \cdots & B_{k-1} & \vec{0} \\
\hline
\vec{V'} & \vec{0} & \vec{0} & \cdots & \vec{0} & \vec{0} \\
\vec{0} & \vec{V'} & \vec{0} & \cdots & \vec{0} & \vec{0} \\
\vec{0} & \vec{0} & \vec{V'} & \cdots & \vec{0} & \vec{0} \\
 & & & \ddots & & \\
\vec{0} & \vec{0} & \vec{0} & \cdots & \vec{0} & \vec{V'}
\end{array}\right)$$

In the first part the symbol set is $V' = \{1, 2, \ldots, v-1\}$.

In the second part B_j is column j of $B = \mathrm{CA}(M; 2, k-1, v-1)$ over the symbol set V', and $\vec{0} = [0, 0, \ldots, 0]$ is a column vector of M zeros.

In the third part $\vec{V'} = [1, 2, \ldots, v-1]$.

FIGURE 22. **Construction D.**

(a)

$$\begin{pmatrix}
0 & 1 & 2 & 3 \\
1 & 0 & 3 & 2 \\
2 & 3 & 0 & 1 \\
3 & 2 & 1 & 0 \\
0 & 2 & 3 & 1 \\
1 & 3 & 2 & 0 \\
2 & 0 & 1 & 3 \\
3 & 1 & 0 & 2 \\
0 & 3 & 1 & 2 \\
1 & 2 & 0 & 3 \\
2 & 1 & 3 & 0 \\
3 & 0 & 2 & 1 \\
0 & 1 & 3 & 2 \\
1 & 0 & 2 & 3 \\
2 & 3 & 1 & 0 \\
3 & 2 & 0 & 1 \\
0 & 2 & 1 & 3 \\
1 & 3 & 0 & 2 \\
2 & 0 & 3 & 1 \\
3 & 1 & 2 & 0 \\
0 & 3 & 2 & 1 \\
1 & 2 & 3 & 0 \\
2 & 1 & 0 & 3 \\
3 & 0 & 1 & 2
\end{pmatrix}$$

(b)

$$\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1
\end{pmatrix}$$

FIGURE 23. **Ingredients to construct a CA(24 + 6(8); 3, 4, 4).**

one which makes a CA of strength t with the r columns of the partial solution, and such that the rows and columns of the new partial solution are sorted lexicographically. If no such column is found the algorithm backtracks to column $r-1$.

C. GENERATION OF NON-ISOMORPHIC CAs

Torres-Jimenez and Izquierdo-Marquez [57] proposed an algorithm to generate non-isomorphic CAs. For given values N, t, k, v the set of all covering arrays CA(N; t, k, v) is partitioned into classes of isomorphic CAs. The algorithm constructs one CA from each isomorphism class, and so its output is a set of non-isomorphic CAs. The CAs are constructed column by column verifying that the current partial CA is the *minimum* of its class. The minimum of an isomorphism class is defined as the CA with the minimum lexicographic order when its elements are arranged in column-major order. If for some parameters N, t, k, v none CA with k columns was constructed, then CA(N; t, k, v) does not exists. The optimality of a CA(N; t, k, v) is proven by showing that the set of non-isomorphic CA($N-1$; t, k, v) is empty. This algorithm takes into account the three symmetries of CAs to accelerate the search.

D. CONSTRAINT PROGRAMMING

Hnich *et al.* [58] developed constraint programming models for the construction of CAs. The first model, called *naive matrix model*, uses variables x_{ri} such that $x_{ri} = m$ if the entry (r, i) of the matrix is equal to m. Consider $t = 3$; to express that the tuple at row r in columns (i, j, l) is (m, n, p) the constraint is $x_{rijlmnp} = (x_{ri} = m \ \& \ x_{rj} = n \ \& \ x_{rl} = p)$. The constraint that each t-tuple must occur at least once in every subset of t columns is expressed by $\sum_r x_{rijlmnp} \geq 1$.

In the *alternative matrix model* a tuple of t variables of the first model is represented by a *compound* variable. With $t = 3$, for example, the compound variable $y_{r(i,j,l)}$ represents the tuple of variables (x_{ri}, x_{rj}, x_{rl}). The domain of a compound variable is $\{0, 1, \ldots, v^t - 1\}$; and the coverage constraints require that in every subset of t columns there must be at least one compound variable for each number from 0 to $v^t - 1$.

The *integrated model* combines the variables of the two previous models. Assigning a value to a compound variable assigns a value to each variable of the naive model, and in the same way assigning a value to a variable of the naive model reduces the domain of the compound variable.

E. SAT ENCODINGS

Lopez-Escogido *et al.* [59] introduced a SAT encoding for strength-two CAs. The model uses v variables for each entry of the matrix $M = (m_{ij})$ that will contain the CA; if M has size $N \times k$ the total number of variables is Nkv. Element m_{ij} gets the value $0 \leq x < v$ if and only if the variable $m_{i,j,x}$ is true. The clauses of the model guarantee that (1) each element of M takes at least one value from the set $\{0, 1, \ldots, v-1\}$, (2) each element of M takes only one value, and (3) the matrix M satisfy the coverage properties to be a CA.

The work of Ansótegui *et al.* [60] proposes a SAT encoding for the construction of optimal CAs, but in this work the encoding is solved as a MAXSAT instance. Banbara *et al.* [54] also developed two SAT encodings to construct CA; these encodings are called *order encoding* and *mixed encoding* respectively.

V. GREEDY METHODS

Greedy methods have the characteristic of generating good solutions in short time. Commonly, greedy algorithms are use to generate both CAs and MCAs. Most of the public available tools to generate test suites use greedy algorithms (AETG [61], TCG [62], ACTS [63], and IPOG-F [64]). Other important greedy algorithms are the Building-Block Algorithm (BBA) of Ronneseth and Colbourn [65], the Intersection Residual Pair Set Strategy (IRPS) of Younis *et al.* [66], the Deterministic Density Algorithm (DDA) of Bryce and Colbourn [67], and the method of Calvagna and Gargantini [68] based on coverage inheritance. An interesting experimentation to identify empirically the best greedy algorithm for strength-two CAs was done by Nie *et al.* [69].

A. TEST CASE GENERATOR (TCG)

Tung and Aldiwan [62] proposed a tool called Test Case Generator (TCG). This tool constructs strength-two MCAs one row at a time. The k parameters of the MCA are permuted in such a way they are arranged in non-increasing order of the cardinality of their alphabets. Let M be the cardinality of the larger alphabet. TCG generates up to M candidate rows, where each candidate row is constructed element by element following a special procedure based on counting the new pairs that are covered when an element is added to the current partial row. From the M candidate rows, the one that covers the greatest number of new pairs is taken as the winner row. Rows are added in this way until the MCA is complete.

B. DETERMINISTIC DENSITY ALGORITHM (DDA)

Bryce and Colbourn [67] developed an algorithm to construct strength-two MCAs called Deterministic Density Algorithm (DDA). Let v_i denote the number of *levels* (symbols) that the i-th *factor* (parameter) can take. The local density for factors i and j is $\delta_{i,j} = r_{i,j}/v_i v_j$, with $r_{i,j}$ being the number of missing pairs between factors i and j. The global density is the sum of the local densities of each pair of factors: $\delta = \sum_{1 \leq i < j \leq k} \delta_{i,j}$. The objective of DDA is to generate rows covering at least δ uncovered pairs. To construct a new row, a factor is fixed and the densities are updated; the process is repeated until all factors are fixed. Suppose that f_s is a free factor; the level for f_s is selected as follows: let $\rho_{i,s,\sigma}$ be $1/v_i$ times the number of missing pairs involving a level of f_i and level σ of f_s; then, select the value σ that maximizes $\sum_{\substack{1 \leq i \leq k \\ i \neq s}} \rho_{i,s,\sigma}$.

The method also requires a way to assign a density to each factor and to each level. Different rules for computing the factor density are applied depending on the number of remaining levels for the factor; the same is true for level density. Algorithm 1 shows the DDA pseudocode taken from Bryce and Colbourn [67]. Factor or level tie-breaking rules are applied to select respectively a factor or level among the ones with equal maximum densities. The work of Bryce and Colbourn [70] extends the density method to higher strengths.

Algorithm 1 DDA, Deterministic Density Algorithm (Bryce and Colbourn [67]).

```
begin
    start with an empty CA
    while there are uncovered pairs do
        compute factor density for each factor
        initialize a new row with all factors not fixed
        while there is a factor whose level is not fixed in the new row do
            select such a factor $f$ with largest density, using a factor tie-breaking rule
            compute level density for each level of factor $f$
            select a level $\ell$ for $f$ with maximum density using a level tie-breaking rule
            fix $f$ to level $\ell$
            recompute densities for each factor
        end while
        add the constructed row to the CA
    end while
end
```

C. IN-PARAMETER-ORDER (IPO)

Lei and Tai [71] introduced a new method for pairwise testing called In-Parameter-Order (IPO). This method begins with the generation of a test set with two parameters, and iteratively adds new parameters. The extension process has two stages: horizontal growth and vertical growth. Suppose the new parameter has q symbols. For $1 \leq j \leq q$ the row j of the new parameter gets the value j; and the remaining elements of the new parameter get the value that covers the greatest number of uncovered pairs. The pairs not covered in the horizontal growth are stored in a list π, and they will be covered later in the vertical growth stage.

Rows added by the vertical growth stage may contain unassigned positions denoted by "$-$". Suppose that the current number of parameters is k. To cover a missing pair (w, u) between a parameter r ($r < k$) and parameter k, the algorithm checks if there is a row that has "$-$" in column r and u in column k; in this case "$-$" is replaced by w. If no such row exists the algorithm adds a new row where all positions are "$-$" except for the corresponding to columns r and k that contain the pair (w, u). Algorithm 2 shows the pseudocode of the IPO strategy as described in Lei and Tai [71]. Lei *et al.* [72] generalized the IPO strategy to multi-way testing. The new algorithm was called In-Parameter-Order-General (IPOG).

D. INTERSECTION RESIDUAL PAIR SET STRATEGY (IRPS)

Younis *et al.* [66] introduced a novel strategy for generating strength-two CAs called Intersection Residual Pair Set Strategy (IRPS). The IRPS strategy performs the following steps:

1) Generate all pairs to be covered by any two factors and store the pairs into a compact linked list called *Pi*. For a test set with k parameters, the *Pi* list contains $(k - 1)$

Algorithm 2 IPO, In-Parameter-Order (Lei and Tai [71]).

```
1  begin
2    T ← initial test set for the parameters $f_1$ and $f_2$
3    for $i = 3, \ldots, n$ do
         /* horizontal growth                 */
4        assume that the values for factor $f_i$ are $v_1, \ldots, v_q$
5        $\pi$ ← { pairs between values of $f_i$ and values of
         $f_1, \ldots, f_{i-1}$ }
6        if $|T| \leq q$ then
7            for $1 \leq j \leq |T|$ assign $v_j$ to row $j$ and remove
             from $\pi$ the covered pairs
8        end if
9        else
10           for $1 \leq j \leq q$ assign $v_j$ to row $j$ and remove
             from $\pi$ the covered pairs
11           for $q \leq j \leq |T|$ assign to row $j$ a value of $f_i$
             such that the resulting test covers the
             greatest number of uncovered pairs, and
             remove from $\pi$ the covered pairs
12       end if
         /* vertical growth                   */
13       $T' \leftarrow \emptyset$
14       foreach pair in $\pi$ do
15           assume that the pair contains value $w$ of $f_r$,
             $1 \leq r < i$, and value $u$ of $f_i$
16           if $T'$ contains a row with "–" as the value
             of $f_r$ and $u$ as the value of $f_i$ then
17               modify this row by replacing "–" by $w$
18           end if
19           else
20               add a new test to $T'$ that has $w$ as the
                 value of $f_r$, $u$ as the value of $f_i$, and "–"
                 as the value of every other parameter
21           end if
22       end foreach
23       $T \leftarrow T \cup T'$
24   end for
25 end
```

linked lists, so *Pi* is a linked list of linked lists. Every pair of values to be covered by any two factors appears exactly once in *Pi*.

2) Search the *Pi* list to construct the row or test case that covers the larger number of uncovered pairs.
3) Repeat step 2 until *Pi* is empty.

Consider a system with $k = 4$ factors X, Y, Z, and W, each one with three possible values $\{x_0, x_1, x_2\}$, $\{y_0, y_1, y_2\}$, $\{z_0, z_1, z_2\}$, and $\{w_0, w_1, w_2\}$ respectively. The number of distinct pairs of values is $\binom{4}{2}3^2 = 54$. The weight of a candidate row is defined as the number of pairs that are uncovered in the current test suite but covered in the candidate row. For example, the row $x_0y_0z_0w_0$ covers the pairs $\{x_0, y_0\}$, $\{x_0, z_0\}$, $\{x_0, w_0\}$, $\{y_0, z_0\}$, $\{y_0, w_0\}$, and $\{z_0, w_0\}$; so its weight is 6 if none of these pairs has been covered by another row. The first added row has maximum weight because at the beginning all pairs are uncovered. From the second row onward, the algorithm tries to construct a test case with maximum weight, but if no such test case is found then the algorithm searches for a test case with weight equal to $wmax - 1$. Thus, the desired weight is reduced in one unit when no row for the current desired weight is found. For every new generated row, the variables involved in it are deleted from the *Pi* list. The algorithm iterates in this manner until the *Pi* list is empty.

E. BUILDING-BLOCK ALGORITHM (BBA)

Ronneseth and Colbourn [65] introduced a new algorithm to construct CAs, the Building-Block Algorithm (BBA). The fundamental idea of this algorithm is to combine smaller CAs to form a partial CA with a number of columns equal to the sum of the number of columns of the smaller CAs; the partial CA has a number of uncovered pairs which are covered by adding rows to the partial CA. The rows of the smaller CAs are reordered to reduce the number of rows needed to complete the partial CA. Pairs involving two factors in the same smaller CA are covered in the partial CA, but pairs involving factors of distinct smaller CAs may be uncovered in the partial CA; additional rows are added to the partial CA to cover these *cross pairs*. The most important decision is how to reorder and combine the rows of the building blocks in order to minimize the number of uncovered pairs. After combining rows, some new rows are added to cover the uncovered cross pairs; this can be done using greedy algorithms like AETG, DDA, and TCG, or using heuristic algorithms such as simulated annealing, tabu search, and hill climbing.

F. TWO STAGE ALGORITHMS

The method of Torres-Jimenez *et al.* [73] integrates a greedy technique and a simulated annealing algorithm to increase the number of columns of an initial CA. The first stage consists in adding a new column to the input CA, trying to cover the greatest number of tuples between the new column and $(t-1)$ of the previous columns. At the beginning the new column has all its cells unassigned; at each iteration one of the free cells is filled with the value that gives the least number of uncovered tuples. The second stage is a simulated annealing algorithm whose objective is to cover the tuples not covered in the first stage. If the new column was added successfully, then the two stages are repeated to add another column. Occasionally, a row is added to help the simulated annealing algorithm to cover the missing tuples.

Sarkar and Colbourn [74] developed a two-stage framework for the construction of CAs. In the first stage a randomized algorithm creates the initial array; this initial array should cover all but at most ρ tuples. The uncovered tuples are saved in a list L, and in the second stage they are covered using a deterministic strategy. For the first state there are two options: a basic randomized algorithm, and a Moser-Tardos type algorithm. For the second stage there are four options: adding one row per uncovered tuple, a greedy coloring strategy, the density algorithm, and a graph coloring algorithm.

G. RANDOM EXTENSION OF CPHFs

The method of Colbourn *et al.* [75] improved a great number of upper bounds of CAN by constructing CPHFs using column resampling and random extension algorithms. The column resampling method is a Moser-Tardos type algorithm in which the initial CPHF is created by selecting all its elements uniformly at random from the set of possible values (the set of all permutation vectors); then, the columns that are part of uncovered t-subsets of columns are resampled independently and uniformly at random until all t-subsets of columns are covered. In the random extension algorithm the input is a CPHF with k_0 columns and the objective is to construct a CPHF with $k \geq k_0$ columns. The new columns are added by performing several iterations of a loop that generates a random column and verifies if that column makes a CPHF with the previous columns. When a candidate column does not form a CPHF with the current columns, that candidate column can replace one of the current columns of the CPHF if one column of the CPHF is involved in all uncovered combinations generated by adding the candidate column to the current CPHF. The random extension algorithm is a greedy algorithm, and it performs better than the column resampling algorithm. Using these algorithms Colbourn and Lanus [76] constructed CPHFs with restricted entries; these CPHFs generate smaller CAs than the ones given by CPHFs without restricted entries.

VI. METAHEURISTIC METHODS

Some metaheuristics that have been used to construct CAs are simulated annealing [77]–[80], tabu search [44], [81]–[83], memetic algorithms [84], genetic algorithms [81], [85], [86], ant colony [87], particle swarm optimization [88]–[90], harmony search [91], and cuckoo search [92]. Some of these approaches have been implemented following a parallel programming paradigm, like the parallel simulated annealing approach to construct CAs implemented in [93]. Also, hyperheuristic search has been applied to construct CAs [94], [95]. Metaheuristic methods are currently the best methods to construct MCAs and other variations of CAs such as variable strength CAs and constrained CAs, but for uniform CAs only tabu search and simulated annealing have succeeded in constructing some of the best-known uniform CAs.

A. SIMULATED ANNEALING

Torres-Jimenez and Rodriguez-Tello [79] developed a simulated annealing (SA) algorithm to construct binary CAs. The main characteristics of this algorithm are the creation of the initial solution, and the neighborhood function. The initial solution is generated randomly, but every column has $\lfloor N/2 \rfloor$ zeros and $N - \lfloor N/2 \rfloor$ ones. The neighborhood function is formed by two functions N_1 and N_2, which are based on two procedures called respectively $switch(A, i, j)$ and $swap(A, i, j, l)$. An application of *switch* changes the (i, j) entry of the current solution A from 0 to 1, or from 1 to 0; and *swap* exchanges the contents of rows i and l in column j of the current solution. The function N_1 makes ω successive calls to *switch* with distinct values i and j, and the final result of the function is the switch that minimizes the number of missing tuples. Function N_2 behaves similarly to N_1 but uses the *swap* procedure. The algorithm finalizes when a CA without missing tuples is constructed, when the final temperature is reached, or when the best global solution is not improved in ϕ consecutive temperature decrements.

Avila-George *et al.* [93] presented three parallel SA algorithms. The first one is denominated *independent search*; here every process works independently, and the final solution is selected from the best solutions found by each process. In the *semi-independent search* the Markov chains are divided among the available processes. After generating their corresponding subchains, the processes share their intermediate solutions to obtain the current best global solution; from this global solution the processes restart the search. The *cooperative search* is similar to the semi-independent search, the difference is that a process can update the global best solution without waiting for the other processes to finish their subchains. Thus, the global best solution is updated asynchronously, and also the processes start their subchains asynchronously taking as initial solution the current best global solution.

SA has also been used to construct CPHFs. In [96] it was developed a SA algorithm to construct CPHFs of order $v = 3$, and in [97] SA was used to construct CPHFs of order $v = 4$.

B. TABU SEARCH

Tabu search (TS) was used by Nurmela [82] to construct CAs. In this method the initial solution is generated randomly. The cost of a solution is defined as the number of uncovered t-tuples in the solution. Uncovered tuples are covered by selecting one of them randomly and searching if there are some rows that can cover the tuple by updating only one element of the row; these kind of changes are called *moves*. If several moves can cover the tuple, then the move that produces the smaller cost is selected; in case of a tie, the winner move is selected randomly among the best moves. A move is *tabu* if it changes an element updated in the last T moves. If a missing tuple can not be covered by a move, then the tuple is covered in any of the rows. The algorithm performs a move after another until the current solution becomes a CA. When a CA is constructed a row is deleted and the tabu search is restarted to try to construct another CA with one less row.

Zekaoui [83] developed two TS algorithms called respectively Point Tabu Search (POT) and Pair Tabu Search (PAT) to construct CAs and MCAs of strength two. In POT the neighborhood of the current solution consists in all arrays having one entry distinct of the current solution. From this neighborhood a random sample is taken and the array with the lowest cost (the smallest number of uncovered tuples) is selected; in case of a tie, one of the best arrays is chosen randomly. PAT is very similar to the algorithm of Nurmela [82]; here an uncovered pair is selected randomly and the neighborhood is formed by all arrays that cover the selected pair by changing one or two values, although priority is given to arrays that only require one change. The cost of each array in the neighborhood is computed and the array with the lowest cost is selected as the new current solution.

Walker II and Colbourn [98] used TS to construct CAs indirectly through the construction of CPHFs.

C. GENETIC ALGORITHMS

Stardom [81] developed a genetic algorithm (GA) to construct CAs. The population is a set of matrices with uncovered tuples. There are two ways to mate the genes (the elements of the population) *tournament selection* and *quick convergence*. In *tournament selection* the population is partitioned into groups of four elements; in each group the two fittest genes are selected as parents and recombined to produce two offspring, which replace the other two genes not selected as parents. In *quick convergence* the population of size S is partitioned into two groups of size $S/2$. The i-th members of each group are mated, and then the S offspring are mutated by changing one of its entries by a random value. After that, the median fitness value of the population is computed and the T arrays with a fitness smaller than the median value pass to the next generation, together with other $S - T$ arrays selected randomly from those arrays with a fitness greater than or equal to the median value. Parents can be mated in three ways called respectively *row crossover*, *column crossover*, and *point crossover*. Suppose the size of the arrays is $N \times k$, and define two values $1 < i < N$ and $1 < j < k$. In row crossover the parents are divided into two groups of rows determined by i; similarly, in column crossover the parents are divided into two groups of columns according to the value of j; and finally, in point crossover the parents are divided into two blocks of cells, one formed by the first i rows and the first j columns, and other formed by the remaining cells.

Shiba *et al.* [85] proposed a genetic algorithm to construct test suites. The approach of this algorithm is to generate one test case at a time. To generate a test case the algorithm creates P random candidate solutions. These candidate solutions are evolved until the best solution is not improved in T consecutive generations. The fitness of a candidate solution S is defined as the number of tuples not covered in the initial test but covered in S. The crossover operator consists in exchanging with a probability of 0.5 the values of every position of two candidate solutions. The mutation operator replaces the value of one position of the candidate solution with the value of another position of the same candidate solution.

D. ANT COLONY SYSTEM

Chen *et al.* [87] constructed test suites by using an ant colony strategy. Test suites are constructed following a one-test-at-a-time strategy. The first step of the method is to construct a directed graph $G = (F, E)$ that represents the solution space. The set of nodes F is formed by nodes $f_1, \ldots, f_k$, where each f_i denotes a factor, and by a special node *End* that only has incoming edges. The set of all edges leaving node f_i represents the symbol set of f_i. A path from the first node f_1 to the final node *End* is a valid test. To construct a test case, a set of ants is placed in node f_1, and from this node every ant generates a test case by applying an edge selection rule that is directed by the pheromone information. The winner test case is the one that covers the greatest number of uncovered tuples.

E. MEMETIC ALGORITHM

Rodriguez-Tello and Torres-Jimenez [84] presented a memetic algorithm to construct binary CAs of strength three. In this algorithm the population is initialized randomly, but the symbols are balanced in every column of an individual. At every generation the population is partitioned into groups of four individuals. Within every group, the two fittest individuals are recombined to generate two offspring; these offspring are improved by means of a local search operator, and finally the two individuals with lower fitness are replaced by the improved offspring. The recombination operator randomly selects a row index i, then the rows of the two parents are divided into two sets of rows: the rows with indices less than or equal to i, and the rows with indices greater than i. The two offspring are formed by taking a set of rows from one parent and a set of rows from the other parent. The local search operator is formed by two neighborhood functions that make small changes in the offspring; these changes are: switching an element, and exchanging two elements in a column. Simulated annealing is used as the local search technique.

F. PARTICLE SWARM OPTIMIZATION

The work of Ahmed *et al.* [89] constructs uniform and variable strength CAs by means of a particle swarm optimization algorithm called PSTG (Particle Swarm-based t-way Test Generator). Here we describe the algorithm in the context of uniform CAs. The first step in PSTG is to generate the interaction elements (IEs), which are the t-tuples that must be covered in a combination of t parameters (or columns). For example, suppose the CA to be constructed has strength $t = 2$, order $v = 3$, and $k = 5$ parameters; then, for each of the $\binom{5}{2} = 10$ combinations of $t = 2$ parameters there are $3^2 = 9$ IEs. Take for example the parameters 2 and 5; one of the nine IEs for this combination of parameters is $(* \; 2 \; * \; * \; 0)$, which covers the tuple $(2, 0)$ in parameters 2 and 5; the positions corresponding to parameters 1, 3, and 4 contain $*$ in the IEs. The set of all IEs is stored in a list called Ps.

The particles are k-dimensional vectors $X_j = (X_{j,1}, X_{j,2}, \ldots, X_{j,k})$, where each dimension represents a parameter and contains an integer of the set $\{0, 1, \ldots, v - 1\}$; that is, a particle is a candidate test case, or a row of the CA. As the algorithm iterates, the velocity and position of the particles are updated using standard equations of PSO. For each particle PSTG computes its weight, that is defined as the number of IEs covered by the particle. If a particle has maximum weight then that particle is added to the final CA, and the IEs covered by the particle are removed from the *Ps* list. In the above example the maximum weight is $\binom{5}{2} = 10$, which is the maximum number of t-tuples that a single test case can cover. If no particle has the maximum possible weight then PSTG chooses the particle covering most IEs as the local best solution *lBest*. In the following iterations the positions of the particles are updated taking into account the current value of *lBest* and if a better *lBest* is found then *lBest* is updated. When *lBest* is not further improved, PSTG sets *lBest* as the global best *gBest* and adds it to the final CA. The algorithm iterates until *Ps* is empty.

G. CUCKOO SEARCH

Ahmed *et al.* [92] introduced a cuckoo search method to construct CAs. The method starts by generating a list of the t-tuples to be covered. If there are k parameters then the elements of this list are length-k vectors with elements from $\{0, 1, \ldots, v-1\}$ for the parameters in the combination and with $*$ for parameters not in the combination. For example if $t = 3$, $v = 2$, and $k = 7$, the vector $(1\ 0 * * \ 0 * *)$ represents the tuple $(1, 0, 0)$ for the combination of parameters $\{1, 2, 5\}$. After generating the t-tuples list, a random population of nests is created; each nest represents a candidate test case, and the weight of a nest is the number of t-tuples covered by the nest that are not covered in the current partial CA. The current partial CA is constructed by adding one test case at a time. In every iteration of the algorithm the nests are sorted according to their weights; the nests with the lowest weights are abandoned, so a Lévy flight is performed to replace the current test case by another one. For the nests with the better weighs also a Lévy flight is performed, but they are updated only if a better test case is obtained. Nests reaching the maximum possible weight are added to the final CA; if no nest reaches the maximum weight after a predefined number of iterations then the nest with the best weight is added to the final CA.

(a)

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&1&1&1&0&0&0&1&1&1\\
0&0&1&1&0&1&1&0&1&1&0&0&1\\
0&0&1&1&1&0&1&1&0&1&0&1&0\\
0&0&1&1&1&1&0&1&1&0&1&0&0\\
0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&1&1&1&1&1&1&1\\
1&0&0&1&1&1&1&1&1&1&1&1&1\\
1&0&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&1&1&1&1&0&0&1&0\\
1&1&0&0&1&0&1&0&1&1&1&0&0\\
1&1&0&0&1&1&0&1&0&1&0&0&1\\
1&1&1&1&0&0&1&1&0&0&1&0&1\\
1&1&1&1&0&1&0&0&0&1&1&1&0\\
1&1&1&1&1&0&0&0&1&0&0&1&1
\end{pmatrix}$$

(b)

$$\begin{pmatrix}
0&0&0&0&*&*&*&*&*&*&0&0&0\\
0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&1&1&1&0&0&0&1&1&1\\
0&0&1&1&0&1&1&0&1&1&0&0&1\\
0&0&1&1&1&0&1&1&0&1&0&1&0\\
0&0&1&1&1&1&0&1&1&0&1&0&0\\
0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&1&1&1&1&1&1&1\\
1&0&0&1&1&1&1&1&1&1&1&1&1\\
1&0&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&1&1&1&1&0&0&1&0\\
1&1&0&0&1&0&1&0&1&1&1&0&0\\
1&1&0&0&1&1&0&1&0&1&0&0&1\\
1&1&1&1&0&0&1&1&0&0&1&0&1\\
1&1&1&1&0&1&0&0&0&1&1&1&0\\
1&1&1&1&1&0&0&0&1&0&0&1&1
\end{pmatrix}$$

FIGURE 24. Detection of redundant elements in a CA(16; 13, 3, 2).

VII. TRANSFORMATIONS

A given CA can be subject to a set of changes to modify its internal structure without affecting the property of being a CA, or to obtain another CA with distinct parameters. We classified as *transformations* the methods that follow one of these two approaches. Postoptimization algorithms such as the randomized method of Nayeri *et al.* [99], the metaheuristic method of Torres-Jimenez and Rodriguez-Cristerna [100], and the graph-based method of Perez-Torres and Torres-Jimenez [101], belong to the first class of transformation methods.

A. DETECTION OF WILDCARDS

Sometimes a CA contains elements that can be freely modified without affecting the coverage properties of the CA. These elements are called *redundant elements* or *wildcards*. Fig 24(a) shows a CA(16; 3, 13, 2), and Fig 24(b) shows the same CA with redundant elements replaced by asterisks; it can be checked that the matrix of Fig 24(b) is still a CA of strength three. From [57] we know that the CA(16; 3, 13, 2) is optimal; therefore, even optimal CAs can have redundant elements. In non-optimal CAs it can happen that all elements of a row are redundant, and that row can be deleted to reduce the size of the CA.

The postoptimization method of Nayeri *et al.* [99] is based on detecting wildcards to select the rows to be eliminated. The method of Kim *et al.* [102] replaces wildcards in a CA of strength two by other valid symbols; this is done with the objective of finding CAs of strength two that have high desirable properties, such as a good coverage of 3-tuples.

B. PROJECTION

Colbourn [103] introduced two projection techniques for strength $t = 2$. The first one takes an MCA$(N; 2, k, (v_0, \ldots, v_{k-1}))$ to produce an MCA$(N-1; 2, k+1, (v_0 - 1, \ldots, v_{k-1} - 1, s))$ for some $s \leq k$. This construction reduces in one unit the order of every column $j \in \{0, \ldots, k-1\}$, but adds a new column of order s. The second construction takes an OA(v^2; 2, $v+1$, v) to produce an MCA(v^2-r; 2, $v+1+r$, $(v-r)^{v+1}s^r$) for some $1 \le r \le v$ and $1 \le s \le v-r$. When $s = v-r$ the result is a CA(v^2-r; 2, $v+1+r$, $v-r$). This construction is based on repeating r times the first construction, deleting one symbol at each stage.

We will illustrate the second projection technique with inputs A = OA(25; 2, 6, 5), $r = 2$, and $s = 5-2 = 3$. As shown in Fig 25(a) the first $v = 5$ rows of A have five constant entries and the last entry is equal to 0. Denote by x the first row of A. Partition the rows of A other than x in $s+1 = 4$ classes C_0, C_1, C_2, and D, where class C_0 contains the rows of A that have symbol $x_0 = 0$ in column 0, class C_1 contains the rows of A that have symbol $x_1 = 0$ in column 1, class C_2 contains the rows of A that have symbol $x_2 = 0$ in column 2, and class D contains the rows not in C_0, C_1, or C_2. Then $C_0 = \{5, 10, 15, 20\}$, $C_1 = \{9, 13, 17, 21\}$, $C_2 = \{8, 11, 19, 22\}$, and $D = \{1, 2, 3, 4, 6, 7, 12, 14, 16, 18, 23, 24\}$. Delete row x from A. For $i = 0, 1, 2$ replace the column i of the four rows of C_i by a permutation of the symbols $\{2, 3, 4\}$ plus one wildcard ($*$). In this way, all occurrences of symbol 0 are deleted in columns 0, 1, and 2 of A. In the columns 3, 4, and 5 of A replace every occurrence of symbol 0 by $*$. Add a new column to A such that for $l = 1, \dots, N-1$ the symbol of the new column is equal to i if $l \in C_i$, or the symbol is equal to $*$ if $l \in D$. The result is the array A' shown in Fig 25(b).

Now, take A' as the base array. Denote by x the first row of A'. Partition the rows of A' other than x in $s+1 = 4$ classes C_0, C_1, C_2, and D, where for $i = 0, 1, 2$ class C_i contains the rows of A' that have symbol $x_i = 1$ in column i, and class D contains the rows not in C_0, C_1, or C_2. Then $C_0 = \{5, 10, 15, 20\}$, $C_1 = \{4, 13, 17, 21\}$, $C_2 = \{8, 11, 14, 22\}$, and $D = \{1, 2, 3, 6, 7, 9, 12, 16, 18, 19, 23\}$. Delete row x from A'. For $i = 0, 1, 2$ replace the column i of the four rows of C_i by a permutation of the symbols $\{2, 3, 4\}$ plus one wildcard. In the columns 3, 4, and 5 of A' replace every occurrence of symbol 1 by $*$. Add a new column to A' such that for $l = 1, \dots, N-1$ the symbol of the new column is equal to i if $l \in C_i$, or the symbol is equal to $*$ if $l \in D$. The result is the array A'' shown in Fig 25(b). Finally, for $i = 0, 1, 2$ replace the last $r = 2$ wildcards of row i by symbol i to get a CA(23; 2, 8, 3).

C. FUSION

Colbourn *et al.* [16] introduced the *fusion* technique. For general CAs the fusion operation reduces in one unit the order of the CA, but deletes two rows. For OAs with $t = 2$ and $k \le v+1$ three rows are deleted. We will describe the first case. Given a CA(N; t, k, v) permute symbols in the k columns to obtain a constant row with symbols equal to $v-1$. Delete this row and choose a second row R with entries $(r_0, \dots, r_{k-1})$. In all rows other than R whenever a symbol $v-1$ occurs in column j replace the symbol $v-1$ by r_j if $r_j \le v-2$, and replace the symbol $v-1$ by any symbol in $\{0, \dots, v-2\}$ if $r_j = v-1$. The result is CA($N-2$; t, k, $v-1$).

Rodriguez-Cristerna [104] generalized the fusion operator to MCAs. The generalized fusion operator (GFO) uses a well

(a)

$$\begin{pmatrix}
0&0&0&0&0&0\\
1&1&1&1&1&0\\
2&2&2&2&2&0\\
3&3&3&3&3&0\\
4&4&4&4&4&0\\
0&1&2&3&4&1\\
1&2&3&4&0&1\\
2&3&4&0&1&1\\
3&4&0&1&2&1\\
4&0&1&2&3&1\\
0&2&4&1&3&2\\
1&3&0&2&4&2\\
2&4&1&3&0&2\\
3&0&2&4&1&2\\
4&1&3&0&2&2\\
0&3&1&4&2&3\\
1&4&2&0&3&3\\
2&0&3&1&4&3\\
3&1&4&2&0&3\\
4&2&0&3&1&3\\
0&4&3&2&1&4\\
1&0&4&3&2&4\\
2&1&0&4&3&4\\
3&2&1&0&4&4\\
4&3&2&1&0&4
\end{pmatrix}$$

(b)

$$\begin{pmatrix}
1&1&1&1&1&*&*\\
2&2&2&2&2&*&*\\
3&3&3&3&3&*&*\\
4&4&4&4&4&*&*\\
2&1&2&3&4&1&0\\
1&2&3&4&*&1&*\\
2&3&4&*&1&1&*\\
3&4&2&1&2&1&2\\
4&2&1&2&3&1&1\\
3&2&4&1&3&2&0\\
1&3&3&2&4&2&2\\
2&4&1&3&*&2&*\\
3&3&2&4&1&2&1\\
4&1&3&*&2&2&*\\
4&3&1&4&2&3&0\\
1&4&2&*&3&3&*\\
2&4&3&1&4&3&1\\
3&1&4&2&*&3&*\\
4&2&4&3&1&3&2\\
*&4&3&2&1&4&0\\
1&*&4&3&2&4&1\\
2&1&*&4&3&4&2\\
3&2&1&*&4&4&*\\
4&3&2&1&*&4&*
\end{pmatrix}$$

(c)

$$\begin{pmatrix}
2&2&2&2&2&*&*&*\\
3&3&3&3&3&*&*&*\\
4&4&4&4&4&*&*&*\\
2&2&2&3&4&*&0&1\\
2&2&3&4&*&*&*&0\\
2&3&4&*&*&*&*&*\\
3&4&2&*&2&*&2&*\\
4&2&2&2&3&*&1&2\\
3&2&4&*&3&2&0&*\\
3&3&3&2&4&2&2&0\\
2&4&3&3&*&2&*&2\\
3&3&2&4&*&2&1&*\\
4&3&3&*&2&2&*&1\\
4&3&4&4&2&3&0&2\\
4&4&2&*&3&3&*&0\\
2&4&3&*&4&3&1&*\\
3&4&4&2&*&3&*&1\\
4&2&4&3&*&3&2&*\\
&4&3&2&&4&0&*\\
&&4&3&2&4&1&0\\
2&*&*&4&3&4&2&1\\
3&2&*&*&4&4&*&2\\
4&3&2&*&*&4&*&*
\end{pmatrix}$$

FIGURE 25. Example of projection. (a) The initial OA(25; 2, 6, 5). (b) The array after the first projection. (c) The array after the second projection.

designed combination of procedures that involve a main cycle that detects redundant symbols and reduces the number of rows by exploiting redundant symbols. When the main cycle can not be applied (that is, the resulting MCA has missing tuples) a simulated annealing algorithm is used. If the SA reduces the number of missing tuples to zero the main cycle runs one more time, otherwise GFO stops.

D. DERIVATION

In Colbourn *et al.* [16] it is described a technique called *derivation*. This technique allows the reduction of the strength in one unit while the number of columns is also reduced in one unit. The following inequality taken from [16] defines the technique:

$$\mathrm{CAN}(t, k, v) \leq \left\lfloor \frac{\mathrm{CAN}(t+1, k+1, v)}{v} \right\rfloor$$

In a given CA(N; $t+1$, $k+1$, v) select any column j and permute the rows of the CA in such a way the elements of column j are sorted in lexicographic order. In this manner, the N rows are partitioned into $v+1$ subsets, where for $0 \leq i \leq v$ the i-th subset contains the rows that have symbol i in column j. Take the subset that has the less number of rows and delete column j in this subset of rows; the result is CA with k columns, strength t, and number of rows at most $\lfloor N/v \rfloor$.

One example of the use of the derive method is the CA(12; 3, 11, 2), which was derived from CA(24; 4, 12, 2). Fig 26 shows a covering array A = CA(24; 4, 12, 2) where the elements of the first column are sorted in lexicographic order. The covering array B = CA(12; 3, 11, 2) is obtained by taking the first 12 rows and the last 11 columns of A.

$$A = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&1&1&1&0&0&0&1&1&1\\
0&0&1&0&1&1&0&1&1&0&0&1\\
0&0&1&1&0&1&1&0&1&0&1&0\\
0&0&1&1&1&0&1&1&0&1&0&0\\
0&1&0&0&1&1&1&1&0&0&1&0\\
0&1&0&1&0&1&0&1&1&1&0&0\\
0&1&0&1&1&0&1&0&1&0&0&1\\
0&1&1&0&0&1&1&0&0&1&0&1\\
0&1&1&0&1&0&0&0&1&1&1&0\\
0&1&1&1&0&0&0&1&0&0&1&1\\
1&0&0&0&1&1&1&0&1&1&0&0\\
1&0&0&1&0&1&1&1&0&0&0&1\\
1&0&0&1&1&0&0&1&1&0&1&0\\
1&0&1&0&0&1&0&1&0&1&1&0\\
1&0&1&0&1&0&1&0&0&0&1&1\\
1&0&1&1&0&0&0&0&1&1&0&1\\
1&1&0&0&0&1&0&0&1&0&1&1\\
1&1&0&0&1&0&0&1&0&1&0&1\\
1&1&0&1&0&0&1&0&0&1&1&0\\
1&1&1&0&0&0&1&1&1&0&0&0\\
1&1&1&1&1&1&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

$$B = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1\\
0&0&1&1&1&0&0&0&1&1&1\\
0&1&0&1&1&0&1&1&0&0&1\\
0&1&1&0&1&1&0&1&0&1&0\\
0&1&1&1&0&1&1&0&1&0&0\\
1&0&0&1&1&1&1&0&0&1&0\\
1&0&1&0&1&0&1&1&1&0&0\\
1&0&1&1&0&1&0&1&0&0&1\\
1&1&0&0&1&1&0&0&1&0&1\\
1&1&0&1&0&0&0&1&1&1&0\\
1&1&1&0&0&0&1&0&0&1&1
\end{pmatrix}$$

FIGURE 26. Example of derivation. B = CA(12; 3, 11, 2) was derived from A = CA(24; 4, 12, 2).

VIII. GENERAL CHARACTERISTICS OF THE CLASSES OF METHODS

In this section we analyze some general features of the classes in which the methods studied were grouped. We also give some guidelines about what kind of methods are more convenient to employ according to the values of t, k, and v, and based on the time and computational resources willing to invest in the construction of the CA.

Algebraic methods construct the CAs very fast, given that no search or little search is done. Moreover, some algebraic methods yield optimal CAs. A drawback of this kind of methods is that they are not applicable to all combinations of values of t, k, and v. For example, the methods based on LFSR sequences [38], [39] only work for v prime-power. However, when the target CA has appropriate parameters t, k, v for an algebraic method, then there is a good chance that algebraic approaches are the best option. Another area where algebraic algorithms are good options is for binary CAs with large k and t; the methods of constant weight vectors [30] and binomial coefficients [32] can construct arbitrarily large binary CAs. Also very large CAs can be constructed by cyclotomy [37]; in this case the restrictions are that k must be prime-power and v must be related to k in such a way $k \equiv 1 \pmod{v}$.

Recursive algorithms are also fast algorithms that generally do not perform a computational search to construct the final CA; some exceptions are augmented annealing [43] that uses simulated annealing to construct some parts of the final CA, and tower of covering arrays [47] where the next CA in the tower is constructed by exploring a number of arrangements of the columns of the input CA. On the other hand, the classical recursive techniques of product, Roux-type constructions, and powering only takes the input ingredients and construct the output CA by following a fixed procedure to combine the inputs; in this methods the size of the final CA is known in advance, and so the quality of the output CA depends on the quality of the input ingredients. This kind of algorithms are the more convenient option for constructing CAs with large k and t; for many of such cases they currently give the best-known CAs (reported in [20]). A disadvantage of recursive methods is the excessive redundancy in the final CAs; this redundancy is needed to ensure that the final result is a CA. Generally, non-recursive techniques provides better solutions when they can be applied to the given parameters t, k, and v. Similarly to algebraic methods, recursive constructions are only applicable for certain CA parameters.

Exact algorithms have the advantage of generating optimal CAs. Also, near-optimal CAs can be constructed by exact methods if the search is not exhaustive. However, they are applicable only to small CAs. The main use of exact algorithms is in proving the optimality of CAs, as in [57], or in improving the lower bound of some CAs, as in [54]. For instances where exact methods are not applicable, metaheuristic methods generally construct those instances in a very short time, but without ensuring the optimality of the constructed CA.

The class of greedy algorithms can be subdivided into two subclasses: methods that construct the CA *one-row-at-a-time*, and methods that construct the CA *one-parameter-at-a-time*. Most greedy methods belong to the first subclass, and the most successful of them is the deterministic density algorithm [67] and its variants; for the second subclass the most successful method are in-parameter-order [71] and its variants, and the recent method to construct CPHFs [75]. These methods are good options to balance the quality of the solutions and the execution time, because they are generally faster than metaheuristic methods and the quality of the solutions is acceptable; in fact a lot of the current upper bounds of CAN reported in [20] were generated by greedy methods. In addition, most of the greedy methods can be used with any values of t, k, and v, that is, they are general constructions. Greedy methods execute faster than exact and metaheuristic methods, so they are good option to construct CAs with sizes such that metaheuristic methods take to much time. Other positive characteristic of greedy methods is that they can be adapted to construct other kind of arrays such as MCAs and variable strength CAs. For cases where metaheuristic methods can be executed without taking to much time, the result of greedy methods is generally inferior to the result of metaheuristic methods.

Metaheuristic methods are a good option to construct small and medium-size CAs, say tens and hundreds of columns. For small CAs this kind of methods give results very close to the optimal ones, but as the size of the CA increases the results of metaheuristic methods also move away from the estimated lower bounds. Metaheuristic methods are also general methods that can be used to construct CAs with any mixture of parameters t, k, v; but for large parameter values these methods are slow or can not be executed at all, due to some of them require data structures that grows as the CA parameters grow; for example, the particle swarm optimization method developed in [89] requires a list of size $\binom{k}{t} \cdot k$. Currently, the development of metaheuristic methods to construct CAs is very active, especially for constructing MCAs, variable strength CAs, and constrained CAs. For uniform CAs the most successful techniques have been simulated annealing and tabu search.

Transformation methods can be applied to any CA, although the best results have been obtained by applying postoptimization techniques to CAs constructed with greedy methods, as in [100]; also, postoptimization is recommended with CAs produced by recursive techniques because those CAs have a lot of redundancy. The fusion method have reported some of the best-known CAs, particularly for non-prime-power orders where the application of fusion to CAs constructed using LFSR sequences and from CPHFs produce good CAs (see [20]).

IX. FUTURE IMPROVEMENTS ON THE SIZE OF UNIFORM COVERING ARRAYS

The current best sizes of covering arrays (or the current upper bounds of CAN) with strengths $2 \leq t \leq 6$ and orders $2 \leq v \leq 25$ are reported in the Covering Array Tables [20]. These tables have been used as the main point of comparison to report improvements on the upper bounds of covering array numbers. Next, we will give some of the most representative constructions that currently appear in the tables for each strength $2 \leq t \leq 6$:

$t = 2$: Simulated annealing, product, starter vectors, projection.
$t = 3$: Simulated annealing, augmented annealing, two stage, duplication, LFSR sequences, CPHFs, power, cyclotomy.
$t = 4$: Cyclotomy, Roux-type construction, group construction, power, CPHFs, LFSR sequences, DDA, IPO.
$t = 5$: Cyclotomy, Roux-type construction, power, CPHFs, DDA, IPO, augmentation.
$t = 6$: Cyclotomy, Roux-type construction, power, CPHFs, DDA, IPO.

In the above list only the generic method is mentioned, but we will assume that all specializations of the method are included. In addition, the transformation approaches of wildcard detection, fusion, and derivation have also contributed to reach some of the current best upper bounds of CAN.

Although each time it is more difficult to improve the size of uniform CAs, we will make some comments about which type of algorithms may continue to improve the current sizes in the near future. The comments are based only on existing methods to construct CAs, but of course completely new methods may appear in the near future. Firstly, let us make a very simply partition of the universe of CAs based on the number of columns they have:

- *Small*: CAs with $k \leq 100$ columns.
- *Medium*: CAs with $100 < k \leq 1000$ columns.
- *Large*: CAs with $1000 < k \leq 10000$ columns.

For small CAs and small orders (say $v \leq 7$) metaheuristic and greedy algorithms have a good chance to continue improving the current upper bounds; especially simulated annealing, which has been the most successful metaheuristic technique. For larger alphabets the methods based on CPHFs (Subsection V-G) may become more important. In a very recent work of Colbourn *et al.* [75] CPHFs were used to improve the asymptotic upper bound of CAN(t, k, v), but also concrete CPHFs whose derived CAs improve a current bound were given. Other candidates to improve the current upper bounds are methods that use starter vectors.

In the case of medium CAs the two-stage methods seem to be good candidates to improve the current upper bounds. It is probable that *multi-stage* algorithms will be developed in the near future; these algorithms may combine recursive or algebraic constructions, with greedy and metaheuristic constructions, plus some postoptimization technique. On the other hand, the methods based on LFSR sequences (Subsection II-G) may also be extended to greater strengths (currently there is only one direct construction for strength three, and one construction involving computer search for strength four). CPHF-based methods are expected to play an important role in this range of columns because they allow the generation of medium CAs using metaheuristic and greedy techniques. Also CPHF-based methods could be part of multi-stage algorithms.

Probably for larger CAs the recursive constructions of product, power, and Roux-type will continue to be the main constructions. As soon as some small or medium CAs are improved, they can be used in the recursive constructions to improve some large CAs. On the other hand, the cyclotomy technique may be benefited from parallel computing, because its main restriction is the time it takes to verify that a cyclotomic matrix is a CA. Finally, as for the other two cases CPHF-based methods are expected to contribute some new upper bounds on the size of large CAs.

X. CONCLUSIONS

This paper presented a summary of different approaches for the construction of uniform covering arrays. The revised methods were grouped into algebraic, recursive, exact, greedy, metaheuristic, and transformation techniques. Most of the analyzed methods were accompanied with examples and/or pseudocodes; in fact one important difference with previous related works is that the strategy of several methods was clarified with an example, especially in the case of algebraic and recursive algorithms. We studied methods with very different strategies in each class of methods to give an idea of the available strategies to construct uniform covering arrays. The final part of the document provides a discussion about the general characteristics of each class of methods.

ACKNOWLEDGEMENTS

The authors acknowledge "Xiuhcoatl"-CGSTIC of CINVESTAV for providing access of high performance computing.

REFERENCES

[1] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *IT Prof.*, vol. 10, no. 3, pp. 19–23, May 2008.

[2] D. R. Wallace and D. R. Kuhn, "Failure modes in medical device software: an analysis of 15 years of recall data," *Int. J. Rel., Quality Saf. Eng.*, vol. 8, no. 4, pp. 351–371, 2001.

[3] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Proc. 27th Annu. NASA Goddard Softw. Eng. Workshop (SEW-)*, Washington, DC, USA, Dec. 2002, pp. 91–95.

[4] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Trans. Softw. Eng.*, vol. 30, no. 6, pp. 418–421, Jun. 2004.

[5] D. R. Kuhn and V. Okum, "Pseudo-exhaustive testing for software," in *Proc. 30th Annu. IEEE/NASA Softw. Eng. Workshop*, Washington, DC, USA, Apr. 2006, pp. 153–158.

[6] X. Yuan, M. B. Cohen, and A. M. Memon, "GUI interaction testing: Incorporating event context," *IEEE Trans. Softw. Eng.*, vol. 37, no. 4, pp. 559–574, Jul./Aug. 2011.

[7] P. Yang, X. Tan, H. Sun, D. Chen, and C. Li, "Fire accident reconstruction based on LES field model by using orthogonal experimental design method," *Adv. Eng. Softw.*, vol. 42, no. 11, pp. 954–962, 2011.

[8] D. E. Shasha, A. Y. Kouranov, L. V. Lejay, M. F. Chou, and G. M. Coruzzi, "Using combinatorial design to study regulation by multiple input signals. A tool for parsimony in the post-genomics era," *Plant Physiol.*, vol. 127, no. 4, pp. 1590–1594, 2001.

[9] H. Ordoñez, J. Torres-Jimenez, A. Ordoñez, and C. Cobos, "Clustering business process models based on multimodal search and covering arrays," in *Advances in Soft Computing*, O. Pichardo-Lagunas and S. Miranda-Jiménez, Eds. Cham, Switzerland: Springer, 2016, pp. 317–328.

[10] G. O. H. Katona, "Two applications (for search theory and truth functions) of Sperner type theorems," *Periodica Math. Hungarica*, vol. 3, nos. 1–2, pp. 19–26, 1973.

[11] D. J. Kleitman and J. Spencer, "Families of k-independent sets," *Discrete Math.*, vol. 6, no. 3, pp. 255–262, 1973.

[12] K. A. Bush, "Orthogonal arrays of index unity," *Ann. Math. Statist.*, vol. 23, no. 3, pp. 426–434, 1952.

[13] C. J. Colbourn, *CRC Handbook of Combinatorial Designs*. Boca Raton, FL, USA: CRC Press, 1996.

[14] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, *Orthogonal Arrays*. New York, NY, USA: Springer-Verlag, 1999.

[15] K. A. Johnson and R. Entringer, "Largest induced subgraphs of the n-cube that contain no 4-cycles," *J. Combinat. Theory, B*, vol. 46, no. 3, pp. 346–355, 1989.

[16] C. J. Colbourn, G. Kéri, P. P. R. Soriano, and J.-C. Schlage-Puchta, "Covering and radius-covering arrays: Constructions and classification," *Discrete Appl. Math.*, vol. 158, no. 11, pp. 1158–1180, Jun. 2010.

[17] S. Choi, H. K. Kim, and D. Y. Oh, "Structures and lower bounds for binary covering arrays," *Discrete Math.*, vol. 312, no. 19, pp. 2958–2968, 2012.

[18] N. Francetić and B. Stevens, "Asymptotic size of covering arrays: An application of entropy compression," *J. Combinat. Des.*, vol. 25, no. 6, pp. 243–257, 2017.

[19] K. Sarkar and C. J. Colbourn, "Upper bounds on the size of covering arrays," *SIAM J. Discrete Math.*, vol. 31, no. 2, pp. 1277–1293, 2017.

[20] C. J. Colbourn, *Covering Array Tables for $t = 2, 3, 4, 5, 6$*. Accessed: Sep. 6, 2017. [Online]. Available: http://www.public.asu.edu/ ccolbou/src/tabby/catable.html

[21] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms* (Operations Research/Computer Science Interfaces Series), vol. 34, M. C. Golumbic and I. B.-A. Hartman, Eds. New York, NY, USA: Springer, 2005, pp. 237–266.

[22] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche*, vol. 59, nos. 1–2, pp. 125–172, 2004.

[23] J. Lawrence, R. N. Kacker, Y. Lei, D. R. Kuhn, and M. Forbes, "A survey of binary covering arrays," *Electron. J. Combinat.*, vol. 18, no. 1, 2011, Art. no. P84.

[24] V. V. Kuliamin and A. A. Petukhov, "A survey of methods for constructing covering arrays," *Program. Comput. Softw.*, vol. 37, no. 3, pp. 121–146, 2011.

[25] J. Torres-Jimenez and I. Izquierdo-Marquez, "Survey of covering arrays," in *Proc. 15th Int. Symp. Symbolic Numeric Algorithms Sci. Comput. (SYNASC)*, Sep. 2013, pp. 20–27.

[26] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, Feb. 2011.

[27] S. K. Khalsa and Y. Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, Nov. 2014, pp. 323–334.

[28] J. Zhang, Z. Zhang, and F. Ma, *Automatic Generation of Combinatorial Test Data*. Berlin, Germany: Springer, 2014.

[29] A. Rényi, *Foundations of Probability*. Hoboken, NJ, USA: Wiley, 1971.

[30] D. T. Tang and L. S. Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Trans. Comput.*, vol. C-32, no. 12, pp. 1145–1150, Dec. 1983.

[31] J. Martinez-Pena and J. Torres-Jimenez, "A branch and bound algorithm for ternary covering arrays construction using trinomial coefficients," *Res. Comput. Sci.*, vol. 49, no. 1, pp. 61–71, 2010.

[32] J. Torres-Jimenez, I. Izquierdo-Marquez, A. Gonzalez-Gomez, and H. Avila-George, "A branch & bound algorithm to derive a direct construction for binary covering arrays," in *Advances in Artificial Intelligence and Soft Computing*, G. Sidorov and S. Galicia-Haro, Eds. Cham, Switzerland: Springer, 2015, pp. 158–177.

[33] M. Chateauneuf and D. L. Kreher, "On the state of strength-three covering arrays," *J. Combinat. Des.*, vol. 10, no. 4, pp. 217–238, 2002.

[34] K. Meagher and B. Stevens, "Group construction of covering arrays," *J. Combinat. Des.*, vol. 13, no. 1, pp. 70–77, 2005.

[35] J. R. Lobb, C. J. Colbourn, P. Dazinger, B. Stevens, and J. Torres-Jimenez, "Cover starters for covering arrays of strength two," *Discrete Math.*, vol. 312, pp. 943–956, Mar. 2012.

[36] Y. Akhtar, S. Maity, and R. C. Chandrasekharan, "Covering arrays of strength four and software testing," in *Mathematics and Computing*, R. Mohapatra, D. Chowdhury, and D. Giri, Eds. New Delhi, India: Springer, 2015, pp. 391–398.

[37] C. J. Colbourn, "Covering arrays from cyclotomy," *Des., Codes Cryptogr.*, vol. 55, nos. 2–3, pp. 201–219, 2010.

[38] S. Raaphorst, L. Moura, and B. Stevens, "A construction for strength-3 covering arrays from linear feedback shift register sequences," *Des., Codes Cryptogr.*, vol. 73, no. 3, pp. 949–968, 2014.

[39] G. Tzanakis, L. Moura, D. Panario, and B. Stevens, "Constructing new covering arrays from LFSR sequences over finite fields," *Discrete Math.*, vol. 339, no. 3, pp. 1158–1171, 2016.

[40] G. B. Sherwood, S. S. Martirosyan, and C. J. Colbourn, "Covering arrays of higher strength from permutation vectors," *J. Combinat. Des.*, vol. 14, no. 3, pp. 202–213, 2006.

[41] J. Torres-Jimenez and I. Izquierdo-Marquez, "Covering arrays of strength three from extended permutation vectors," *Des., Codes Cryptogr.*, vol. 86, no. 11, pp. 2629–2643, Nov. 2018. doi: 10.1007/s10623-018-0465-6.

[42] G. Roux, "k-propriétés dans des tableaux de n colonnes; cas particulier de la k-surjectivité et de la k-permutivité," Ph.D. dissertation, Dept. Math., Univ. Paris, Paris, France, 1987.

[43] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, "Constructing strength three covering arrays with augmented annealing," *Discrete Math.*, vol. 308, no. 13, pp. 2709–2722, 2008.

[44] C. J. Colbourn, S. S. Martirosyan, T. Van Trung, and R. A. Walker, II, "Roux-type constructions for covering arrays of strengths three and four," *Des., Codes Cryptogr.*, vol. 41, no. 1, pp. 33–57, Oct. 2006.

[45] S. Martirosyan and T. van Trung, "On t-covering arrays," *Des., Codes Cryptogr.*, vol. 32, nos. 1–3, pp. 323–339, 2004.

[46] C. J. Colbourn, S. S. Martirosyan, G. L. Mullen, D. Shasha, G. B. Sherwood, and J. L. Yucas, "Products of mixed covering arrays of strength two," *J. Combinat. Des.*, vol. 14, no. 2, pp. 124–138, 2006.

[47] J. Torres-Jimenez, I. Izquierdo-Marquez, R. N. Kacker, and D. R. Kuhn, "Tower of covering arrays," *Discrete Appl. Math.*, vols. 190–191, pp. 141–146, Aug. 2015.

[48] L. Ji, Y. Li, and J. Yin, "Constructions of covering arrays of strength five," *Des., Codes Cryptogr.*, vol. 62, no. 2, pp. 199–208, Feb. 2012.

[49] C. Colbourn and J. Torres-Jimenez, "Profiles of covering arrays of strength two," *J. Algorithms Comput.*, vol. 44, no. 1, pp. 31–60, 2013.

[50] C. J. Colbourn and J. Torres-Jimenez, "Heterogeneous hash families and covering arrays," *Contemp. Math.*, vol. 523, pp. 3–15, Sep. 2010.

[51] C. J. Colbourn and J. Zhou, "Improving two recursive constructions for covering arrays," *J. Stat. Theory Pract.*, vol. 6, no. 1, pp. 30–47, 2012.

[52] C. J. Colbourn, "Augmentation of covering arrays of strength two," *Graphs and Combinatorics*, vol. 31, no. 6, pp. 2137–2147, 2015.

[53] J. Bracho-Rios, J. Torres-Jimenez, and E. Rodriguez-Tello, "A new backtracking algorithm for constructing binary covering arrays of variable strength," in *Advances in Artificial Intelligence* (Lecture Notes in Computer Science), vol. 5845, A. H. Aguirre, R. M. Borja, and C. A. R. García, Eds. Berlin, Germany: Springer-Verlag, 2009, pp. 397–407.

[54] M. Banbara, H. Matsunaka, N. Tamura, and K. Inoue, "Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers," in *Proc. 17th Int. Conf. Logic Program., Artif. Intell., Reasoning (LPAR)*. Berlin, Germany: Springer-Verlag, 2010, pp. 112–126.

[55] J. Yan and J. Zhang, "Backtracking algorithms and search heuristics to generate test suites for combinatorial testing," in *Proc. 30th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1. Washington, DC, USA: IEEE Computer Society, Sep. 2006, pp. 385–394.

[56] J. Yan and J. Zhang, "A backtracking search tool for constructing combinatorial test suites," *J. Syst. Softw.*, vol. 81, no. 10, pp. 1681–1693, 2008.

[57] J. Torres-Jimenez and I. Izquierdo-Marquez, "Construction of non-isomorphic covering arrays," *Discrete Math., Algorithms Appl.*, vol. 8, no. 2, 2016, Art. no. 1650033.

[58] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith, "Constraint models for the covering test problem," *Constraints*, vol. 11, no. 2–3, pp. 199–219, Jul. 2006.

[59] D. Lopez-Escogido, J. Torres-Jimenez, E. Rodriguez-Tello, and N. Rangel-Valdez, "Strength two covering arrays construction using a SAT representation," in *Advances in Artificial Intelligence* (Lecture Notes in Computer Science), vol. 5317. Berlin, Germany: Springer, 2008, pp. 44–53.

[60] C. Ansótegui, I. Izquierdo, F. Manyà, and J. Torres-Jiménez, "A max-sat-based approach to constructing optimal covering arrays," in *Artificial Intelligence Research and Development* (Frontiers in Artificial Intelligence and Applications), vol. 256, K. Gibert, V. Botti, and R. Reig-Bolaño, Eds. Amsterdam, The Netherlands: IOS Press, 2013, pp. 51–59.

[61] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," *IEEE Softw.*, vol. 13, no. 5, pp. 83–88, Sep. 1996.

[62] Y.-W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proc. IEEE Aerosp. Conf.* Big Sky, MT, USA: IEEE Computer Society, vol. 1, Mar. 2000, pp. 431–437.

[63] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for T-Way software testing," in *Proc. 14th Annu. IEEE Int. Conf. Workshops Eng. Comput.-Based Syst. (ECBS)*. Tucson, AZ, USA: IEEE Computer Society, Mar. 2007, pp. 549–556.

[64] 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. Standards Technol.*, vol. 113, no. 5, pp. 287–297, 2008.

[65] A. H. Ronneseth and C. J. Colbourn, "Merging covering arrays and compressing multiple sequence alignments," *Discrete Appl. Math.*, vol. 157, no. 9, pp. 2177–2190, 2009.

[66] M. I. Younis, K. Z. Zamli, M. F. J. Klaib, Z. H. C. Soh, S. Abdullah, and N. Isa, "Assessing IRPS as an efficient pairwise test data generation strategy," *Int. J. Adv. Intell. Paradigms*, vol. 2, no. 1, pp. 90–104, 2010.

[67] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Softw. Test., Verification Rel.*, vol. 17, no. 3, pp. 159–182, 2007.

[68] A. Calvagna and A. Gargantini, "T-wise combinatorial interaction test suites construction based on coverage inheritance," *Softw. Test., Verification Rel.*, vol. 22, no. 7, pp. 507–526, 2012.

[69] C. Nie, J. Jiang, H. Wu, H. Leung, and C. J. Colbourn, "Empirically identifying the best greedy algorithm for covering array generation," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2013, pp. 239–248.

[70] R. C. Bryce and C. J. Colbourn, "A density-based greedy algorithm for higher strength covering arrays," *Softw. Test., Verification Rel.*, vol. 19, no. 1, pp. 37–53, 2009.

[71] Y. Lei and K.-C. Tai, "In-parameter-order: A test generation strategy for pairwise testing," in *Proc. 3rd IEEE Int. Symp. High-Assurance Syst. Eng. (HASE)*. Washington, DC, USA: IEEE Computer Society, Nov. 1998, pp. 254–261.

[72] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG-IPOG-D: Efficient test generation for multi-way combinatorial testing," *Softw. Test., Verification Rel.*, vol. 18, no. 3, pp. 125–148, 2008.

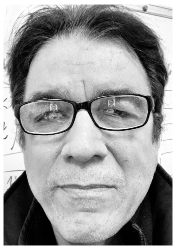
[73] J. Torres-Jimenez, H. Avila-George, and I. Izquierdo-Marquez, "A two-stage algorithm for combinatorial testing," *Optim. Lett.*, vol. 11, no. 3, pp. 457–469, 2017.

[74] K. Sarkar and C. J. Colbourn. (2016). "Two-stage algorithms for covering array construction." [Online]. Available: https://arxiv.org/abs/1606.06730

[75] C. J. Colbourn, E. Lanus, and K. Sarkar, "Asymptotic and constructive methods for covering perfect hash families and covering arrays," *Des., Codes Cryptogr.*, vol. 86, no. 4, pp. 907–937, 2018.

[76] C. J. Colbourn and E. Lanus, "Subspace restrictions and affine composition for covering perfect hash families," *Art Discrete Appl. Math.*, vol. 1, no. 2, pp. 1–19, 2018.

[77] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in *Proc. 25th Int. Conf. Softw. Eng.*, May 2003, pp. 38–48.
[78] H. Avila-George, J. Torres-Jimenez, L. Gonzalez-Hernandez, and V. Hernández, "Metaheuristic approach for constructing functional test-suites," *IET Softw.*, vol. 7, no. 2, pp. 104–117, Apr. 2013.
[79] J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," *Inf. Sci.*, vol. 185, no. 1, pp. 137–152, 2012.
[80] J. Torres-Jimenez and E. Rodriguez-Tello, "Simulated annealing for constructing binary covering arrays of variable strength," in *Proc. IEEE Congr. Evol. Comput.* Barcelona, Spain: IEEE Computer Society, Jul. 2010, pp. 4102–4109.
[81] J. Stardom, "Metaheuristics and the search for covering and packing arrays," M.S. thesis, Dept. Math., Simon Fraser Univ., Burnaby, BC, Canada, 2001.
[82] K. J. Nurmela, "Upper bounds for covering arrays by tabu search," *Discrete Appl. Math.*, vol. 138, nos. 1–2, pp. 143–152, 2004.
[83] L. Zekaoui, "Mixed covering arrays on graphs and tabu search algorithms," M.S. thesis, Ottawa-Carleton Inst. Comput. Sci., Univ. Ottawa, Ottawa, ON, Canada, 2006.
[84] E. Rodriguez-Tello and J. Torres-Jimenez, "Memetic algorithms for constructing binary covering arrays of strength three," in *Artifical Evolution* (Lecture Notes in Computer Science), vol. 5975. Berlin, Germany: Springer, 2010, pp. 86–97.
[85] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*. Hong Kong: IEEE Computer Society, Sep. 2004, pp. 72–77.
[86] R.-Z. Qi, Z.-J. Wang, and S.-Y. Li, "A parallel genetic algorithm based on spark for pairwise test suite generation," *J. Comput. Sci. Technol.*, vol. 31, no. 2, pp. 417–427, Mar. 2016.
[87] X. Chen, Q. Gu, A. Li, and D. Chen, "Variable strength interaction testing with an ant colony system approach," in *Proc. 16th Asia–Pacific Softw. Eng. Conf.*, Dec. 2009, pp. 160–167.
[88] T. Mahmoud and B. S. Ahmed, "An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use," *Expert Syst. Appl.*, vol. 42, no. 22, pp. 8753–8765, 2015.
[89] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Application of particle swarm optimization to uniform and variable strength covering array construction," *Appl. Soft Comput.*, vol. 12, no. 4, pp. 1330–1347, 2012.
[90] H. Wu, C. Nie, F. C. Kuo, H. Leung, and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 575–591, Aug. 2015.
[91] X. Bao, S. Liu, N. Zhang, and M. Dong, "Combinatorial test generation using improved harmony search algorithm," *Int. J. Hybrid Inf. Technol.*, vol. 8, no. 9, pp. 121–130, 2015.
[92] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm," *Inf. Softw. Technol.*, vol. 66, pp. 13–29, Oct. 2015.
[93] H. Avila-George, J. Torres-Jimenez, and V. Hernández, "New bounds for ternary covering arrays using a parallel simulated annealing," *Math. Problems Eng.*, vol. 2012, Jul. 2012, Art. no. 897027.
[94] Y. Jia, M. B. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction test generation strategies using hyperheuristic search," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 1, May 2015, pp. 540–550.
[95] K. Z. Zamli, B. Y. Alkazemi, and G. Kendall, "A tabu search hyper-heuristic strategy for t-way test suite generation," *Appl. Soft Comput.*, vol. 44, pp. 57–74, Jul. 2016.
[96] J. Torres-Jimenez and I. Izquierdo-Marquez, "A simulated annealing algorithm to construct covering perfect hash families," *Math. Problems Eng.*, vol. 2018, Jul. 2018, Art. no. 1860673.
[97] I. Izquierdo-Marquez, J. Torres-Jimenez, B. Acevedo-Juárez, and H. Avila-George, "A greedy-metaheuristic 3-stage approach to construct covering arrays," *Inf. Sci.*, vols. 460–461, pp. 172–189, Sep. 2018.
[98] R. A. Walker, II, and C. J. Colbourn, "Tabu search for covering arrays using permutation vectors," *J. Stat. Planning Inference*, vol. 139, no. 1, pp. 69–80, 2009.
[99] P. Nayeri, C. J. Colbourn, and G. Konjevod, "Randomized post-optimization of covering arrays," *Eur. J. Combinatorics*, vol. 34, no. 1, pp. 91–103, 2013.
[100] J. Torres-Jimenez and A. Rodriguez-Cristerna, "Metaheuristic post-optimization of the NIST repository of covering arrays," *CAAI Trans. Intell. Technol.*, vol. 2, no. 1, pp. 31–38, 2017.
[101] J. C. Perez-Torres and J. Torres-Jimenez, "A graph-based postoptimization approach for covering arrays," *Qual. Rel. Eng. Int.*, vol. 33, no. 8, pp. 2171–2180, Dec. 2017.
[102] Y. Kim, D.-H. Jang, and C. M. Anderson-Cook, "Selecting the best wild card entries in a covering array," *Qual. Rel. Eng. Int.*, vol. 33, no. 7, pp. 1615–1627, 2017.
[103] C. J. Colbourn, "Strength two covering arrays: Existence tables and projection," *Discrete Math.*, vol. 308, nos. 5–6, pp. 772–786, 2008.
[104] A. Rodriguez-Cristerna, "Construcción de covering arrays mixtos usando una generalización del operador de fusión," M.S. thesis, Center Res. Adv. Studies Nat. Polytech. Inst., Inf. Technol. Lab., Ciudad Victoria, Mexico, 2012.



JOSE TORRES-JIMENEZ received the Ph.D. degree from ITESM Cuernavaca, Mexico. He is currently a Teacher and a Researcher with Cinvestav Tamaulipas, Mexico.

He is also an Expert in combinatorial optimization. He has graduated more than ten Ph.D. professionals and more than 50 M.Sc. professionals. He has dedicated more than a decade to build many of the best-known covering arrays (mathematical objects that are used to do software and hardware testing). He has many international collaborations in the USA, Spain, Colombia, France, and Austria. He is a Level III Member of the National System of Researchers, Mexico. He has been an IEEE Senior Member, since 1999.

IDELFONSO IZQUIERDO-MARQUEZ is currently pursuing the Ph.D. degree with Cinvestav Tamaulipas, Mexico. He has published 12 journal papers, mostly on covering arrays, and three conference papers. He has carried out three research stays in Mexico and Spain. His research interest includes the construction and classification of the combinatorial objects called covering arrays.

HIMER AVILA-GEORGE received the Ph.D. degree in computer science from the Politeìcnica de Valeìncia, Spain. He was a Researcher and the Academic Manager in various universities in Mexico. In consideration of his professional career, since 2014, he has been a Distinguished National Researcher (Level I) with the National System of Researchers, Mexico. He is currently a Full Professor with the University of Guadalajara, Mexico. His current research interests include combinatorial optimization, software engineering, and machine learning.