Fachbereich II
Mathematik - Physik - Chemie

Berliner Hochschule für Technik

01/2025

Ulrike Grömping

Generalizing a Sherwood (2008) construction for mixed level covering arrays

Verallgemeinerung einer Konstruktion von Sherwood (2008) für gemischtstufige abdeckende Felder (englischsprachig)

Reports in Mathematics, Physics and Chemistry
Berichte aus der Mathematik, Physik und Chemie

ISSN (print): 2190-3913

Reports in Mathematics, Physics and Chemistry

Berichte aus der Mathematik, Physik und Chemie

The reports are freely available via the Internet:
https://www1.bht-berlin.de/FB_II/reports/welcome.htm

01/2025, July 2025

Generalizing a Sherwood (2008) construction for mixed level covering arrays
Verallgemeinerung einer Konstruktion von Sherwood (2008) für gemischtstufige abdeckende Felder (englischsprachig)

Editorial notice / Impressum

Published by / Herausgeber:
Fachbereich II
Berliner Hochschule für Technik
Luxemburger Str. 10
D-13353 Berlin
Internet: https://www.bht-berlin.de/ii

Responsibility for the content rests with the author(s) of the reports.
Die inhaltliche Verantwortung liegt bei den Autor/inn/en der Berichte.

ISSN (print): 2190-3913

Generalizing a Sherwood (2008) construction for mixed level covering arrays

Ulrike Grömping

Berliner Hochschule für Technik, Germany

Abstract

Covering arrays are used for covering interactions between experimental variables in testing complex systems, and the "strength" of the covering array determines the degree of the interactions that are completely covered. Mixed level covering arrays permit different numbers of levels for different experimental variables. Sherwood (2008) proposed methods for constructing strength 2 mixed level covering arrays. This note generalizes his first construction, in support of the R package **CAs**.

1 Introduction

Covering arrays (CAs) of strength t are $N \times k$ arrays for experimental designs in k experimental factors, such that for any set of columns $\{c_1, \ldots, c_t\} \subseteq \{1, \ldots, k\}$, all $v_{c_1} \cdot \ldots \cdot v_{c_t}$ t-tuples of levels are covered at least once by the N rows, where v_c denotes the number of levels of column c. It is a convention for CAs to denote by a "*" positions in the CA that can take an arbitrary value from the level set of their column without deteriorating the coverage of t-tuples; these are called flexible values. In the following, the focus is solely on $t = 2$ constructions, i.e., on covering all pairs of values.

Sherwood (2008) proposed constructions for strength 2 mixed level covering arrays (MCAs), i.e., constructions for obtaining covering arrays for arbitrary patterns of numbers of levels for the factors. Such constructions are less common than constructions for uniform covering arrays (UCAs), for which all factors have the same number of levels. Some sources use the term "CAs" to denote only UCAs; this note uses it as the generic term for both MCA and UCA.

The first Sherwood (2008) construction consists in expanding some columns of a uniform strength 2 orthogonal array (OA) into more columns at fewer levels than before, resulting in a strength 2 MCA. His first two theorems explicitly request an OA as the starting point and return designs whose number of rows equals that of the OA. For a resulting run size larger than the run size of the OA, Sherwood (2008) works with increasing the run size of the OA, which requires using so-called "Latin Ordered Designs" (LODs). The construction in this paper instead works by starting from a suitable CA which already has more rows than the OA; it therefore can use ordered designs (ODs) that do not have the "Latin" property.

The next section provides terminology and notation for the construction elements. In the third section, two algorithms for the construction are stated and exemplified; these are implemented in function `MCA2` of Grömping (2025)'s R package **CAs**, which is currently under development. The purpose of this note is to provide an unmistakable justification of this construction. Section 4 isolates and proves the key requisite: for a strength 2 CA, any column in v levels with at least $rv(v-1)$ flexible values can be expanded into s^r columns, where $s > 1$ is the maximum number of columns of an ordered design in v levels and $v(v-1)$ runs. The final section discusses potential improvements of the approach.

2 Terminology and notation

CAs were already introduced in the introduction. An OA of strength 2 has each pair of level combinations *the same number of times*. In the context of CAs, one is most interested in OAs for which each level combination occurs *exactly once*; those are called "OAs of index unity". A CA of strength 2 has each pair of level combinations *at least once*. Thus, every OA is a CA, but the reverse is not true.

Table 1: OA$(9, 2, 4, 3)$ from the Bose construction (B_3), with its structure highlighted by the formatting

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

Table 2: Dimensions of OD(v) for v=2,...,10

v	rows	columns
2	2	2
3	6	3
4	12	4
5	20	5
6	30	3
7	42	7
8	56	8
9	72	9
10	90	3

A strength 2 MCA with k_j columns at v_j levels, $j = 1, \ldots, \ell$, $\ell > 1$, $v_1 > \cdots > v_\ell \geq 2$ is denoted as $\mathrm{CA}_2(N, v_1^{k_1} \ldots v_\ell^{k_\ell})$. This naturally generalizes the UCA notation $\mathrm{CA}_2(N, v^k)$. For OAs, the analogous notation can be used with "OA" instead of "CA".

According to Bierbrauer (2007), an OD of strength 2 is an array whose rows have distinct elements, with the guarantee that each ordered pair of values occurs the same number of times. A strength 2 OD of index unity in v levels thus must have $v(v-1)$ rows. According to Sherwood (2008), LODs are a special case of ODs, for which the runs can be grouped into so-called Latin rectangles of v rows each, for each of which each column holds all levels.

For q a prime or prime power, there is an $\mathrm{OA}_2(q^2, q^{q+1})$ that is constructed according to Theorem 1 of Bose (1938). In this note, a specific variant of this OA is denoted as B_q, as it is the basis of the (L)OD for q levels: the first q rows are constant in the first q columns with zeroes in the last column. Table 1 shows an instance of B_3. B_q can be obtained with the R command `SCA_Bose(q)` of R package **CAs** (which uses function `createBose` of R package **lhs**, Carnell (2024)). Its bottom left corner is the LOD of index unity for 3 levels with the maximum possible number of columns: it is easy to verify that all ordered pairs of levels (01 10 02 20 12 21) occur exactly once for each of the three column pairs, and it is of course not possible to have more than three 3-level columns with rows having distinct levels; the array is an **L**OD, because its runs consist of two groups of three rows (latin rectangles) that each hold each level in each column. In general, the bottom left $q \cdot (q-1) \times q$ corner of B_q is an LOD. The "Latin" requirement is not needed in the proofs of this note, because the construction does not increase the number of rows of the ingoing array. For v that are not a prime power, an OD has at least two columns, obtained as the cross product of the two sets of levels and deleting the rows with both levels the same. In some cases, more columns can be created from OAs for which v rows can be made constant, e.g., for $v = 6$ there is a three column OD, as there is an $\mathrm{OA}_2(36, 6^3)$ whose first 6 rows can be made constant. For $v = 10$, there is a four-column OA, which does not permit 10 constant rows; reducing it to three columns permits an OD by making 10 rows constant and removing them. This paper denotes an OD with as many as possible columns in $v(v-1)$ runs as OD(v) (and it does not distinguish between different such ODs). Table 2 shows the achievable dimensions of ODs for $v = 2, \ldots, 10$.

3 An algorithm for the construction of strength 2 mixed-level CAs

The goal is to create a $\mathrm{CA}_2(N, v_1^{k_1} \ldots v_\ell^{k_\ell})$, $v_1 > \cdots > v_\ell$, from a $\mathrm{CA}_2(N', v_1^{k'})$, $N' \geq N, k' \leq k = k_1 + \cdots + k_\ell$. A trivial construction with $k = k'$ consists in reducing the numbers of levels of the $k - k_1$ columns for which fewer than v_1 levels are needed. v_j levels can be achieved by making $k_1 - k_j$ levels of the uniform CA flexible in the k_j relevant columns. Of course, this is somewhat wasteful, and one can attempt to postprocess the result for reducing the number of rows, e.g., by the Nayeri et al. (2013) approach.

Sherwood (2008) proposed a different strategy for the flexible values: exploit them for expanding a single column into several columns. Function `MCA2` of the package **CAs** embraces this approach and proceeds as follows:

Algorithm 3.1. ***Create an MCA***

Input*: a level pattern $v_1^{k_1} \ldots v_\ell^{k_\ell}$ with $v_1 > \cdots > v_\ell$*

Output*: a mixed level array with the requested level pattern*

Initialize

1. *Start from a start array D, which is a suitable available $CA_2(N', v_1^{k'})$ with well-chosen k' (see Algorithm 3.2).*
2. *Create an $N' \times k$ matrix M of missing values that will hold the final MCA.*
3. *For c in 1 to k_1, populate column c of M with column c of D (columns without reduction in the number of levels).*

Repeat

4. *Increase c by one. Take column c of D, and reduce its number of levels to the largest v_j for which M still needs columns, by making its most frequent $v_1 - v_j$ level(s) flexible (rearrange levels so that these are the largest ones).*
5. *Let k'_j denote the number of v_j-level columns still needed for M ($k'_j = k_j$ at the first round for v_j levels). If $k'_j > 1$, determine the number f_c of flexible values in column c, and identify the largest integer r such that $f_c \geq r v_j(1 - v_j)$.*
   a. *If $r = 0$, each v_j-level column needs a new column of D. Populate M the first free column of M with column c of D, and move to step 4.*
   b. *If $r \geq 1$, create up to s_2^r v_2-level columns from column c in the following way:*
      - *Create an $v_j(v_j - 1) \times s_j$ $OD(v_j)$.*
      - *Horizontally concatenate s_j identical copies of column c into the $N' \times s_j$ matrix E, which initially has constant rows.*
      - *Select $v_j(1 - v_j)$ rows with flexible values, and populate them with $OD(v_j)$. Now E is a $CA_2(N', v_j^{s_j})$ (see proof in the next section).*
      - *If $r > 1$, each column of E still has at least $v_j(v_j - 1)$ flexible values, and the process can be repeated with each column.*
      - *… which eventually yields s_j^r v_j-level columns in E*
   c. *If $k'_j > s_j^r$, place all columns of E in M, reduce k'_j by s_j^r, and go to step 4.*
   d. *If $k'_j \leq s_j^r$, take k'_j columns from E and populate the missing v_j-level columns in M with these columns. If $j < \ell$, go to step 4. If $j = \ell$, return M.*

Algorithm 3.1 returns a $\mathrm{CA}_2(N', v_1^{k_1} \ldots v_\ell^{k_\ell})$. It will often be possible to reduce N' by the Nayeri et al. (2013) method.

Algorithm 3.2. ***Determine ingredients for Algorithm 3.1***

Input*: a level pattern $v_1^{k_1} \ldots v_\ell^{k_\ell}$ with $v_1 > \cdots > v_\ell$*

Output*: k', N' and D for Algorithm 3.1*

1. *Let $k' = k_1 + \ell - 1$.*
2. *Determine the smallest feasible N' for a $CA(N', v_1^{k'})$.*
3. *Given N', determine the minimum achievable f_j (number of flexible values in a column with v_j levels) for $j = 2, \ldots, \ell$, as $f_j = \lceil N'(v_1 - v_j)/v_1 \rceil$.*

Table 3: A $\mathrm{CA}_2(22, 4^3 3^8 2^6 0)$ created with Algorithm 3.1.
D was used for constructing M, as described in Example 3.1.
`*` denotes a flexible value.
M was created with the command
`M <- MCA2(c(rep(4,3), rep(3,8), rep(2,30)), outerRetry = 0)`
using version 0.10 of the R package **CAs**

	D	M
1	0000000	00000000000000000000000000000000000000000000
2	0111212	01111122211000000000000000011111111111111
3	2011121	20111111122111111111111111111111111111111
4	1201112	12011111111111111111111111100000000000000
5	2120111	21200011111111111111111111111111111111111
6	1212011	12122200011111111111111111111111111111111
7	1121201	11211122001111111111111111111111111111111
8	1112120	11122211122000000000000000000000000000000
9	0222323	02222201222111111110000000011111111000000
10	3022232	30222222201000000001111111100000000111111
11	2302223	23022222222000011110000111100001111000011
12	3230222	32300022222111100001111000011110000111100
13	2323022	23201200022110011001100110011001100110011
14	2232302	22322120000011001100110011001100110011001100
15	2223230	22212022212000000000000000000000000000000000
16	0333131	03320111120111111111111111111111111111111111
17	1033313	10302120111010101010101010101010101010101010
18	3103331	31010202102111111111111111111111111111111111
19	1310333	13100010210101010101010101010101010101010010
20	3131033	31311100021******************************
21	3313103	33121011100******************************
22	3331310	33311210110000000000000000000000000000000000

4. *Calculate the minimium numbers of feasible v_j-level columns ($s_j^{r_j}$) with r_j the maximum r such that $f_j \geq r v_j(v_j - 1)$, and s_j the number of columns of $OD(v_j)$, as before.*
5. *Let $k'_j = \lceil k_j/(s_j^{r_j}) \rceil$ the number of columns of D needed for obtaining k_j v_j-level columns for M. This yields $k' = k'_1 + \cdots + k'_\ell$ as the new k'.*
6. *Repeat steps 2 to 5 once.*
7. *Determine whether N' can be reduced. If so, check whether the reduced N' still permits the construction (are there still enough flexible values for each number of levels?). If yes, keep the reduced N', otherwise keep the larger N'.*
8. *Return k' and D (as the $CA_2(N', v^{k'})$ with the N' of Step 7).*

Step 1 in the algorithm for k' corresponds to the expectation that each v_1-level factor needs a column of D, and the other variants can be constructed from a single dedicated column for each v_j. If that was overoptimistic, step 4 figures out how many columns would be needed for the small N' for the initial guess of k'. Subsequently, N' is updated to the run size of a CA for this number of columns, and the numbers of flexible values f_j are updated to reflect this new N'. Based on these flexible values, it is checked whether k' may be reduced again. Step 7 checks whether this in turn leads to a new and smaller N', and whether or not this smaller N' still permits the construction (might go wrong because of a reduced number of flexible values), and returns either the reduced N' or the larger N', depending on the outcome. Step 8 returns k' and the suitable N'-run CA D.

Example 3.1. This example describes the construction of a $\mathrm{CA}_2(22, 4^3 3^8 2^{30})$ using the algorithms stated above. The resulting array can be found in Table 3.

Algorithm 3.2 determines $k' = 7$:

1. The initial guess is $k' = 3 + 3 - 1 = 5$.
2. It can be accommodated in a 16×5 matrix D, i.e., $N' = 16$, e.g., B_5.

3. This permits $16 \cdot (4-3)/4 = 4$ flexible values for 3-level columns and $16 \cdot (4-2)/4 = 8$ flexible values for 2-level columns.
4. $r_2 = 0$, as OD(3) has $3 \cdot 2 = 6$ rows. Thus, a single column of D can generate only $s_2^0 = 1$ 3-level column. $r_3 = 4$ and $s_3 = 2$, which implies that a single column of D can generate $2^4 = 16$ 2-level columns.
5. With $N' = 16$ runs for D,
$$k' = \underbrace{\left\lceil \frac{3}{4^0} \right\rceil}_{v_1=4} + \underbrace{\left\lceil \frac{8}{3^0} \right\rceil}_{v_2=3} + \underbrace{\left\lceil \frac{30}{2^4} \right\rceil}_{v_3=2} = 3 + 8 + 2 = 13.$$
6. Repeat 2.: 13 4-level columns can be accommodated in $N' = 25$ runs (using the construction of Colbourn et al. (2010)).
Repeat 3.: There are at least 7 flexible values for 3-level columns and at least 13 flexible values for 2-level columns.
Repeat 4.: This implies $r_2 = 1$, i.e., and each column of D can generate $s_2^{r_2} = 3^1 = 3$ 3-level columns for M. Furthermore, $r_3 = 6$, so that a single column of D can generate up to $s_3^{r_3} = 2^6 = 64$ 2-level columns for M. Repeat 5.: Hence, with $N' = 25$,
$$k' = \underbrace{\left\lceil \frac{3}{4^0} \right\rceil}_{v_1=4} + \underbrace{\left\lceil \frac{8}{3^1} \right\rceil}_{v_2=3} + \underbrace{\left\lceil \frac{30}{2^6} \right\rceil}_{v_3=2} = 3 + 3 + 1 = 7.$$
7. This number of columns can be accommodated in $N' = 22$ runs (the best known array according to Colbourn (n.d.) has 21 runs with construction "simulated annealing (Cohen)"; this array is at present not available in R package **CAs**). With the $N' = 22$ runs, it is still possible to produce the target CA with a 22×7 D (6 flexible values are possible for 3-level columns by making the most frequent level missing, and $2^6 = 64$ flexible values are possible for 2-level columns, by making the two most frequent levels missing).
8. $k' = 7$, and D is obtained from the Meagher and Stevens (2005) construction (see Table 3).

Algorithm 3.1 creates the MCA M of Table 3 by successively processing the columns of D: The first $k_1 = 3$ columns of D become the first k_1 columns of M (the 4-level columns), the next $k_2 = 8$ columns of M are based on columns 4 to 6 of D (the 3-level columns), and the 30 2-level columns of M are created from column 7 of D.

- Consider columns 4 to 6 of M, which are based on column 4 of D: Where column 4 of D is not 3, columns 4 to 6 of M all have the level of column 4 of D. For the 6&nbps;rows for which column 4 of D has the level 3, columns &nbps;4 to 6 of M were initially made flexible, and these the 6×3 flexible submatrix of columns 4 to 6 of M was then replaced with the 6×3 OD(3), whose rows are 012, 120, 201, 021, 210, 102. Columns 7 to 9 of M are treated analogously, based on column 5 of D, and columns 10 and 11 of M are obtained as the first two of three columns resulting from the analogous processing of column 6 of D.
- For the 2-level columns, recursive processing is needed for obtaining 30 columns of M from a single column of D, based on the 2×2 OD(2), whose rows are 01, 10.
  - Initially, levels 2 and 3 of column 7 of D are made flexible. The column is then duplicated, and the first 2×2 flexible submatrix, consisting of rows 2 and 4, is replaced with OD(2).
  - The two resulting columns are then treated again, starting with the second one: each is duplicated, and the now-first 2×2 flexible submatrix in rows 9 and 10 is replaced with OD(2) in reverse column order. After this step, there are four 2-level columns, all of which are now again duplicated, populating the now-first 2×2 flexible submatrices in rows 11 and 12 with OD(2). The eight resulting columns are now treated, populating the now-first flexible submatrices in rows 13 and 14 with OD(2) in reverse column order. The resulting 16 columns have to be treated once more, populating the now-first 2×2 flexible submatrices in rows 17 and 19 with OD(2). Of the thus-created 32 columns, the last two are omitted because they are not needed.
    Note that the alternation of OD(2) with reverse OD(2) is caused by the processing order of the array columns from right to left and by moving the new columns to the very right of the matrix E in recursive expansion.
    Also note that the rows for which the last column of D has non-flexible values (0,1) are constant throughout all the columns generated from it.

Table 4: The strength 2 covering array of Table 3 after optimization with the Nayeri et al. (2013) algorithm.
`*` denotes a flexible value.
The optimization can be reproduced with the command
`Moptimized <- postopNCK(M, 2, seed=22822, outerRetry=2)`,
as implemented in version 0.10 of R package **CAs**.

	M after optimization
1	20102111212100011011111111001111101000100
2	21210201021001111011001111001001100010011
3	12110020122110100000000011100110001000110
4	02212102010111111010010000000111010110100
5	32220221211110100110010010010010010110110
6	11122000201011111101001111110000001101001
7	13220021110010010000110101010101111101101
8	12011211111001101011101010000100110110101
9	30311100220111000011000110110010110001100
10	31000202102101000010110000110111111 0001*0
11	33112011001010101110111*01111000010011011
12	22302022100101000**1110110011111*11**1011
13	230221022200000**1*10011111*111000*0100*1
14	01121222210000000**010110011***01**00***0
15	103022101100001*11*0100001**1001000111011
16	00001020001*1*0101010***01**1001000*11011
17	102201220020****1*0******010*0***00111011
18	23201210022111**010011*000000*001*110***0
19	313*01120*1*0**1**0*0*1****0**0**01*100*0
20	03320111122**********1*********1******

Example 3.2. Postprocessing the 22×41 matrix M of the previous example with the Nayeri et al. (2013) algorithm, using the default settings of function `postopNCK` of R package **CAs** (version 0.10) quickly reached run size 21, and after further optimization rounds, run size 20; this result was obtained with the randomly assigned seed 22822, with which it can be reproduced. The optimized 20 run CA is shown in Table 4.

The next section will provide the proof that Algorithm 3.1 actually delivers strength 2. The proof is basically unchanged from Sherwood (2008), as Sherwood's proof of the strength 2 property did not make use of the OA property that he requested for the ingoing array. The OA property was solely needed for being able to state optimality of the result in case of at least 2 columns at the largest number of levels. This paper does not state optimality of the result, but contends with the strength 2 property. In fact, as was seen in the examples, in cases for which the run size does not coincide with a known lower bound, it is worthwhile to attempt improvement of the run size by a post-processing method like that of Nayeri et al. (2013).

4 Proof for the central construction step

The following theorem states the construction for the to-be-expanded column in the last position (w.l.o.g.). It avoids notation for mixed levels. The CA in the theorem can have mixed or fixed levels, and there is no requirement on numbers of levels being in any specific order.

Theorem 4.1. *Let C be an $N \times k$ strength 2 CA whose kth column in v levels has f flexible values, with $f \geq rv(v-1)$. Then, the $N \times (k-1+s^r)$ matrix obtained as the first $k-1$ columns of C, combined with the $N \times s^r$ matrix E constructed like in step 5.b of Algorithm 3.1 is a strength 2 CA.*

Proof. Let C_{-k} denote the first $k-1$ columns of C.

- Obviously, C_{-k} is a strength 2 CA.

- All columns of the matrix E coincide with the kth column of C in all non-flexible positions of that column. Therefore, all pairs of levels are covered for column pairs with one column from C_{-k} and the other from E.

It remains to be shown that pairs of columns from E have all their level pairs covered. This can be seen as follows:

- Pairs with identical levels are covered by the constant rows that hold the previously non-flexible levels.
- For $r = 1$, $v(v-1)$ previously flexible rows of E hold an OD(v) that covers all pairs with distinct levels. This completes the proof for $r = 1$.
- For $r > 1$, the above reasoning can be applied repeatedly to all interim steps, which completes the proof also for this case.

□

5 Discussion

Sherwood (2008) proposed a highly flexible construction for mixed level CAs. His first two theorems were generalized to using arbitrary strength 2 CAs as ingoing matrices. This note details the implementation for this generalized version and proves that it yields a strength 2 CA, as intended; the implementation of the function `MCA2` in the R package **CAs** is thus mathematically sound.

The implementation does not attempt the use of a single column of the matrix D for different numbers of levels in the outcome matrix M: with $r_j > 1$, it would be possible to use a column of D for, e.g., $s_j^{r_j-1}$ columns in v_j levels and subsequently use remaining flexible values for creating columns in $v_{j+1} < v_j$ levels. This should be doable (if tedious) and might sometimes help to achieve a smaller k' in Algorithm 3.2, which leads to fewer rows in D. Without post-optimization of run size, it is important to use the smallest possible D for row expansion. With post-optimization, this is not quite as important, as run size post-optimization likely offsets at least part of the size difference.

References

Bierbrauer, J. (2007), "Ordered Designs, Perpendicular Arrays, and Permutation Sets," *Handbook of combinatorial designs*, Discrete mathematics and its applications, Boca Raton, FL: Chapman & Hall/Taylor & Francis.

Bose, R. C. (1938), "On the application of the properties of Galois fields to the problem of construction of hyper-Graeco-Latin squares," *Sankhyã*, 3, 323–338.

Carnell, R. (2024), "lhs: Latin Hypercube Samples." https://doi.org/10.32614/CRAN.package.lhs.

Colbourn, C. J. (n.d.). "Covering array tables: 2<=v<=25, 2 <=t<=6, t<=k<=10000, 2005–23," Formerly available at https://www.public.asu .edu /~ccolbou/src /tabby; November 2024 status of tables available at https://github.com/ugroempi/CAs/blob/main/ColbournTables.md.

Colbourn, C., Kéri, G., Rivas Soriano, P. P., and Schlage-Puchta, J.-C. (2010), "Covering and radius-covering arrays: constructions and classification," *Discrete Appl. Math*, 158, 1158–1180. https://doi.org/10.1016/j.dam.2010.03.008.

Grömping, U. (2025), CAs: Creating Covering Arrays, R package development version 0.10. Current version available at https://github.com/ugroempi/CAs.

Meagher, K., and Stevens, B. (2005), "Group construction of covering arrays," *Journal of Combinatorial Designs*, 13, 70–77. https://doi.org/10.1002/jcd.20035.

Nayeri, P., Colbourn, C. J., and Konjevod, G. (2013), "Randomized post-optimization of covering arrays," *European Journal of Combinatorics*, 34, 91–103. https://doi.org/10.1016/j.ejc.2012.07.017.

Sherwood, G. B. (2008), "Optimal and near-optimal mixed covering arrays by column expansion," *Discrete Mathematics*, Elsevier BV, 308, 6022–6035. https://doi.org/10.1016/j.disc.2007.11.021.