Fachbereich II
Mathematik - Physik - Chemie

Berliner Hochschule für Technik

01/2026

Ulrike Grömping

Implementing uniform covering arrays based on the Colbourn tables

Implementierung von Covering Arrays auf Basis der Colbourn tables (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

01/2026, January 2026

Implementing uniform covering arrays based on the Colbourn tables
Implementierung von Covering Arrays auf Basis der Colbourn tables
(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

Implementing uniform covering arrays based on the Colbourn tables

Ulrike Grömping

2026-01-02

Contents

1 Preface

This technical report documents the status at the end of a sabbatical semester during which its author collected mathematics-based (as opposed to purely search-based) covering array (CA) constructions and implemented many of them in the R package **CAs**. It serves two purposes:

- document the state of knowledge reached, for the author herself and other mathematically-minded people with an interest in the implementation of CAs, in support of future work on and with the package **CAs**; the report is not a manual for that package, but is meant to provide an overview of constructions, including the information which constructions are implemented in which functions at the time of writing.
- support the future re-implementation of the so-called Colbourn tables (Colbourn (n.d.)), which were maintained by Charles Colbourn for many years but have now been discontinued due to their author's retirement in conjunction with a re-design of his university's website which disappeared the original tables. (Versions of the tables are still available on the internet archive at https://web.archive.org/web/20250211193915/https://www.public.asu.edu/~ccolbou/src/tabby/catable.html, and, with slightly more up-do-date tables, at the author's GitHub repository at https://github.com/ugroempi/CAs/blob/main/ColbournTables.md.) Deciphering the meaning of the tables' source entries is a non-trivial task, to which this report contributes.

As the author's time budget for working on this project was dramatically reduced after the sabbatical semester, this interim work-in-progress state of work is documented, instead of postponing the reporting to a more matured state.

2 Introduction

$\mathrm{CA}(N, t, k, v)$ denotes a uniform covering array (CA) in N runs, with strength t, for k variables (i.e., with k columns) in v levels each, which is an $N \times k$ matrix with entries $0, \ldots, v-1$, for which every t-tuple of columns contains each of the v^t level combinations from $\{0, \ldots, v-1\}^t$ at least once. The CA is called "uniform" because all its columns have the same number of levels. The covering array number $\mathrm{CAN}(t, k, v)$ is the smallest N for which a $\mathrm{CA}(N, t, k, v)$ exists, and is systematically known for $t = 2$ and $v = 2$, or where a CA with $N = v^t$ is available (and by proving coincidence of upper and lower bounds for selected other situations). CAs have also been researched under different terminology: surjective codes, transversal covers, qualitatively independent partitions, ...; see e.g. Sloane (1993) or Moura, Stardom, Stevens, and Williams (2003)).

CAs can have so-called *flexible* or *don't care* or *wildcard* values, denoted as stars (or in R as `NA`-values), indicating that the respective position can be filled with an arbitrary level without reducing the strength of the CA. This is different from a so-called "unsatisfiable constraint" or "shielded parameter", which means that a particular column does not have any valid level for certain combinations of some other columns, e.g., because a certain feature of a product does not exist in base versions of the product so that all columns referring to that feature are necessarily at that "unsatisfiable" level, or because the "-h" argument to a function incapacitates all other arguments that might be specified with it (example from Zhang, Zhang, and Ma (2014)). JMP Statistical Discovery LLC (2024) provides functionality for the latter situation; this report does not cover it. Neither does it cover the more common situation of *constraints* on level combinations, i.e., certain level combinations being prohibited by design.

Colbourn (n.d.) is a compiled list, called "Covering Array tables", which holds the smallest N for a known construction for all combinations of strengths $t = 2, \ldots, 6$, alphabet sizes $v = 2, \ldots, 25$ and numbers of columns k up to 10000 (20000 for strength 2). This source is henceforth called "the Colbourn tables". It used to be an authoritative source for the current state of smallest achievable uniform CAs, before it

was discontinued in 2025, after Charles' Colbourn's retirement and a website overhaul at Arizona State University. Diverse states of the Colbourn tables are available from the web archive, and a November 2024 status is kept at https://github.com/ugroempi/CAs/blob/main/ColbournTables.md; to the best of the author's knowledge, these hold the latest available status, including, e.g., the strength&nbps;6 CAs from Wagner, Kampel, and Simos (2021) (source entry "SIPO (Wagner-Kampel-Simos)"). In this report and in the R package **CAs**, the stated sizes of the Colbourn tables are called eCANs, with "e" for empirical. While the table entries refer to arrays with known constructions, the constructions themselves are not provided but only alluded to, in a column named "Source", whose content needs to be deciphered in order to work out how to construct the actual arrays for the stated sizes. The disappearance of the Colbourn tables may be seen as a chance of re-implementation of a new version of "eCAN tables" that are somewhat more actionable (see below). Besides its accompanying the R package **CAs**, this report aims at supporting efforts at re-implementation of the eCAN tables, which will hopefully be undertaken by members of the combinatorial testing community.

The Colbourn tables in their current form lack actionability for obtaining or creating actual arrays. This document structures their source information and provides explicit references in support of implementing as many of the smallest CAs as possible in the R package **CAs**. The goal is to accompany the R package with a guide for users who are interested in background information. The approach is pragmatic, with emphasis on mathematical constructions and de-emphasizing constructions that are resource-intensive in terms of run time. Many of the published smallest arrays were obtained by numerical searches. In some cases, the search results can be summarized by individual vectors or even a simple recipe (e.g., cyclotomy constructions according to Colbourn (2010) or cover starter constructions according to Meagher and Stevens (2005)). For such cases, the actual optima are provided in the package. In other cases, searches improve the run sizes or the number of columns for arrays that were constructed with mathematical approaches; whether or not such arrays are made available depends on their importance (e.g., as ingredients in further optimal designs from recursive methods), availability and size.

Figure 1 shows that a few source entries are very frequent, while many others occur a few times only. The review papers Colbourn (2004), Hartman (2005), Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010) and Torres-Jimenez, Izquierdo-Marquez, and Avila-George (2019) (among others) are helpful for locating actual references for arrays or array constructions; for strength 2 CAs, Colbourn (2008) gives an overview of the different constructions at the time, some of which were meanwhile slightly improved by numeric optimization. There is always only one source entry in the Colbourn tables (and also in Colbourn (2008)), even though there may be several constructions that yield the same array size. This paper is particularly focused on methods with a strong algebraic footing that rely little or not at all on search and optimization methods.

In an initial attempt at getting an overview of source entries in Colbourn (n.d.), cluster analysis of the source entries was used in order to group related entries, as there are many entries that differ in a small detail like a numeric specification only. The details and results of this analysis are reported in Appendices A and B.

The next section provides notation and basic ingredients, followed by a section on repositories and sections on mathematical methods (with very few excursions to search-based techniques) that are relevant for at least some competitive CAs, by either occurring themselves as source entries in the Colbourn tables, yielding arrays as good as such source entries, or at least yielding arrays better than other implemented techniques in some cases. Appendices A and B report on a cluster analysis of the source entries of the Colbourn tables (Colbourn (n.d.)), which was used as one of the starting points of the implementation project. The long-standing research area uses an enormous body of terminology, and it is outside of the scope of this report to explain every concept. Some important concepts and ingredients are explained in the next section, others with the constructions for which they are used. Readers are encouraged to draw on fundamental literature such as Hedayat, Sloane, and Stufken (1999) or Colbourn and Dinitz (2007) for background information.

3 Notation and basic ingredients

Vectors that consist of single digit symbols are abbreviated as the string of those symbols, e.g., 012120201. Note that the scope of this notation can be broadened by expanding the digits to more than 10 possibilities via using letters for larger integers (e.g., a for 10, etc.). Column and row vectors are not distinguished by this notation; context will usually be sufficient. Furthermore, matrices with all elements 0 or one can be

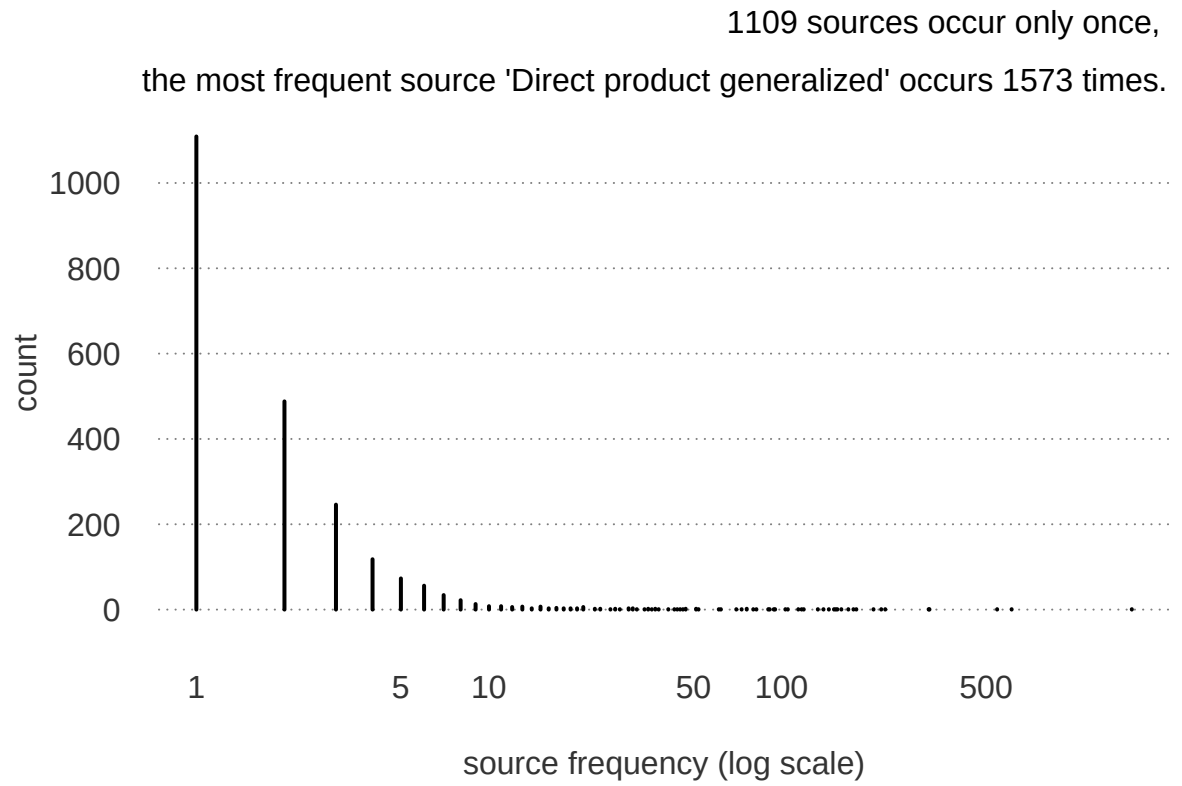

Figure 1: Frequency distribution of source entries

denoted as $\mathbf{0}_{m\times n}$ or $\mathbf{1}_{m\times n}$, respectively, length n column vectors of all zeroes or all ones as $\mathbf{0}_n$ or $\mathbf{1}_n$ and $\mathbf{0}_{1\times n} = \mathbf{0}_n^\top$ or $\mathbf{1}_{1\times n} = \mathbf{1}_n^\top$ for row vectors.

The subsequent ingredient sections describe those ingredients that are created in the package. For other ingredients, readers are referred to Colbourn and Dinitz (2007).

The Kronecker product, denoted by $\otimes$, is a convenient way of mathematically describing common constructions:

- $\mathbf{1}_k^\top \otimes A$ horizontally concatenates k identical copies of an $N \times \ell$ matrix A,
- $A \otimes \mathbf{1}_k^\top$ consists of ℓ groups of k identical columns instead.

Vertical concatenation can be analogously denoted using column vectors $\mathbf{1}_k$, and block repetition can be represented using a matrix $\mathbf{1}_{s\times r}$. In this report, the symbol $\otimes$ always refers to the Kronecker product with matrices of "1" entries only.

3.1 $\mathbb{Z}_m$ and GF(q)

$\mathbb{Z}_m$ denotes the ring of integers modulo m (where a ring is a set with addition and multiplication, and neutral elements for both, but not necessarily an inverse for multiplication). Where m is prime, it is also a field, as there is also a multiplicative inverse. Table 1 shows the addition and the multiplication table for $\mathbb{Z}_8$. As 8 is a prime power but not a prime, some elements do not have a multiplicative inverse, in this case all the even numbers: the elements $\{0, 2, 4, 6\}$ are a subgroup, and multiplication of one of its elements with any element of $\mathbb{Z}_8$ does not leave that subgroup and never reaches the neutral element 1.

GF(q) is a finite field with q elements, where q is a prime or prime power. In many accounts, it is defined via polynomials; here, it will be considered via its addition and multiplication tables only, and its elements will be denoted as $0, \ldots, q-1$. For prime q, $\mathbb{Z}_q$ is a GF(q). For prime powers, things are more complicated. In general, there may be different GF(q), but they are all structurally isomorphic. However, for some constructions it may matter which isomorphic representative is used. For example, GF(8) can be constructed from two different polynomials, $x^3 + x + 1$ or $x^3 + x^2 + 1$. These yield the same addition table (Table 2) and different multiplication tables (see Table&nbps;3). The multiplication tables coincide in the rows and columns for 0 and 1, as well as in the top left 4×4 corner of the tables (this is in line

Table 1: Addition and multiplication table of $\mathbb{Z}_8$

	Addition								Multiplication							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7	0	0	0	0	0	0	0	0
1	1	2	3	4	5	6	7	0	0	1	2	3	4	5	6	7
2	2	3	4	5	6	7	0	1	0	2	4	6	0	2	4	6
3	3	4	5	6	7	0	1	2	0	3	6	1	4	7	2	5
4	4	5	6	7	0	1	2	3	0	4	0	4	0	4	0	4
5	5	6	7	0	1	2	3	4	0	5	2	7	4	1	6	3
6	6	7	0	1	2	3	4	5	0	6	4	2	0	6	4	2
7	7	0	1	2	3	4	5	6	0	7	6	5	4	3	2	1

Table 2: Addition table for GF(8) (same for both isomorphic instances)

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

with the fact that the 4×4 multiplication table of GF(4) is unique). All other entries differ between the two variants, which may seem counter-intuitive, as the addition tables are the same. However, with an Galois field for non-prime q, one has to keep in mind that the numbers $0, \ldots, q-1$ serve as symbols only but cannot be interpreted at their numeric value: For example, under addition, $6+6=0$, so that $2 \cdot 6 \neq 6+6$ in both fields, as it is $2 \cdot 6 = 1$ in the left-hand side multiplication table and $2 \cdot 6 = 7$ in the right-hand side multiplication table.

3.2 Prominent ingredient OAs

An important ingredient for constructing CAs is the orthogonal array according to a construction via $q-1$ Latin squares as stated in Theorem 1 of Bose (1938), based on a finite field with q elements for a prime or prime power q; this yields an $\mathrm{OA}(q^2, 2, q+1, q)$, with the notation analogous to that for CAs and q a prime or prime power. An orthogonal array (OA) of strength 2 has each pair of level combinations *the same number of times*, which is (of course) a more stringent requirement than that for a CA; thus each OA is a CA, but not vice versa. In this text, a specific variant of this $\mathrm{OA}(q^2, 2, q+1, q)$ is denoted as B_q, as it is the basis of various constructions for covering arrays: the first q rows are constant in the first q

Table 3: Multiplication tables for GF(8) for two different but isomorphic instances

	from R package lhs								the other							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
2	0	2	4	6	5	7	1	3	0	2	4	6	3	1	7	5
3	0	3	6	5	1	2	7	4	0	3	6	5	7	4	1	2
4	0	4	5	1	7	3	2	6	0	4	3	7	6	2	5	1
5	0	5	7	2	3	6	4	1	0	5	1	4	2	7	3	6
6	0	6	1	7	2	4	3	5	0	6	7	1	5	3	2	4
7	0	7	3	4	6	1	5	2	0	7	5	2	1	6	4	3

Table 4: 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 5: The two 3-by-3 orthogonal latin squares behind B_3, with row and column labels

	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

	0	1	2
0	0	1	2
1	2	0	1
2	1	2	0

columns with zeroes in the last column. Table 4 shows an instance of B_3, and Table 5 shows the two latin squares behind this OA (row numbers are the first, column numbers the last column of the OA, cell entries make up the second and third columns, respectively). 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)).

A further important ingredient is another orthogonal array of index unity from the construction of Bush (1952), which, like B_q, has $q + 1$ columns in q levels each, but has q^t runs for strength t. This array is implemented in function `SCA_Busht` and has an SCA structure like B_q. Like B_q, it can be used in constructions that require one or more ingredient CAs. An example is shown in Table 6.

For prime powers q, there is yet another orthogonal array construction for a uniform strength 2 OA:

$$\text{OA}\left(q^m, 2, \frac{q^m - 1}{q - 1}, q\right).$$

This OA is *saturated*, i.e., the factor main effects use up all degrees of freedom that the run size provides. For $m = 2$, the setting is that of B_q. For general m, it can be constructed in the following way (Rao-Hamming construction, see e.g. Hedayat, Sloane, and Stufken (1999)):

- Obtain a full factorial with the earlier columns changing faster than the later ones; call it F.
- Obtain another full factorial with the earlier columns changing more slowly than the later ones.
- From the latter, remove all rows whose leftmost entry is not 1. This leaves $\frac{q^m-1}{q-1}$ rows; call it C.
- Obtain the $q^m \times \frac{q^m-1}{q-1}$ array as $FC^\top$, where multiplication happens in GF(q).

Table 6: OA$(27, 3, 4, 3)$ from the Bush construction, with structure analogous to B_3

Runs 1 to 9			
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

Runs 10 to 18			
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

Runs 19 to 27			
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

3.3 Ordered designs

In an ordered design (OD, see Colbourn and Dinitz (2007), Chapter VI.38, but in transposed form), each row contains distinct levels only; hence, an OD for v levels can have at most v columns. An OD of strength t has the property that for each t-set of factors, each ordered t-tuple of t distinct levels occurs exactly once (or exactly λ times; this report assumes $\lambda = 1$). Such an OD must thus have $v!/(v-t)!$ runs. The R package **CAs** calculates strength 2 and strength 3 ODs for prime power v via internal functions `OD2` and `OD3` (see below).

If v is a prime or prime power, there is a strength 2 OD in v columns and $v^2 - v$ runs, obtainable as the bottom left block of B_v (`OD2`). For v that is not a prime power, an array that covers all ordered pairs of distinct levels can be obtained from a strength 2 CA in v^2 runs that permits v constant rows by removing the constant rows (implemented for $v = 6$ and $v = 10$, each with $k = 3$). For any v, a trivial $v(v-1) \times 2$ array can be obtained by removing the constant rows from a full factorial in two columns.

For strength 3, internal function `OD3` of package **CAs** implements the construction stated in Cohen, Colbourn, and Ling (2003), which works for all v for which $v-1$ is a prime power and yields $v(v-1)(v-2)$ runs with v columns. For numbers of levels v for which the construction is not applicable, an array for three columns can again be obtained from a full factorial by removing all rows for which any two elements are the same.

3.4 Permutation vector representation of an array

Sherwood, Martirosyan, and Colbourn (2006) introduced permutation vectors; these were later used and modified by various other papers, as they can be used for a very parsimonious representation of arrays, which also makes it easier to search for large arrays. Sherwood, Martirosyan, and Colbourn (2006)'s permutation vectors are now explained using the terminology as it was used in later papers. They are related to generator-based constructions of orthogonal arrays.

Let q a prime or prime power, and GF(q) a Galois field. Furthermore, consider $\mathcal{V}_t = \{0, 1, \ldots, q, \ldots, q^t - 1\}$. Each $i \in \mathcal{V}_t$ can be represented in base q, i.e., as

$$i = \beta_0(i) + \beta_1(i)q + \cdots + \beta_{t-1}(i)q^{t-1}, \tag{1}$$

where the functions $\beta_0, \ldots, \beta_{t-1}$ return the base q digits for i, i.e., a unique tuple of coefficients from GF(q) for representing i; these are used in the above equation for calculations over $\mathbb{Z}$. The matrix of tuples $(\beta_0(i), \ldots, \beta_{t-1}(i))$, $i = 1, \ldots, q^t$, is the $q^t \times t$ full factorial design matrix for t columns in q levels each.

The tuples are used for constructing an affine combination with the help of so-called "permutation vectors" $(h_1, \ldots, h_{t-1})$ in the following way:

- Pick a length $t-1$ permutation vector $(h_1, \ldots, h_{t-1})$ of elements of GF(q).
- Calculate an affine combination vector of length q^t by looping over $i = 1, \ldots, q^t$, obtaining the coefficients $\beta_0(i), \beta_1(i), \ldots, \beta_{t-1}(i)$, and calculating the ith vector element as

$$\beta_0(i) + \beta_1(i)h_1 + \cdots + \beta_{t-1}(i)h_{t-1},$$

where calculations are now conducted over the Galois field GF(q). *For later reference, note that this is equivalent to picking a length t tuple $(h_0, h_1, \ldots, h_{t-1})$, fixing h_0 at 1, and using the equation*

$$\beta_0(i)h_0 + \beta_1(i)h_1 + \cdots + \beta_{t-1}(i)h_{t-1}. \tag{2}$$

Such longer permutation vectors $(h_0, \ldots, h_{t-1})$ can reach the entire $\mathcal{V}_t$; with h_0 fixed at 1, they can only reach a shifted $t-1$-dimensional sub space, with shift vector $(\beta_0(0), \ldots, \beta_0(v^t - 1))$.

In the analogy of regular OAs, the coefficients h_j take on the role of column generators. The affine combination can be represented by $\overrightarrow{h} = \overrightarrow{(h_1, \ldots, h_{t-1})}$ (or including h_0) or by the integer number from $\{0, 1, \ldots, q^t - 2\}$ (or $\{0, 1, \ldots, q^t - 2\}$ with an extended tuple) whose base v representation is given by $(h_1, \ldots, h_{t-1})$ (or including h_0). There are at least two risks for confusion:

- ordering of the base q representation digits both for β coefficients and, in some cases, for h coefficients,

Table 7: The digits of i from 0 to 63 in base 4. Row labels: power of q in Equation (1).

	i from 0 to 63
0	0123012301230123012301230123012301230123012301230123012301230123
1	0000111122223333000011112222333300001111222233330000111122223333
2	0000000000000000111111111111111122222222222222223333333333333333

- choice of isomorphic GF(q) for non-prime prime power $q \neq 4$, as isomorphic GF(q) are not necessarily equivalent for these constructions (see, e.g., Section 6.6).

3.4.1 Reduced and extended permutation vectors

Sherwood, Martirosyan, and Colbourn (2006) also proposed a *reduced permutation vector*, that uses the length $t-1$ tuple $(h_1, \ldots, h_{t-1})$ differently and only uses the digits $\beta_j(i)$ for powers j up to $t-2$:

$$\beta_0(i)h_1 + \beta_1(i)h_2 + \cdots + \beta_{t-2}(i)h_{t-1}.$$

This means that the resulting linear combinations are all in a $t-1$-dimensional sup-space of $\mathcal{V}_t$.

Both Colbourn and Lanus (2018a) and Torres-Jimenez and Izquierdo-Marquez (2018) extended the h-tuples to t elements and used them as indicated in Equation 2 with one more summand.

3.4.2 Example: Permutation vector representations for the Bush construction

Sherwood, Martirosyan, and Colbourn (2006) gave the following example: The columns of the arrays of the Bush construction of Section 3.2 can be obtained by using the q permutation vectors $(h^1, \ldots, h^{t-1})$, $h \in$ GF(q) for creating the first q columns and the length $t-1$ reduced permutation vector $(0, \ldots, 0, 1)$ for the last column. This is now exemplified for $q = 4$ and $t = 3$:

- The digits 0 to 3 represent GF(q), and the β coefficients are given in Table 7.
- For $t = 3$, the four permutation vectors (h, h^2) are needed, with $h \in$ GF(4) and calculated over GF(4). These are 00, 11, 23 and 32 (using the multiplication table of GF(4)).
- The four corresponding affine combinations can now be calculated using these tuples and the digits of Table 7:
  - The permutation vector 00 yields the first row of Table 7.
  - The permutation vector 11 yields the sum of all three rows of Table 7, taken over GF(4), which is
    01231032230132101032012332102301230132100123103232102301103201 23.
  - The permutation vectors 23 and 32 multiply row 2 with 2 and row 3 with 3 or vice versa, and sum them with row one, yielding for 23
    01232301321010323210103201232301103232102301012323010123103232 10
    and for 32
    01233210103223012301103232100123321001232301103210322301012332 10.
- The additional reduced permutation vector 01 yields 0 times the first row plus 1 times the second row of Table 7, i.e., the second row.

A similar construction can also make the OA($2^{3\ell}, 3, 2^\ell + 2, 2^\ell$) (i.e., $q = 2^\ell$) that improve on the number of columns of the Bush construction for $t = 3$, by using the $q = 2^\ell$ permutation vectors (h, h^2) with $h \in$ GF(2^ℓ) and in addition the reduced permutation vectors from triples (0,1,0) and (0,0,1) (reduced for $t = 4$).

Continuing the example, we just have to add the column from the reduced permutation vector 001 (as 010 was covered by the equivalent 01 above): This adds the third row of Table~@ref(tab:digits) to the design. The transposed completed design is shown in Table 8.

As was illustrated by the example, the length $t-1$ permutation vectors give rise to linear/affine combinations of a base of t linearly independent vectors over GF$(q)^t$. This can be restated more homogeneously using length t permutation vectors as follows (again, for this example with $\ell = 2$, i.e., $q = 2^\ell = 4$, and $t = 3$):

Table 8: The transposed example array

012301230123012301230123012301230123012301230123012301230123012301230123
012310322301321010320123321023012301321001231032321023011032012 3
012323013210103232101032012323011032321023010123230101231032321 0
012332101032230123011032321001233210012323011032103223010123321 0
000011112222333300001111222233330000111122223333000011112222333 3
000000000000000011111111111111112222222222222222333333333333333 3

- The first four coefficient vectors are 100, 111, 123, 132 (prepending 1 to each length 2 permutation vector (h, h^2) for $h \in \mathrm{GF}(4)$), the subsequent permutation vectors are 010 and 001.
- An equivalent but different design is obtained by appending instead of prepending 1 (001, 111, 231, 321, with the subsequent coefficient vectors 010 and 100): swapping two columns and permuting the levels of two further columns transforms it into the original CA.
- An equivalent but different design is obtained by reversing all the permutation vectors, which is equivalent to reversing the β-coefficients (i.e., β_0 becomes β_{t-1}, and so forth, which is equivalent to changing the run order of the full factorial).
- Combining the two reversals of the previous bullets yields the original design again.

A length t permutation vector can be denoted as the number $\sum_{j=0}^{t-1} h_j x^j$; this is very similar to the generators of regular fractional factorial designs, which are often reduced to $k - t$ elements, because the first t elements are taken as the columns of the full factorial.

This paragraph explains how to represent a permutation vector as a number. In the so-called Yates order of regular 2-level designs, which describes the order in which individual vectors and interactions between two or more individual vectors are arranged, the actual vectors of the full factorial are in positions $2^0, \ldots, 2^{t-1}$,e.g., 1, 2, 4, 8, 16 for $t = 5$. For $q = 2$, this is the entire story, as there is only one degree of freedom (df) for each column. For $q > 2$, each factor has $q - 1$ dfs, and it is possible to use the vector itself or one of its multiples w.r.t. $\mathrm{GF}(q)$ multiplication, where multiplication permutes the factor levels. The possible linear combinations consist of all columns that can be reached by permissible β-coefficients; when arranged in the Yates order, and within each effect, in lexicograhic order, the actual vectors of the full factorial are in positions $q^0, \ldots, q^{t-1}$. The permutation vectors 100, 111, 123, 132, 010 and 001 have the numeric representations 1, 21, 58, 45, 4, 16 when using the β-coefficients in the order of the equation or 16, 21, 27, 30, 4, 1 when using the β-coefficients in reverse order. Both provide valid generators; both also contain 1, 4 and 16, so that one could reduce the generator information to the other three numbers, assuming that the full factorial vectors themselves are always part of the array.

3.5 Linear feedback shift registers

A Linear Feedback Shift Register (LFSR) is a sequence that can be obtained from a polynomial of degree d and d start values from $\mathrm{GF}(q)$ that are not all zero: calculate the next element of the sequence by using the $d + 1$ latest elements in the following way (see, e.g., Brent (2004)):

$$\sum_{k=0}^{d} \alpha_k x_{j-k} = 0, \quad j \geq d,$$

with $\alpha_0 = 1$. The α_k arise from a primitive connection polynomial over $\mathrm{GF}(q)$

$$P(z) = \sum_{k=0}^{d} \alpha_k z^k$$

with a generating function

$$G(z) = \sum_{m=0}^{\infty} x_m z^m = P_0(z)/P(z),$$

where $P_0(z)$ has degree at most $d - 1$, depending on the initial values. According to Brent (2004), if $P(z)$ is primitive of degree d and $P_0(z) \neq 0$, the sequence (x_j) has period $2^d - 1$.

3.6 Perfect hash families and variants

Perfect hash families (PHFs) are treated in Colbourn and Dinitz (2007), Chapter VI.43. A PHF$(n; k, q, t)$ is an $n \times k$ array with q symbols ($1, \ldots, q$ or $0, \ldots, q-1$), such that for each set of t&nbps;columns, there is at least one row for which these columns have distinct entries. Of course, this implies $t \leq q$.

There are several variants:

- (t, v)-*distributing* hash families (DHFs) are a generalization of PHFs that uses in addition to t a number of classes $v \leq t$. The separation requirement is relaxed to requesting that every partition of t columns into v classes must be be separated by a row of the DHF, i.e., that this row has different symbols for any two columns of the t-set that are from different parts. The notation is DHF$(n; k, q, t, v)$. According to Colbourn, Dougherty, and Horsley (2019), the concept was introduced in Colbourn (2009); a DHF$(n; k, q, t, t)$ is a PHF$(n; k, q, t)$.
- Colbourn and Torres-Jimenez (2010) extended the concept to (t, v)-"distributing *heterogeneous* hash families" (DHHF); these permit different numbers of symbols between the rows of the array and are denoted as DHF$(n; k, (w_1, \ldots, w_n), t, t)$; $(w_1, \ldots, w_n)$ may be replaced by exponential notation, analogous to OAs and CAs.

3.7 Covering perfect hash families

Covering perfect hash families (CPHFs) were initially introduced by Sherwood, Martirosyan, and Colbourn (2006), who defined them as a PHF(n, k, q, t) with n rows and k columns in $q = v^{t-1}$ levels with a specific property: Each numeric entry in the CPHF stands for an h-tuple, also called "permutation vector" (terminology of the subsequent literature, different from Sherwood, Martirosyan, and Colbourn (2006)) which in turn stands for an affine combination of basic vectors (see Section 3.4). A t-tuple of elements in the CPHF is identified with the corresponding $t \times t$-matrix of h-tuples, as well as with its resulting $v^t \times t$ matrix built by the t affine combinations of basic vectors. It is called "covering", if the latter matrix is a CA of strength t. The CPHF itself earns its "C" by making sure that, for any set of t columns, there is at least one row for which its t-tuple of entries is covering. Thus, a CPHF(n, k, q, t) must be considered in tandem with the h-tuples, and the affine combinations they give rise to. Sherwood, Martirosyan, and Colbourn (2006) proved that the $v^t \times t$ matrix from t affine combinations is a CA if and only if the corresponding $t \times t$ matrix of h-tuples is non-singular (evaluated over GF(q)), so that the covering property can be more cheaply verified than for many other constructions.

Section 3.4 discussed different approaches to permutation vectors, and this report takes the approach to have t-element permutation vectors in combination with Equation (2) and fix one (or more) of their entries for accommodating other choices. With this approach, a CPHF can be exemplified for the CA$(64, 3, 6, 4)$ of Section 3.4.2: It can be obtained as the 1×6 array (1, 21, 57, 45, 4, 16) or the 1×6 array (16, 21, 27, 30, 4, 1). (Note that strength t (S)CHPFs with only one row construct strength t OAs.) For CPHFs with $q > 4$ a prime power, it is important to report the specific GF(q) used for creating the CPHF together with the CPHF. For example, consider the creation of a CA$(512, 3, 10, 8)$ using the construction presented in Section 3.4.2. For the Galois field used in the Sherwood, Martirosyan, and Colbourn (2006) constructions for 8 levels, the permutation vectors are 100, 111, 124, 135, 146, 157, 162, 173, 010, 001, corresponding to the 1×10 CPHF (1, 73, 273, 345, 417, 489, 177, 249, 8, 64). For the other Galois field for $g = 8$ (the one implemented in the R package **lhs**), the h-tuples are 100, 111, 124, 135, 146, 157, 162, 173, 010, 001, corresponding to the 1×10 CPHF (1, 73, 273, 345, 481, 425, 241, 185, 8, 64). It is, of course, important to use the GF on which CPHF creation was based also in CA construction, as in general permutation vectors based on a particular GF(q) will not work when using a different GF(q) in the construction; for example, the permutation vectors 124, 163 and 156 yield a non-singular 3×3-matrix for the GF(8) of R package **lhs**, but yield a matrix with determinant 0 under the GF(8) used by Sherwood, Martirosyan, and Colbourn (2006), whereas the permutation vectors 124, 157 and 173 are covering for the GF(8) used by Sherwood, Martirosyan, and Colbourn (2006) but yield a matrix with determinant 0 under the GF(8) of the R package **lhs**. Hence, neither of the CPHFs works with the respective other GF.

3.8 Difference covering arrays

Shokri and Moura (2025) give a comprehensive overview of many fundamental ingredients, including difference covering arrays DCA(N, t, k, v), with entries from the group $G = \{0, \ldots, v-1\}$. An $n \times k$ array D is a DCA(N, t, k, v) if for any selection $j_1, \ldots, j_t$ of t columns, the difference tuples $(d_{i,j_1} -$

$d_{i,j_2}, \ldots, d_{i,j_1} - d_{i,j_t}$), $i = 1, \ldots, N$ hold each of the $v^{t-1} - 1$ non-zero tuples at least once. It is well-known that the non-zero rows of the multiplication table of GF(q) yield a DCA($q-1, 2, q, q$); if one wants to include zero differences, the entire multiplication table can be used (see e.g. Colbourn, Martirosyan, Van Trung, and Walker (2006)). Shokri and Moura (2025) state that simple direct product construction (see Section 7.4.1) applied to two DCAs with the same t and v yields a DCA($N_1 + N_2, t, k_1 k_2, v$). As a corollary to this, it follows that a DCA($n(q-1)+1, 2, q^n, q$) can be recursively constructed for any prime power q, including zero differences, or with one row less excluding zero differences. Shokri and Moura (2025) furthermore provide a construction for a DCA($q + (q-1)/2, 2, q+1, q$) for odd prime powers q, i.e., e.g., for a DCA($4, 2, 4, 3$) for $q = 3$. Cohen, Colbourn, and Ling (2008) provide a table of DCA sizes (their Table 13) found by simulated annealing for $v = 3, \ldots, 10$ and $k = 2, \ldots, 10$; some of their sizes are smaller than the ones obtainable by the above constructions; unfortunately, the DCAs themselves are not available.

4 Repositories of stored objects

4.1 Stored CAs

This section discusses repositories that, contrary to the Colbourn tables, hold actual arrays. It also mentions how these are used in the R package **CAs**.

- The oldest and most reliable repository is "NIST Covering Array Tables" (n.d.); unfortunately, most of its CAs are larger than the current state-of-the-art eCAN. Package **CAs** keeps information on available CAs in the matrix `NISTcat`. The arrays themselves, even in zipped format, take more than 2GB storage space and are therefore not included in the package. If an internet connection is available, the function `nistCA` downloads the suitable CA for a specified combination of t, k, v, and the function `bestCA` calls `nistCA`, if no other construction yields a design that is at least as good as the NIST CA.
- A repository maintained by José Torres-Jiménez (Torres-Jimenez (n.d.)) apparently used to hold many arrays, judging from references in the literature and an entry on the NIST website. The author had a glimpse at that repository in 2023; its arrays were downloadable one-by-one using a script-based approach. It was hidden behind a login after a request to its provider whether it would be possible to download many arrays at once. On Feb 6, 2025, the author was able to access it once more; at that point, it held a limited number of very good arrays; the information regarding those is stored in `TJcat`, and the smaller ones of the arrays are available in package **CAs**, in the lists `TJ2level_CAs` and `CAEX_CAs`. Meanwhile, the repository is again unavailable, and the server which held it has completely disappeared due to a redesign of th CINVESTAV website. It appears that work is underway to make it public again after a substantial overhaul; a timeline for this was not given, and it is unfortunate that most arrays for which papers state availability in that repository are currently unavailable.
- Kokkala, Meagher, Naserasr, Nurmela, Östergård, and Stevens (2020) enumerated arrays for current best sizes with $v = 3, 4, 5, 6$ and k from 4 to 10, improving some of the sizes even beyond the Colbourn tables. Many of these are available at Kokkala, Meagher, Naserasr, Nurmela, Östergård, and Stevens (2018) (each row corresponds to an array); files are quite large, and quality criteria for distinguishing the different CAs are not available; so far, these arrays are not provided in the package **CAs**; **it should be considered at some point, whether they may help to close some gaps**.
- Arrays from Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010) are available at http://www.sztaki.hu/~keri/arrays/CA_listings.zip. These are provided in the R package **CAs** as `CKRScat` (overview) and `CKRS_CAs` (actual CAs).
- Dwyer (2024) provides a collection of CAs from many sources, which he obtained via personal communication with Charlie Colbourn. There are 211 arrays, about half of which correspond to the current optimum, among them density constructions and Simulated annealing (TJ-RT). There is slight overlap with `CKRScat`, but most arrays are distinct. Their overview information is provided in R package **CAs** in the `data.frame` object `DWYERcat`. If an internet connection is available, the function `dwyerCA` downloads the array for the specified combination of t, k, v, and the function `bestCA` calls `dwyerCA`, if no other construction yields a design that is at least as good as the CA from the Dwyer database.
- The 43 arrays from Wagner, Kampel, and Simos (2021) are available at https://srd.sba-research.o

rg/data/sipo/. They are all binary and have strength 6. Their overview information is included in the package **CAs** as `WKScat`, and the arrays themselves are in the list `WKS_CAs`.

4.2 Stored cover starters

Cover starters are much smaller objects than CAs: they consist of single vectors, with a recipe for expansion into a CA given in a paper. To the author's knowledge, they are not expressly provided in repositories. They were therefore typed or copy-pasted into R from the papers and theses in which they were found, and the coverage of the resulting CAs was checked (which is doable with limited effort, because they are for strength 2).

- The cover starters from Meagher and Stevens (2005) are in `MeagherStevensStarters` with corresponding info in `MeagherStevensCombis` and construction in function `CS_MS`. In Colbourn (n.d.), these are referenced as "Fix 1 symbols (Meagher-Stevens)".
- The cover starters from Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) (CMMSSY) use a shorter and distinct cover starter, and always yield an SCA (see Section 7.4.3), which is an advantage for potential recursive constructions. Besides the starter vectors available from the paper, it is likely straightforward to search for further such starter vectors; Colbourn (2008) mention a search for cover starters of that type, in analogy to Meagher and Stevens (2005), but do not present the resulting cover starters beyond using them for their existence tables. **check whether these can be found or created**
- The Colbourn tables also contain the source entry "Cover starter (Colbourn, Torres-Jimenez)", for $t = 3, v = 3$, and k one of 14, 16, 46 to 49. These could so far not be located.
- Colbourn and Kéri (2009) provide cover starters for strength 4 CAs with 2-level columns. These are available in `ColbournKeriStarters`. In addition, they provide verified constructions with quadratic residue vectors for strengths 5 and 6, which, together with the strength 4 settings, are provided in `ColbournKeriCombis`.
- Lobb, Colbourn, Danziger, Stevens, and Torres-Jimenez (2012) (LCDST) provide extensive lists of cover starters in the paper, for $v = 3, \ldots, 20$; their cover starters have more than one fixed point, which requires an additional ingredient that is to be used for covering pairs among fixed points. The paper also provides references for these ingredients; unfortunately, a frequent reference is the repository by Torres-Jiménez, which is unavailable at this time. The construction is described in Section 6.3.

  The cover starters for up to 9 levels (exceptions: $k = 8$ for $v = 5$, which is covered by the Meagher and Stevens (2005) construction; $k = 13$ for $v = 8$, **TODO** which has not yet been figured out) have been incorporated into `LCDSTStarters`, with $\infty_0, \ldots, \infty_{f-1}$ (f the number of fixed points) coded as $100 + v - 1, \ldots, 100 + v - f$. Selected cover starters for 10 and 11 levels are also included (easy to handle, which excludes $k = 14$ with $v = 10$, improve on `CS_MS`). Info on the available combinations, including the construction of the additional ingredient, is in `LCDSTCombis`, and the construction function is `CS_LCDST`.

4.3 Stored (C)PHF

Dwyer (2024) does not only provide a CA database but also a PHF data base and a CPHF database, where the latter contains SCPHFs provided by Erin Lanus. There are also other sources for (S)CPHFs, namely the Sherwood, Martirosyan, and Colbourn (2006) paper (SCPHFs available in the package, including the ones provided from Dwyer (2024)) and the download repository (https://srd.sba-research.org/data/cphf ipo/cphfs.zip) for the Wagner, Colbourn, and Simos (2022) paper (the smaller ones are available in the package). Constructions using SCPHFs and CPHFs are available in the package (see Section 6.6). A construction that uses PH(H)Fs has not yet been implemented in a way that is convenient to use (apart from the construction of Section&nbps;7.1.1), which is why there are no stored PHFs either. It would likely be worthwhile to store some PHFs or to provide analytic PHF constructions for relevant settings; most PHFs behind Colbourn table entries starting with "perfect hash family" have not yet been found.

5 Transformations of single CAs

This subsection covers methods that transform a single array, by

- reducing the number of levels and thereby also the number of rows (fuse)

- substantially reducing the number of rows by reducing the strength and sacrificing one column (derive)
- adding a column and reducing the number of levels by 1 (strength 2 only, projection), which can also be repeated
- postprocessing it for reducing the run size

In principle, the tools of this section permit the determination of a run size for a desired scenario, as it is straightforward to determine, which ingoing array will yield the desired outcome; this is, however, limited for "projection", because applicability of "projection" requires a "private row" in the ingoing array (see Section 5.3). Therefore, the R package **CAs** holds functions `Ns_fuse` and `Ns_derive`, but no such general function for projection; it has an immplementation of projection for B_q only (function `projBoseCA`).

5.1 Fuse

"fuse" is a simple technique for reducing the number of levels by 1 and the number of rows by 2, and can be applied repeatedly. A special case for which the number of rows can be reduced by 3 will be discussed separately later. The construction is given in the proof of Lemma 3.1 of Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010). It is implemented in R package **CAs** via the functions `fuse` for a user-specified ingoing array and `fuseBushtCA` for the ingoing array an orthogonal array of strength t from the Bush construction (Bush (1952)); in the latter case, the user needs to specify t, k and v, and the function works out a suitable Bush OA from which fusion yields an appropriate CA (if possible). Fusion processes the ingoing array as follows:

- Recode all columns such that the first row has the largest level, and remove the first row.
- Also eliminate the second (now first) row, in the following way:
  - For columns, in which its entry is the largest level, replace the entries of the largest level with an arbitrary value (or a don't care value, **currently not implemented**).
  - For columns, in which its entry is not the largest level, replace the entries of the largest level with the entry of the row that is eliminated.

The above-described variant of fusing works for all symmetric CAs. When applying fusing to B_q, one can omit three rows (once only), based on a graph coloring approach. This is implemented in function `fuseBose` and yields a CA$(q^2-3, 2, q+1, q-1)$ by eliminating three rows from B_q, as stated in the proof of Lemma 3.2 in Colbourn (2008).

5.2 Derive

"derive" is a simple technique, by which a CA of strength $t+1$ can be made into a CA of strength t in one less column by choosing a column i and a level ℓ of that column, deleting all rows for which column i does not have level ℓ and subsequently deleting the ith column. One should of course use a column and level that has the smallest possible frequency. This is the construction behind Equation (7) of Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010).

5.3 Projection

According to Torres-Jimenez, Izquierdo-Marquez, and Avila-George (2019), "projection" refers to a technique that consists in adding a column and reducing the number of levels for each existing column by 1, as well as reducing the number of rows by 1; the method was suggested in Colbourn (2008). Its prerequisite is the presence of so-called $\mathcal{I}$-*private* rows in the in-going design, where $\mathcal{I}$-private means that, for a set $\mathcal{I} = \{i_1, ..., i_s\}$, the entries of the $\mathcal{I}$-private row are unique w.r.t. each pair of columns indexed by a pair of indices in $\mathcal{I}$. As the condition of privacy is hard to check, Colbourn (2008) presented a construction based on B_q, for which the privacy condition is known to be fulfilled for the first q rows. Theorem 2.3 of Colbourn (2008) allows repeated application of projection up to $c = q-2$ times, in which case the resulting array would have $q - c = 2$ level columns. The construction permits to reduce the number of runs from q^2 to $q^2 - c$, while reducing the number of levels from q to $q - c$ (q a prime power) and increasing the number of columns from $q+1$ to $q+1+c$; the c additional columns can have s levels, with $2 \leq s \leq q - c$, i.e., the resulting array can be uniform for $s = q - c$ or have mixed levels for smaller s. As the proof of Colbourn (2008) is not too explicit regarding $c > 1$, it is very helpful that the construction is also illustrated in Figure 25 of Torres-Jimenez, Izquierdo-Marquez, and Avila-George (2019). Function

`projectionBose` of R package **CAs** implements the construction with general s, function `projBoseCA` provides a uniform CA from this construction (i.e., $s = q - c$) for a given number of columns k and number of levels v. In principle, an array from this construction would always be possible, if necessary by removing columns from the Bose OA for sufficiently large q; however, it will of course have terrible properties for $v >= k$, so that the function requires $k > v$; the construction contributes several source entries to the Colbourn tables, however, mostly with post-processing by exploitation of flexible values, and always with large number of levels ($v \geq 9$, and mostly $v \geq 14$); it can yield reasonably competitive arrays also for smaller cases, e.g., a CA$(48, 2, 9, 6)$, where the optimum is CA$(46, 2, 9, 6)$ from the Meagher and Stevens (2005) or Colbourn, Martirosyan, Van Trung, and Walker (2006) cover starter constructions.

5.4 Post-processings for reducing the run size

Post-processing refers to search-based techniques that process a given CA by

- reducing its number of rows (postop NCK, postop TJ-SA)
- extending the number of columns, like, e.g., the "expansion" step used in Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021).

This section discusses the former case only.

5.4.1 postop NCK

"postop NCK" refers to postprocessing as described in Nayeri, Colbourn, and Konjevod (2013); it occurs in 397 entries of the Colbourn tables. The method is based on detecting wildcard entries (also called "don't care values" or "flexible values", i.e., values that can be changed without deteriorating the CA's strength) and removing rows that consist of wildcard entries only. The wildcard identification step is implemented as function `markflex` in R package **CAs**; it relies on a first step of identifying potential flexible positions (function `flexpos` in R package **CAs**) and moving rows with many flexible positions to the bottom of the array, before applying the duplicate marking algorithm, in order to increase the chance of eliminating entire rows. It may be very resource-intensive for arrays with many columns and/or high strength. It is implemented in function `postopNCK` of R package **CAs**. The algorithm has an outer iteration step, and within each outer iteration, an inner iteration loop. When escaping computations, the result from the last successful outer iteration is returned. Some detail can be found in the function documentation, and in the code.

5.4.2 postop TJ-SA

"postop TJ-SA" refers to a postprocessing that could not yet be located; it applies to strength 2 constructions only. TJ likely stands for Torres-Jimènez, SA might be a person's initials (no such paper was found so far), or stand for simulated annealing (no paper with single author Torres-Jimènez for this found either). It might refer to Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021), which has a section on simulated annealing which refers to Avila-George, Torres-Jimenez, Gonzalez-Hernandez, and Hernández (2013) for this part of the work; this conjecture is compatible with the fact that this is for strength 2 constructions only.

6 Relevant mathematical constructions without CA ingredients

This section presents constructions that create CAs from (relatively) small ingredients like a prime (Cyclotomy, Paley constructions) cover starters or codes. Constructions that involve one or more CAs and possibly other ingredients are covered in the next section.

6.1 Cyclotomy (`cyclotomyCA`)

Colbourn (2010) introduced a construction he named "cyclotomy". Its ingredients are a prime power q and the corresponding Galois field GF(q) and a construction type (one of 1, 2, 3, 3a, 3b, 4, 4a, 4b). Extensive computer searches by Colbourn and others have explored which prime together with which construction type yields which coverage strength.

For all construction types, the prime q yields a cyclotomic vector, i.e., a vector of the logarithms of the elements of GF(q) w.r.t. a primitive element ω of GF(q), which can be cyclically moved to return an

ingredient matrix A. For example, with $q = 5$, 2 is a primitive element, because it yields all non-zero elements of GF(5) as its powers: $2^1 = 2, 2^2 = 4, 2^3 = 3 \mod 5, 2^4 = 1 \mod 5$. When using the prime 5 for a CA in 4 levels, the cyclotomic vector would be $x = 00132$, because it is the vector of logarithms of 1 to 5, taken $\mod 4$: $\log_2(1) = 4$, $\log_2(2) = 1$, $\log_2(3) = 3$, and $\log_2(4) = 2$. Labelling the elements of x with labels 0 to $q-1$, element a_{ij} of the $q \times q$ matrix A is then obtained as x_{j-i}, where the difference $j - i$ is taken over the additive GF(q), in our case simply as $(j - i) \mod 5$. This yields the matrix A with rows 00132, 20013, 32001, 13200, 01320 (for a prime, just a cyclic permutation; more complicated for prime powers). The matrix A is the key ingredient for all construction types.

The function `cyc` is the workhorse function for the construction; experts can use it directly. It yields designs with columns in `v` levels based on prime or prime power `q` based on the matrix A as follows:

- for $N = k = q = 1 \mod v$ a prime power (Construction 4.1, `type=1`)
- for $N = q + v - 1$, $k = q$, with $q = 1 \mod v$ a prime power (Construction 4.2, `type=2`)
  (adds $v - 1$ constant rows with elements 1, 2, ..., v-1 to construction 4.1, as these are missing)
- for $N = vq$, $k = q$, with $q = 1 \mod v$ a prime power (Construction 4.3, `type=3`)
  (juxtaposes vertically A, $A + 1 \mod v$, ..., $A + v - 1 \mod v$; this is called developing modulo v)
  - or Construction 4.3a: $k = q + 1$
    (extends A by a column of zeroes before developing)
  - or Construction 4.3b: $N = vq + v^2 - v$, $k = q + 1$
    (adds non-zero constant rows to A (like in 4.2),
    then adds column of 0s before developing)
- for $N = vq + v$, $k = q$, with $q = 1 \mod v$ a prime power (Construction 4.4, `type=4`)
  (like Construction 4.3, but adds a constant row of zeroes to A before developing)
  - or Construction 4.4a: $k = q + 1$
    (also adds a constant column of zeroes before developing)
  - or Construction 4.4b: $N = vq + v^2 - v$, $k = q + 1$ (same as 3.b)
    add to A all nonzero constant rows like in Construction 2, and a column of *ones*, then develop.

The data frame `CYCLOTOMYcat` in the R package **CAs** contains details for the implemented constructions. Based on this data frame,
function `cyclotomyCA` allows to specify t, k and v and to obtain the smallest *implemented* CA from the cyclotomy construction for this specification (if one is implemented in the package **CAs**). Implementations have been limited to those CAs that have optimum size as of the state of the Colbourn tables of best bounds of November 2024, for which implementation details were be inferred from the combination of N, k and v; this is because various mistakes were found in tables of Colbourn (2010), and the package author does not want to take responsibility for the strength of CAs according to those tables from the paper. Expert users who know that a certain prime power q yields a smaller CA for their requirements can use function `cyc` for creating that CA.

The implemented constructions include three strength 5 cases attributed to Torres-Jimenez, for which $q \mod v = 2$ (and not 1, as required in Colbourn 2010). Limited checks by sampling suggest that these are OK, but no full checks have been conducted, and the implementation trusts that the Colbourn tables are correct (except in one case, which was omitted because the alleged q is not a prime or prime power (13*229)).

6.2 Paley-based constructions (`paleyCA`)

In experimental design, Paley constructions are known because of their use for creating orthogonal arrays as $N - 1$ columns of $N \times N$ Hadamard matrices (Hadamard matrices are $N \times N$ square matrices starting with a column of $+1$ entries, entries $+1$ and -1 only, and row sums and scalar products between columns all zero). Hadamard matrices exist for number of runs a multiple of 4. Here, to be in sync with more than two levels, their typical $-1/+1$ encoding is replaced by 0/1 notation, with 0 corresponding to $+1$ and 1 to -1.

6.2.1 The Paley matrix

A Paley conference matrix (see Section V.6 of Colbourn and Dinitz (2007)) exists for odd primes q and is a $(q+1) \times (q+1)$ matrix, which is symmetric for $q \equiv 1 \mod 4$ and skew for $q \equiv 3 \mod 4$ (see Theorem 6.7 in Chapter V of Colbourn and Dinitz (2007)). The skew Paley matrices are Hadamard matrices, whereas Hadamard matrices constructed from symmetric Paley matrices have dimension $(2q + 2) \times (2q + 2)$. CA

construction is the same for both types of Paley conference matrices. Function `conference_paley` from R package **CAs** constructs the *core* of the Paley conference matrix, which corresponds to its bottom right $q \times q$ block, and function `paley_mat` a Paley matrix, which can be obtained from that core by replacing all -1 entries with zeroes. At the same time, the Paley matrix equals the matrix A that is used in the cyclotomy construction for $v = 2$.

For $q \mod 4 = 3$, the $q \times q$ Paley matrix P_q is the bottom right block of a $(q+1) \times (q+1)$ Hadamard matrix H, whereas there is no such small related Hadamard matrix for P_q with $q \mod 4 = 1$. Nevertheless, regardless of whether or not $q \mod 4 = 3$, the matrices

$$\begin{pmatrix} 1 & 1 \\ 1 & P_q \end{pmatrix}, \quad \begin{pmatrix} 1 & P_q \end{pmatrix}, \quad \begin{pmatrix} 1 \\ P_q \end{pmatrix}, \quad P_q \tag{3}$$

can be used for creating CAs. Colbourn (2015) ran numerical investigations on *generalized types* of those matrices (see next section), with the goal of using them for the creation of small CAs by removing unnecessary rows. As q must be prime or odd prime power, Paley matrix constructions can not be used to construct CAs for all conceivable values of N.

6.2.2 Generalized type of a matrix

Colbourn (2015) introduced the generalized type and generalized half type of a Paley or related matrix and numerically explored it for prime $q < 500$.

The generalized t-type of a matrix M is the minimum number of times with which the worst-case tuple $\langle a_1, ..., a_t \rangle$ (denoting the pair of rows $a_1, ..., a_t$ and $1 - a_1, ..., 1 - a_t$, with $a_j \in \{0, 1\}$) is covered in the worst-case set of t columns of M, originally introduced for M a Hadamard matrix and $t = 4$. If this generalized t-type is positive, the doubled matrix $\binom{M}{1-M}$ has t-coverage; the generalized t-type of M can also be seen as the minimum frequency with which any t-tuple is covered by the *doubled* matrix M. Hence, a positive generalized t-type is equivalent to $\binom{M}{1-M}$ having t-coverage. Colbourn (2015) furthermore introduced the t-halftype, which refers to M itself instead of its doubled version. Table 1 of the paper contains generalized t-types and t-halftypes for strengths $t = 3, 4, 5, 6$ and primes (not prime powers) up to 500 for all four matrices of Equation (3), where the full types refer to the doubled matrices with the extra column and the half types to the matrices without the extra column.

Once the t-type or t-halftype ψ is known, this can be used for removing rows from the (doubled) matrix without sacrificing t-coverage. For example, if the t-halftype is 3, it is safe to remove two rows from the CA, because each row contains at most one t-tuple of each sort, so that removal of two rows leaves at least one of each tuple in the array. Trivially, a positive t-halftype also indicates that matrix M has t-coverage, which is, e.g., the case for the 12×12 Hadamard matrix and for all Hadamard matrices based on $q \geq 17$.

6.2.3 Insights from Colbourn's numerical experiments

Colbourn's Table 1 lists results of those experiments; there are a few small mistakes for small q ($q = 7, 13, 17$). The following summary describes the most important insights:

- For strength 3, only the small q yield competitive run sizes, and only without doubling. This case is therefore not further discussed.
- All Paley matrices from primes $q >= 67$, except for $q = 73$, have 4-coverage. For $q \geq 89$, except for $q = 103$, one or more rows can be omitted without sacrificing 4-coverage. For smaller q, doubling yields strength 4 for $q = 11$ and $q \geq 19$. For $q \geq 23$, at least two rows can be removed (one row from each half).
- Paley matrices from primes 359 379 431 433 463 467 487 491 499 have 5-coverage; for primes 431 463 467 487 491 499, one row can be omitted without destroying 5-coverage; as one row can even be omitted from the Paley matrix, a strength 5 CA has $N - 2$ rows only versus a classical Paley design matrix. These are the current best designs according to the Colbourn tables. For primes $q \geq 67$, except $q = 73$, doubling Paley matrices yields 5-coverage. For $q \geq 89$, except for $q = 103$, at least two rows can be removed (one row from each half).
- For primes 359 379 431 463 467 487 491 499, doubling of Paley matrices yields 6-coverage; for primes 431 463 467 487 491 499, at least two rows can be removed.

Colbourn (2015) did not experiment with prime power q, although these can also be used. Prime powers 9, 25, 27 and 49 were explored by the author, but did not yield better CAs than the ones resulting from Colbourn (2015). They are therefore not in `PALEYcat`.

6.3 Cover starters

There are various constructions with so-called "cover starters" (see also Section 4.2), where a cover starter is a vector that is used for starting a construction algorithm for a larger CA. In the Colbourn tables, the expression "Cover starter" does not appear as often as one could expect, as various entries based on cover starters appear under "Fix ... symbols", emphasizing the group action that is the second ingredient to the respective constructions. Another source entry for cover starter constructions is "cyclic", based on cyclic interim processing steps.

6.3.1 Meagher and Stevens (`CS_MS`)

This construction is referenced in the Colbourn tables as "Fix 1 symbols (Meagher-Stevens)". Meagher and Stevens (2005) give a construction that has as ingredients a starter vector (stored for many combinations of k and v) and a group G that depends on v only. Their construction works for $v \geq 3$. For this account, the levels are denoted as $0, 1, \ldots, v-1$.

All starter vectors hold elements from $\{0, 1, \ldots, v-1\}$ and have the first and only the first element as 0.

- The starter vector is cyclically permuted into a square matrix M, which thus has all-zero diagonal elements.
- The group G is the subgroup of the group of all permutations of the v elements that cyclically permute all non-zero elements and leave the zero element fixed. (when applying it element wise to M, G thus leaves the diagonal unchanged.)
- The construction consists of concatenating a row of zeroes, an unchanged M, and copies of M with all $v-2$ non-identity elements of G applied element-wise to M (which is the same as looping through applying the cycling by one position and appending each interim result).

6.3.2 Colbourn, Martirosyan, Mullen, Shasha, Sherwood and Yucas (`CS_CMMSSY`)

The Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) variant of cover starters, for the vectors provided in the paper, fixes one symbol, like the Meagher and Stevens (2005) construction. It always arranges the resulting array in the form of an SCA, which is a substantial benefit, because it allows to combine arrays by taking their products with function `productPCA` (see Section 7.4.3); Meagher and Stevens (2005) constructions can sometimes but not always be arranged as SCAs.

The approach to fixing one symbol in this construction is arguably simpler than that of Meagher and Stevens: Instead of activating the symmetric group with cyclic permutations, the to-be-fixed symbol is denoted as ∞, the finite symbols are treated with addition modulo $v-1$, and the ∞ symbol is eventually recoded to $v-1$. This approach is also taken by other fixed symbol cover starter constructions, where, for fixing more than 1 symbol, notations like $\infty_0, \infty_1, ...$ are introduced.

The cover starters for the construction are in `CMMSSYStarters`, all constructions that can be created from these are in the data frame `CMMSSYCombis`. These include

6.3.3 Lobb, Colbourn, Danziger, Stevens and Torres-Jiménez (`CS_LCDST`)

Strictly speaking, this construction would belong into Section 7, as it employs a CA ingredient; nevertheless, it was decided to group it with other cover starter and fixed symbol constructions. Lobb, Colbourn, Danziger, Stevens, and Torres-Jimenez (2012) provide cover starters with fixing more than one symbols; the paper and the Colbourn tables have entries with up to 7 fixed symbols. The R package covers the constructions for up to 11 levels, which have $f \leq 4$ fixed points. (The limit was chosen because the implementation requires manual processing of each individual starter vector, and because larger v are not considered too relevant.) The special case for $k = 13$ with $v8$ was omitted, because it does not work with the group $\mathbb{Z}_{v-f}$ (like the implemented cases) and because its run size can be achieved by function `CS_MS` discussed above.

Fixing more than one symbol makes it inadequate to only work with a a single starter vector and a single cycling approach, because the two fixed levels would only occur in fixed distance to each other (as they

are not modified by group action); consequently, the pairs with a different distance would not be covered. Therefore, the construction must be made more complicated. The construction handles this by bringing in additional rows for covering between-fixed-symbols pairs. These are based on a CA($N_{f,k}, 2, k, f$), where f is the number of fixed symbols. For the construction with length k starter vectors (like in `CS_MS`), the run size is $(v - f) \cdot k + N_{f,k}$, where $N_{f,k}$ denotes the run size of the extra CA. As was mentioned before, the R package **CAs** implements the cases for up to $v = 11$ with a few exceptions.

For $f = 2$ and $f = 3$ fixed points, the ingredients CA$(2, k, f)$ for covering interactions between fixed points can be obtained with functions `KSK` and `CAEX`, respectively, and are empirically optimal. For larger f (at present, $f = 4$ only), the best implemented ingredient designs are used, which may not be the best existing designs in all cases. In August 2025, the best existing design for all relevant cases ($k = 25, 26, 27$) with $f = 4$, according to Colbourn (n.d.), has 29 runs and is from simulated annealing by Torres-Jiménez (still the same as in the paper); `LCDSTCombis` holds the N for using this unavailable CA; function `N_CS_LCDST` (and thus also function `Ns`) increases it to the doable run size based on `bestN(2,k,f)`. In the example, the best implemented CA is from the recursive construction of Section 7.5 that exploits the PCA structure of B_4, which has 40 runs (obtainable as `recBoseCA(2,k,4)`), i.e., all implemented arrays with $f = 4$ are 11 runs larger than would be possible if José Torres-Jiménez would share his 29 run CA; as such, they are still the best implemented CAs.

All the starters in R package **CAs** have length k; the tables contain selected length $k - 1$ starters; for the implemented numbers of levels, these have been omitted, because CAs of the respective sizes can be obtained by function `CS_MS`. For larger numbers of levels, there are cases where such a vector yields better results than other constructions. Such vectors from the paper can be manually used in the `CS_CMMSSY` function, e.g., for 13 levels with 18 columns; for making this work, the fixed element must be rotated to the first position. The arrays are included in Colbourn (n.d.) with a source entry containing "(LCDST)" and "Fix ... symbols"; in various cases, "postop TJ-SA" or "postop NCK" is mentioned for reduction of the final array size. The former is not (yet) available in the package, the latter does not yield improvements in its current implementation (see Section 5.4.1).

6.3.4 Colbourn and Kéri (`CS_CK`)

Colbourn and Kéri (2009) provide three construction methods (their lemmata 1 to 3) for creating binary CAs of strengths 4, 5 or 6. Starters for strength 4 are explicitly given in the paper (and in `ColbournKeriStarters`), starters for other strengths are obtained via quadratic residue and are calculated in the package using function `cycvec` (which was originally implemented for the cyclotomy constructions). Besides direct constructions from using the starter vectors (using internal functions `CS_CK1` to `CS_CK3` corresponding to the lemmata of the paper), the method also includes derived arrays for strengths 4 and 5, where appropriate. Construction details are provided in the data frame `ColbournKeriCombis`. The source entry in the Colbourn tables is "Cyclic (Colbourn-Keri)" or "Cyclic, derived (Colbourn-Keri)". **explain how it works**

6.3.5 Unknown cover starter construction

The Colbourn tables have only 18 entries that actually contain the expression "cover starter": they all say "cover starter (Colbourn,Torres-Jimenez)" with slight variations in capitalization and usage of blanks. They have strength 2 with 5, 6 or 7 levels or strength 3 with 3 levels. So far, the corresponding construction was not found.

6.4 Chateauneuf and Kreher NRB (`CK_NRB`)

Near resolvable block designs (NRB, see Colbourn and Dinitz (2007), Chapter II.7) can be used for a proven optimal CA$(33, 3, 6, 3)$ and a current best CA$(88, 3, 8, 4)$ (Chateauneuf, Colbourn, and Kreher (1999)) and a current best CA$(185, 3, 10, 5)$ (Chateauneuf and Kreher (2002)).

6.5 Cross-summing codes

Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010) provided a construction based on cross-summing two codes `A` and `B`, in most cases choosing `A` as a repetition code (RC), direct product of two repetition codes (DRC), or an extended version with extra constant columns of 0s (ERC or EDRC). The cross-summing and the direct sum (for the DRC) are provided in R package **CAs** in functions `crossSum`

and `directSum`, respectively. Unfortunately, it is at present unclear how to obtain suitable codes for the role of `B` in the construction.

6.6 The Sherwood, Martirosyan and Colbourn (2006) (`scphfCA`) construction based on permutation vectors

Sherwood, Martirosyan, and Colbourn (2006) introduced CPHFs in a special form that was later named "Sherwood CPHFs" or SCPHFs. Later authors used both SCPHFs and CPHFs that do not follow that specific form. The following two sections present both approaches. The expression "permutation vectors" is used in the sense of the majority of the literature which is different from Sherwood, Martirosyan, and Colbourn (2006)'s original contribution. As mentioned in Section 3.7, the v^t columns for creating the CA must be calculated using the same Galois field instance that was used for constructing the (S)CPHF.

For prime powers $8 = 2^3$ and $9 = 3^2$, Sherwood, Martirosyan, and Colbourn (2006) and Walker II and Colbourn (2009) used different instances than the R package **lhs** (Sherwood, Martirosyan, and Colbourn (2006): characteristic polynomials $x^3 + x + 1$ for 8 and $x^2 + 2x + 2$ for 9; Walker II and Colbourn (2009) the same for 8 and a different one for 9, given via its multiplication table at https://github.com/robbywalker/ca-phf-research/blob/master/tabu/math/galois_field.cpp; R package **lhs**: characteristic polynomials $x^3 + x^2 + 1$ for 8 and $x^2 + x + 2$ for 9). Thus, either the CPHFs for those prime powers have to be re-done, or a different GF must be used than what is used otherwise in the package; the latter route is chosen by the function `SMC` of R package **CAs**.

6.6.1 SCPHF-based

Sherwood, Martirosyan, and Colbourn (2006) introduced the concept of calculating a CA by vertically concatenating one or more blocks of runs that are composed as affine combinations of v^t-dimensional basic vectors. The information on which combinations of the basic vectors to use is provided in an SCPHF. Sherwood, Martirosyan, and Colbourn (2006) also provided several specific SCPHFs for strengths 3 and 4. Such SCPHFs were also provided by Walker II and Colbourn (2009) (at a GitHub repository by Walker: https://github.com/robbywalker/ca-phf-research/tree/master/results) and by Colbourn and Lanus (2018b) (provided by Lanus for the Dwyer (2024) repository).

SCPHFs use length $t - 1$ permutation vectors (=h-tuples). For given SCPHFs, the construction is simple, using as background information the basics of permutation vectors, as stated in Section 3.4:

- Create the q^{t-1} potential permutation vectors $(h_1, \ldots, h_{t-1})$ with elements from GF(q), and arrange them in the order of the base q representation of the numbers 1 to q^{t-1} ($(t-1) \times q^{t-1}$ matrix; top row for power 0, bottom row for power $t - 1$).
- Use a pre-calculated $n \times k$ SCPHF of strength t with entries from 0 to $q^{t-1} - 1$; the entries can also be given as $(t-1)$-tuples of entries from GF(q), or they can be backtransformed to such tuples.
- For a row of the SCPHF, obtain a $q^t \times k$ matrix as follows:
  - For the jth column, select the suitable permutation vector $(h_1, \ldots, h_{t-1})$ as indicated by the SCPHF entry.
  - Create the ith entry of the jth column as $\beta_0(i) + \beta_1(i)h_1 + \cdots + \beta_{t-1}(i)h_{t-1}$, where $\beta_0(i), \beta_1(i), \ldots, \beta_{t-1}(i)$ are the digits for powers 0 to $t - 1$ of the base q representation of i.
  - Via this step, each row of the SCPHF yields a $q^t \times k$ matrix with q constant rows.
- Vertically concatenate the n matrices, omitting the constant rows in all but one case. This yields $n(q^t - q) + q$ runs for the result.

6.6.2 CPHF-based

This section discusses the approach with length t permutation vectors, different from the length $t - 1$ permutation vectors of the previous subsection.

The difference lies in the entry for power zero, which is always made 1 for SCPHFs (implying q constant rows), whereas it has its own variable coefficient in the permutation vector for CPHFs without the "S" for Sherwood.

Wagner, Colbourn, and Simos (2022) constructed CPHFs for this construction. With a strength t $n \times k$ CPHF, a CA$(n(v^t - 1) + 1, t, k, v)$ is created.

7 Relevant mathematical constructions with CA ingredients

This section presents constructions that need at least one CA and some other ingredient(s):

- Power constructions need a DHHF and one or several CAs (see first subsection)
- Composition, direct product constructions and doubling need two CAs, and can be applied recursively
- the Covering array eXtender (Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021)) refers to an effort undertaken by Torres-Jiménez that yields exceptionally good CAs for strength 2 3-level CAs.

7.1 Power constructions

According to Torres-Jimenez, Izquierdo-Marquez, and Avila-George (2019), **Power** refers to power constructions, which Torres-Jimenez, Izquierdo-Marquez, and Avila-George (2019) attributes to Hartman (2005). The fundamental idea is simple and uses as ingredients a strength t CA D with k columns in the desired number of levels v, and – that is the hard part – another array P with k levels in as many columns and as few rows as possible that is at least a distributing hash family of strength t with v classes ($v \leq t$). Number the columns of the former array by the levels of the latter array, and then, in the latter array, replace each entry by the column corresponding to this entry from the former array.

Distributing heterogeneous hash families are used in the role of P: according to Colbourn, Dougherty, and Horsley (2019), the concept of (t, v)-"distributing hash families" (DHF) was introduced in Colbourn (2009); a DHF with $v = t$ is a perfect hash family. Colbourn and Torres-Jimenez (2010) extended the concept to (t, v)-"distributing *heterogeneous* hash families" (DHHF). There are simple constructions for DHHF, using transposed orthogonal arrays as the starting point:

- According to Lemma 1.2 of Colbourn and Torres-Jimenez (2010), for $q > t$, the transpose of $\mathrm{OA}(q^s, s, q + 1, q)$ from the Bush (1952) construction yields a (t, v)-DHF in $k \leq q + 1$ rows and q^s columns with q&nbps;levels in each row, if $k \geq (s - 1) \cdot \mathrm{Turan}(t, v)$, where $\mathrm{Turan}(t, v)$ is the Turan number.
- According to Lemma 3.6 of the same paper, if $q > s$, and denoting M as the number of rows of the (t, v)-DHF obtained from the $\mathrm{OA}(q^s, s, q + 1, q)$, distributing heterogeneous hash families can be obtained by removing all columns with a certain set of levels in a certain row, e.g. reducing several rows of the DHF by the same number of levels (e.g., by 1) or reducing one row to α levels or one row to α and one to β levels. The lemma states pessimistic numbers of columns for those distributing heterogeneous hash families.

There is a large number of power construction source entries in the Colbourn tables; they all refer to CAs of strength at least 3. Their labels start with "CT" for Colbourn and Torres-Jimenez (2010), "CZ" for Colbourn and Zhou (2012) which uses so-called "quilted" arrays instead of CAs and requires modified DHFs for adequate creation of a CA **that do not seem easily available**. There are also a large number of entries that start with "N-CT", whose source is as yet not known; they are of high strength and have very large run sizes (min 221500, 5%-quantile about 1.4 Mio, 10%-quantile about 5.7 Mio, first quartile about 27 Mio); the achieved numbers of columns are between 483 and 10000 (the upper bound used by Colbourn in the tables).

7.1.1 CT constructions (partly implemented, `powerCA`)

The CT constructions use a Perfect *Heterogeneous* Hash Family (PHHF) or a Distributing Heterogeneous Hash Family (DHHF) in the role of P; heterogeneous means that P may have rows with different numbers of levels. As was stated above, the number of rows of the DHF or DHHF must exceed

$$(s - 1) \cdot \mathrm{Turan}(t, v),$$

where s is the strength of the ingoing OA (`toa` in the package **CAs**), Turan is the Turan number, t is the target strength of the DHF (`tdhf`; must be at least the strength of the resulting CA), and v is the number of classes of the DHF (`vdhf`; t in case of a PHF, at least v for constructing a CA with v levels); for a PHF, this bound is $(s - 1) \cdot (t - 1) \cdot t/2$.

For each number of levels in the DHHF, there must be a CA D_i with as many columns as needed for the respective rows of P; the best available CAs with as many as possible constant runs should be used for D_i. The paper provides many instances for 2-level CAs. The Colbourn tables also contain arrays with

more than two levels. These can be created by constructing the DHHFs corresponding to the construction from the paper according to the information stated in the source entry; *at present, symbols "S" and "Arc" in the source entries are not understood, neither is symbol "c".* For example, the CA corresponding to the source entry "Power CT11^3T7T3c" for a CA$(49, 3, 352, 2)$ is created as follows (the "c" in the source entry presumably refers to the fact that the CA with 8 columns must be chosen as one with two constant rows):

- Construct the smallest DHF for the prime $q = 11$ with 11^3 columns and of strength $s = t = 3$, with $v = 2$ classes:
  `P <- createDHF(SCA_Busht(11,3), toa=3, tdhf=3, vdhf=2).`
  This DHF corresponds to Lemma 1.2 of Colbourn and Torres-Jimenez (2010) and has $M = 5$ rows; its number of rows must exceed $(3 - 1) \cdot \text{Turan}(3, 2) = 2 \cdot 2 = 4)$, which it does.
- Using function `Tred`, remove 7 levels from the last row (T7) and three levels from the last but one row (T3) (order does not really matter):
  `P <- Tred(P, rs=5, reduce=7); P <- Tred(P, rs=4, reduce=3).`
  The DHHF now has $k = 352$ remaining columns, and $u_1 = 3$ rows with 11 levels, $u_2 = 1$ row with 8 levels and $u_3 = 1$ row with 4 levels. This step follows the idea of Lemma 3.1 of Colbourn and Torres-Jimenez (2010), where the number of columns can sometimes be kept larger than that stated in the lemma in case of removing levels for more than one row, by exploiting imbalance in level frequencies and removing the rarest levels in each subsequent step.
- Identify the smallest suitable CAs for the three different types of rows in the DHHF, i.e., in this case, $D_1 = \text{CA}(12, 3, 11, 2)$ e.g., from the Paley construction, $D_2 = \text{CA}(12, 3, 8, 2)$ and $D_3 = \text{CA}(8, 3, 4, 2)$ (orthogonal array).
- Ascertain the number of constant rows for the CAs: the paper claims one constant row for D_1 and two constant rows each for D_2 and D_3. D_1 and D_3 are found as stated, whereas D_2 has only been found with one constant row: all subsets of 8 columns from the known 11 column CAs permit only one constant row, CAgen Wagner, Kleine, Simos, Kuhn, and Kacker (2020) yields 16 instead of 12 rows, and all 12 run arrays yielded by JMPPro JMP Statistical Discovery LLC (2024) also permit one constant row only). The paper cites Torres-Jiménez as a source of that array; Torres-Jimenez (n.d.) is no longer accessible and its author does not react to questions. The downloaded CAs from the Torres-Jimenez homepage do not have a suitable array; *hence, the number of rows achievable is one larger than that stated in the Colbourn tables.*
- Arrange the CAs in a list and process them with P to yield the new CA:
  `Dlist <- list(paleyCA(3,11), miscCA(3,8,2), CK_doublingCA(4,2))`
  yields the list, which can then be processed into the CA via
  `DHHF2CA(P, Dlist)`
  The function `DHHF2CA` takes care of making as many rows as possible constant. With the code above, the construction achieves the 49 runs claimed in the Colbourn tables. If one does not take care of using a D_2 that allows two constant rows, e.g., by using
  `Dlist <- list(bestCA(3,11,2), bestCA(3,8,2), bestCA(3,4,2))`
  instead of the above, then D_2 permits only one constant row (instead of two), and the resulting CA does not have 49 runs, as the Colbourn table claims, but 50 runs, which is still very good, of course.

There are some elements of the source entries that likely refer to the construction of Colbourn and Torres-Jimenez (2010) that have not yet been understood / located: **"N-CT"** (is this a third author?), **"Arc"** and **"S"** appear in some of the entries. Such entries have been kept untreated for now (questions asked to various experts remained unanswered).

There are also further source entries starting with "perfect hash family" that follow the exact same construction, except for the construction of the PH(H)F that has not been figured out yet. Once the PHF with the stated numbers of rows and columns have been identified, these can be implemented in the same way as described above.

7.1.2 CZ constructions (not implemented)

The CZ constructions by Colbourn and Zhou (2012) use a so-called Φ-Separated Heterogeneous Hash Family P (Φ-SHHF, $M \times k$) instead of a PHHF or DHHf, and so-called $\mathbb{S}$-quilted arrays $\mathbb{S}$-QA(R_i, t, k_i, v) instead of covering arrays. In this context, $\mathbb{S}$ is a set of multisets, of cardinality t each, whose corresponding interactions have to be covered in the $\mathbb{S}$-QA; Φ is a function from $\mathbb{S}$ into the power set of the row indices of P such that $S \in \mathbb{S}$ is separated by all rows in $\Phi(S)$, Ψ_i is the inverse function that allocates the set of

S for which $i \in \Phi(S)$. Using a pair of corresponding P and $\mathbb{S}$-QAs yields a CA whose number of rows is the sum of all run sizes R_i of the $\mathbb{S}$-QA(R_i, t, k_i, v) corresponding to rows $i = 1, \ldots, M$ of P. The method is exemplified in the paper for $v = 2, 3$ and occurs in the Colbourn tables for strengths $t = 3, 4, 5, 6$ and $v = 2, 3, 6$, both for small and large arrays.

The method has not been implemented, as its ingredients have so far not been found. Colbourn (2014b) might help.

7.2 Composition (`crossCAs`, `compositCA`)

"Composition" refers to building the column-wise cross products of a CA(N, t, k, v) with a CA$(M, t, k, vw$ into a CA$(N \cdot M, t, k, v \cdot w)$. It is implemented in R package **CAs** as `crossCAs`. For a non-prime number of levels, composition might occasionally yield the best possible CA; this is especially true for situations for which the best known construction has not yet been implemented, like for $t = 4$, $k = 12$, $v = 8$, where the best implemented design is a cyclotomy construction with 70088 runs (which would be able to cover 8762 columns). Here, applying composition twice using a CA$(24, 4, 12, 2)$ as the only ingredient yields $24^3 = 13824$ runs, which is much worse than the 7680 from "3-restricted SCPHF RE (CL)" that yields the eCAN, but also much better than what would be doable otherwise. Function `compositCA` of package **CAs** automates finding a composition construction for given t, k and v, where v must not be prime.

7.3 Ordered design based construction (`ODbasedCA`)

The Colbourn tables (Colbourn (n.d.)) hold 12 entries with source "ordered design (CCL)". This construction is for strength $t = 3$ with $k \leq v$ and $q = v - 1$ a prime power. Cohen, Colbourn, and Ling (2003) is the primary source for this construction, which is (more or less) repeated in Colbourn (2004) and Cohen, Colbourn, and Ling (2008). The basic idea is as follows:

- Construct an ordered design (OD) of strength 3 in $v(v-1)(v-2) = (q+1)q(q-1) = q^3 - q$ runs and v columns; the construction is provided in Cohen, Colbourn, and Ling (2003) and implemented in the internal function `OD3` of R package **CAs**. All its rows contain v distinct values, and the design covers all triples with three distinct values.
- Vertically concatenate it with $\binom{v}{2}$ strength 3 designs in two levels, where each such design handles a distinct pair of levels. The CA ingredient for this step is a CA$(N_2, 3, k, 2)$, whose levels are filled in turn with those of a particular value pair.

For every CA, at least one row can be made constant. *If the CA ingredient has a single constant row only*, this row can be removed, so that the number of added rows is $\binom{v}{2} \cdot (N_2 - 1)$ for covering all triples for which there are fewer than three distinct values. After switching the order of the pair $(0, v-1)$ to $(v-1, 0)$, each level occurs at least once in the role of 0 and in the role of 1 in the CA ingredient, so that it is unnecessary to add any constant row back in. *If the CA ingredient has two constant rows*, these can also be removed, but this possibly removes all triples with three identical elements. Therefore, after applying the construction to the CA ingredients without their constant rows, v constant rows have to be added to complete the final CA.

Equations (4) and (5) show the formulae for the runs sizes for constructions of this type. They are adapted from Cohen, Colbourn, and Ling (2008) to also cover the case $k < v$. The formulae for the run size N for a CA$(N, 3, k, v)$ with $k \leq v$ and $v - 1$ a prime power are stated below for one constant row ((4), including the subtraction of the constant rows) and two constant rows (5), with $q = v - 1$:

$$N = q^3 - q + \binom{v}{2} \cdot (\mathrm{CAN}(3, k, 2) - 1) \tag{4}$$

$$N = q^3 - q + \binom{v}{2} \cdot \mathrm{CAN}(3, k, 2) - (q^2 - 1) \tag{5}$$

Table 9 shows the achieved run sizes as N_{my}. It can be seen that the package achieves the runs sizes from the formulae both for one and for two constant rows in the CA ingredient, whereas the Colbourn tables rely on a single constant row, presumably because the arrays with two rows were not available during construction or were assumed not to be available to later users. The sizes given in the paper are for $k = v$; for two constant rows in the CA ingredient, the paper gives the numbers of the formula. For one constant

Table 9: Run sizes from Colbourn tables (N_{tables}), paper (N_{paper}), and the implementation in package CAs (N_{CAs}). The entries in column N_{paper} are from Cohen, Colbourn, Ling (2003, construction B in Table 12). n_{constant} indicates the number of constant rows of the CA used in my implementation. Columns 'Eq. (2)' and 'Eq. (3)' show calculated numbers CA ingredients with one or two constant rows, respectively.

	t	k	v	N_{tables}	N_{paper}	N_{CAs}	n_{constant}	Eq. (2)	Eq. (3)
added1	3	6	6		276	276	2	285	276
added2	3	8	8		624	624	2	644	624
added3	3	9	9		909	900	1	900	873
added4	3	12	12		2190	2190	2	2244	2190
Colbourn tables 1	3	10	10	1215		1215	1	1215	1180
Colbourn tables 2	3	11	12	2046		2046	1	2046	1992
Colbourn tables 3	3	11	14	3185		3185	1	3185	3108
Colbourn tables 4	3	12	14	3458		3381	2	3458	3381
Colbourn tables 5	3	14	14	3549	3654	3549	1	3549	3472
Colbourn tables 6	3	11	18	6579		6579	1	6579	6444
Colbourn tables 7	3	11	20	8930		8930	1	8930	8760
Colbourn tables 8	3	12	20	9500		9330	2	9500	9330
Colbourn tables 9	3	14	20	9690		9690	1	9690	9520
Colbourn tables10	3	16	20	9880		9880	1	9880	9710
Colbourn tables11	3	20	20	10070		10070	1	10070	9900
Colbourn tables12	3	11	24	15180		15180	1	15180	14928

row, its numbers deviate from those of the formula, and might contain a mistake / typo for $v = k = 14$. The construction is open in the sense that it can be created for arbitrary $v = q + 1$ with $k \leq v$.

The implementation of the construction for two constant rows is somewhat lazy in the package **CAs**, in that it does not extract and reintroduce constant rows but simply removes duplicate rows in the end. In all considered cases, this yields the run size of Equation (5)

7.4 Direct product constructions for strength 2 CAs

7.4.1 Simple direct product

The simplest direct product construction combines two strength 2 CAs in v levels of dimensions $N \times k$ (D_1) and $M \times \ell$ (D_2), respectively, into an $(N + M) \times k\ell$ strength 2 CA of v levels, by

- horizontally concatenating ℓ copies of D_1, i.e., calculating $\mathbf{1}_\ell \otimes D_1$ ($N \times k\ell$)
- horizontally concatenating ℓ blocks obtained by repeating the ith column of D_2 k times, $i = 1, \ldots, \ell$, i.e., calculating $D_2 \otimes \mathbf{1}_k$ ($M \times k\ell$),
- and vertically concatenating the two matrices ($(N + M) \times k\ell$).

This construction is implemented in `productCA`, when changing the argument `generalized` to `generalized=FALSE`.

7.4.2 Generalized direct product benefitting from constant rows

The simple direct product can be modified to reduce the number of runs by at least 2 and at most v through making use of constant rows, as stated in Colbourn and Torres-Jimenez (2013), cited in Corollary 1 of Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021). The construction is now first described for two CAs with a single constant row each (which can always be achieved by swapping levels for each column). If D_1 has a constant row at level s_1, and D_2 has a constant row at level s_2, with $s_1 \neq s_2$ (always achievable by swapping levels), the above construction applied to the designs without their constant row yields a CA$(N + M - 2, 2, k\ell, v)$; it is straightforward to see that the missing constant rows are covered in the runs that result from the respective other CA. The generalization to more constant rows is also straightforward:

- If D_1 has s constant runs and D_2 has r constant runs, apply equivalence transformations to achieve a total of $\min(s + r, v)$ constant runs *in distinct levels*; for $s + r > v$, there is more than one choice

of rows with which this can be accomplished.
- Remove these distinct constant rows before applying the construction.
- The result is a CA$(N + M - \min(s + r, v), 2, k\ell, v)$.

This construction is implemented in `productCA` (default). It uses the algorithm searching for constant runs, which may be time-consuming for arrays with large run sizes (search for the largest clique in a graph whose vertices are the rows); the argument `one_is_enough` prevents this search and uses all existing constant rows or makes at most one row constant for each ingoing CA.

7.4.3 Product of two PCAs (function `productPCA`)

The product construction of this section is also called "cut-and-paste" method in the literature (e.g., Colbourn and Torres-Jimenez (2013)). It uses partitioned covering arrays (PCA), as introduced in Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006): for a "PCA", an $N \times (k_1 + k_2)$ CA can be partitioned into four blocks, for which

- a $v \times k_1$ block called P consists of k_1 identical columns, each consisting of the sorted levels,
- a $v \times k_2$ block to the right of P is arbitrary (for a PCA) or is constant (for an SCA),
- the remaining $(N - v) \times k_1$ and $(N - v) \times k_2$ blocks are arbitrary.

PCA and SCA are denoted as PCA$(N, t, (k_1, k_2), v)$ and SCA$(N, t, (k_1, k_2), v)$, with $t = 2$ relevant for the construction. Use of a PCA or SCA in a multiplication construction requires that the constant block is set to 0 in the SCA and that the levels of the right-hand side columns are permuted such that the first row can only hold the first level, the second row can hold any of the first two levels, and so forth. This is always possible.

The Bose construction yields an SCA$(q^2, 2, (k, 1), q)$ for prime power q. For example, Table 4 shows the SCA$(3^2, 2, (3, 1), 3)$ with the rows and columns arranged such that the top left 3×3 block serves as P, the top right 3×1 block fulfills the SCA condition, and the bottom two blocks are arbitrary. Note, for later use, that the bottom left block is again a PCA, with $k_1 = 3$ and $k_2 = 0$.

According to Theorem 3.2 of Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006), the product of a PCA$(N, 2, (k_1, k_2), v)$ with an SCA$(M, 2, (\ell_1, \ell_2), v)$ yields a PCA$(N + M - v, 2, (k_1\ell_1, k_1\ell_2 + k_2\ell_1), v)$, i.e., a PCA in $N + M - v$ runs and $k_1\ell_1 + k_1\ell_2 + k_2\ell_1$ columns. From Figure 4 of the paper, it is obvious that the use of two SCAs in the construction produces an SCA (and not a PCA only); this is also informally stated in Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006). It is, however, not really relevant, as the original requirement for the second array to be an SCA was later dropped (Colbourn and Torres-Jimenez (2013), referencing Colbourn (2011); the construction requires the PCA to be coded such that entries in the ith row of the top right part are at most $i - 1$). In any case, the resulting array can again be used in the construction. In this way, chains of strength 2 CAs can be produced.

For example, the Meagher and Stevens (2005) construction for 10 columns in 7 levels each yields an SCA$(61, 2, (9, 1), 7)$ (optimal), the Bose construction an SCA$(49, 2, (7, 1), 7)$ (optimal), which leads to an SCA$(103, 2, (63, 16), 7)$ (optimal), which can in turn be used for producing an SCA$(145, 2, (441, 175), 7)$ (optimal run size, but not optimal k, which is 625), which can in turn be used for producing an SCA$(187, 2, (3087, 1666), 7)$ (optimal run size but not optimal k), and so forth. For also achieving optimal k, it can sometimes be successful to increase the k_1 in the partitioned result. For example, the 625 columns in 145 runs have presumably arisen from the split (72, 7) instead of (63, 16) for the 103 run SCA, which can presumably be achieved but requires a lengthy search process); this would immediately yield a split into (504, 121) for the 145 run array, yielding an SCA$(187, 2, (3528, 1351), 7)$, which falls short by 49 columns versus the Colbourn table entry; the latter is presumably achievable by optimizing the split of the 145 run array to (553,72), and so forth. Such serial optimizations (and more optimization steps) have been conducted for three levels by Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021).

The construction can also be applied for a PCA with itself. For example, using the Bose SCA for $v = 3$ (SCA$(9, 2, (3, 1), 3)$) combined with itself yields an SCA$(15, 2, (9, 6), 3)$, which is as good as the current entry in the Colbourn tables from Nurmela (2004) tabu search. However, while the CA from the PCA construction achieves up to 15 columns, the tabu search array can accommodate up to 20 columns. For all other prime powers q, however, the PCA-product of the Bose SCA with itself yields the current optimum SCA in $2 \cdot q^2 - q$ runs and $q^2 + 2q$ columns; for all these, the Colbourn table source entry is "PCAx2PCA".

Theorem 3.3 of Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) provides a different number of columns, which can be larger under some conditions: for cases where the bottom left block of the PCA is itself a PCA, as was, e.g., observed for the Bose array in Table 4: If $\ell_1 = \eta + (\ell_1 - \eta)$ (in case of the above Bose SCA, $\eta = \ell_1 = 3$), the construction yields a PCA$(N + M - v, 2, (\eta(k_1 + k_2), (\ell_1 - \eta)(k_1 + k_2) + k_1\ell_2), v)$, i.e., this split is better for the case $\eta(k_1 + k_2) > k_1 * \ell_1 \iff \eta \cdot k_2 > (\ell_1 - \eta) \cdot k_1$. Under this condition, and if one wants to use the resulting PCA in a further construction with a Bose array, it may be worthwhile to attempt an increase in the number of columns for k_1, by choosing different rows of the array for the v-row partition block. An example of this will occur for `recursiveBose`, where $\ell_1 = \eta$.

It seems that this construction is denoted in the Colbourn tables as "PCAx2PCA", whereas the construction "PCAxPCA" might refer to a construction discussed in Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021), which references Calvagna and Gargantini (2012) for the general principle of "coverage inheritence" (not yet digested).

Note that the entry "PCAx2PCA" does not mean that this construction alone is sufficient for obtaining the respective CA referenced in the Colbourn table: It is often the case that a smaller best array from an optimization tool is extended by this method; for example, for $v = 4$, various arrays with source "simulated annealing (Torres-Jimenez)" were apparently extended by applying this method once, and the resulting CAs then have the source entry "PCAx2PCA"; for a CA$(N, 2, k_N, 4)$, the source array D_1 is a PCA$(N - 12, 2, (k_1, k_2), 4) =$ CA$(N - 12, 2, k, 4)$, with $k_N = k_1 \cdot 4 + k_1 \cdot 1 + 4 \cdot k_2 = 4 \cdot k + k_1$ leading to $k_1 = k_N - 4 \cdot k$ (with $k_1 + k_2 = k$ available from the Colbourn table). For actual implementation of the method, D_1 has to be available and has to be brought into the required PCA format in order to implement the construction.

7.5 `recursiveBose` and `recursiveBoseHartman`

A construction described in Hartman (2005) (Theorem 7.1, Corollary 7.2) increases the number of columns for a strength 2 CA A in q levels (q prime power) from k to $kq + 1$, at the expense of an additional $q^2 - q$ rows, by combining it with B_q. This construction was implemented in the R package **CAs** as `recursiveBoseHartman`. In the simplest case, A is the unreplicated OA$(q^2, 2, q + 1, q)$ by a Bose construction (arranged as in R: `lhs::createBose(q,q+1,brandom=FALSE)`), i.e., the resulting array will have $2q^2 - q$ rows and $q^2 + q + 1$ columns. For all relevant cases, this construction yields the current best run size from the Colbourn tables for its attainable number of columns; however, there is another construction with the same number of runs for more columns (see below, implemented as `recursiveBose` in the R package **CAs**), as can be seen in Table 10.

`recursiveBoseHartman` can be applied to a matrix A that also equals B_q, i.e., it can simply produce a combination of d B_q's, which yields a $(dq^2 - (d-1)q) \times (q^d + q^{d-1} + \cdots + q - 1)$ CA. In this case, recursive application of function `productPCA` yields the same number of runs and more columns, i.e., `recursiveBose` with type PCA is strictly superior to `recursiveBoseHartman`: repeated application of function `productPCA` to a total of d B_q's, with increasing the size of the left-hand side PCA parts of the interim result at each step according to the construction of Theorem 3.3 of Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006), yields a $(dq^2 - (d-1)q) \times ((q+1)D(d,q) + qD(d-1,q))$ PCA, where $D(d,q)$ is obtained recursively via $D(0,q) = 0$, $D(1,q) = 1$, $D(d,q) = q(D(d-1,q) + D(d-2,q))$. Theorem 3.3 of Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) can be applied because B_q is a PCA$(q^2, 2, (q, 1), q)$ (and actually an SCA), and its lower left block is itself an SCA again with $\eta = q$ columns (i.e., the entire matrix) for the left-hand side part. The result of repeated application of Theorem 3.3 to B_q is actually stated in Lemma 3.5 of Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006); note that the stated size, and the split of the number of columns into the two parts k_1 and k_2 is based on repeating the construction without optimizing the interim split, and subsequently optimizing the split in the final result. The R package **CAs** improves the split also for interim results, which improves the number of columns for larger d.

Application of Theorem 3.3 of Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) to any PCA A, and taking the last q rows of the product matrix for forming the first q rows of the new PCA, yields the following recursions for k_{next} and $k_{1,\text{next}}$, starting with $k_{\text{start}} = k_0$ and $k_{1,\text{start}+1} = k_{1;0}$:

$$k_{1,\text{next}} = qk_{\text{prior}}, k_{\text{next}} = k_{1,\text{next}} + k_{1,\text{prior}} = q(k_{\text{prior}} + k_{\text{prior}-1}).$$

This applies to any $N_{\text{prior}} \times k_{\text{prior}}$ PCA A; with B_q as the start matrix, the matrix A_1 has a special

Table 10: Maximum number of columns of the construction with two Bose matrices by Hartman or via (ref:productPCA) (PCA), versus the corresponding empirical CAK entries (eCAK) of the Colbourn tables with their source entry.

q	N	Hartman	PCA	eCAK	Source
2	6	7	8	10	Kleitman and Spencer, or Katona
3	15	13	15	20	tabu search (Nurmela)
4	28	21	24	24	PCAx2PCA
5	45	31	35	35	PCAx2PCA
7	91	57	63	63	PCAx2PCA
8	120	73	80	80	PCAx2PCA
9	153	91	99	99	PCAx2PCA
11	231	133	143	143	PCAx2PCA
13	325	183	195	195	PCAx2PCA
16	496	273	288	288	PCAx2PCA
17	561	307	323	323	PCAx2PCA
19	703	381	399	399	PCAx2PCA
23	1035	553	575	575	PCAx2PCA
25	1225	651	675	675	PCAx2PCA

structure with which one can increase the size of k_1 in subsequent steps by using q rows different from the last q ones as the first q rows of the next matrix. A general formula on how much increase can be obtained in that way has not yet been found, except for the first case: with $d = 3$ B_q's involved, $k_{1,\text{next}}$ can be increased by q. Substantial further improvements are possible in some but not all cases.

Table 10 compares the run sizes of the Hartman recursion with those of the PCA recursion for only two B_q's. It shows that the PCA recursion yields the optimum, except for $q = 2$ and $q = 3$. For all those cases, the source quoted in the Colbourn tables is "PCAx2PCA", corresponding to this construction.

Table 11 shows the comparison for the multiplication of three B_q's. It shows that this recursion still yields optimum CAs for larger q, whereas improvements to the number of columns that can be accommodated in the run size of this construction have been found for smaller values of q (up to $q = 5$). For example, for $q = 3$, the Covering Array Extender construction of Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021) (obtainable in the R package **CAs** via function `CAEX`) involved extending a Bose SCA of $3^2 = 9$ runs in 4 columns successively by further columns up to a PCA$(20, 2, (45, 18), 3)$, with interim steps for $11, \ldots, 19$ runs; the array in 21 runs quoted in Table 11 arises as a `productPCA` result from B_3 with the interim PCA$(15, 2, (14, 6), 3)$, and post-processing the result (which has $3 \cdot 14 + 3 \cdot 6 + 14 \cdot 1 = 74$ columns) by further column extension.

Even though the relative performance of the recursive multiplication of the small B_q gets worse with increasing d, Table 12 shows that it still yields optimal run sizes for some instances of k with $d = 4$.

If one uses function `productCA` instead of `productPCA`, the number of runs is slightly larger than that of either `recursiveHartman` or `recursiveBose` based on `productPCA`, and the number of columns is also larger than that of `recursiveBose` with `productPCA`: `recursiveBose` with type `CA` yields a $(dq^2 - 2d) \times (q + 1)^d$ CA; the number of rows is an upper bound and can be smaller, if more than one row can be made constant in one or both arrays during one or more of the iteration steps. Using PCA type multiplication seems to be more often successful than using CA type multiplication, where possible; but this is not a fixed rule, as can be seen from Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021), where some of the optimum CAs were obtained by using CA type multiplication.

7.6 Systematically adding columns (and rows)

The Colbourn tables contain entries that improve on their preceding row by adding one or more factors; for example, the entry for eCAN(3, 1331, 2) is 55, based on the source "Power CT11^3"; the entry for eCAN(3, 1332, 2) is "Add a factor", which means that a factor is added to the entry for 1331 columns.

Table 11: Maximum number of columns of the construction with three Bose matrices by Hartman or via (ref:productPCA) (PCA), versus the corresponding empirical CAK entries (eCAK) of the Colbourn tables with their source entry.

q	N	Hartman	PCA	eCAK	Source
2	8	15	22	35	Kleitman and Spencer, or Katona
3	21	40	57	93	CA Extender (TJ-AJ-AG)
4	40	85	116	133	simulated annealing (Torres-Jimenez)
5	65	156	205	220	simulated annealing (Torres-Jimenez)
7	133	400	497	497	PCAx2PCA
8	176	585	712	712	PCAx2PCA
9	225	820	981	981	PCAx2PCA
11	341	1464	1705	1705	PCAx2PCA
13	481	2380	2717	2717	PCAx2PCA
16	736	4369	4880	4880	PCAx2PCA
17	833	5220	5797	5797	PCAx2PCA
19	1045	7240	7961	7961	PCAx2PCA
23	1541	12720	13777	13777	PCAx2PCA
25	1825	16276	17525	17525	PCAx2PCA

Table 12: Performance of the recursive Bose PCA construction with d=4 versus the empirical CAN source entries of the Colbourn tables. The 'from' and 'to' columns show values for k.

			run size equal to best CA		source entry better	
q	k	N	from	to	from	to
2	62	10	57	62	63	126
3	219	27	217	219	220	288
4	564	52	504	564	565	652
5	1205	85	898	1205	1206	1289
7	3927	175	1737	3927		
8	6344	232	2431	6344		
9	9729	297	3043	9729		

7.6.1 Add 1 factor (`add1CA`, `add1Column`)

The method behind adding a single factor is from Theorem 3.2 of Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010): When expanding a $\mathrm{CA}(N, t, k, v)$ to $k+1$ columns, the last column of the original CA is repeated, which covers all t-tuples of the new column with $t-1$ of the first $k-1$ columns, and all t-tuples of the last two columns with $t-2$ of the first $k-1$ columns for which the k-th and the $k+1$st column have the same value. Thus, the array needs in addition those t-tuples with $t-2$ of the first $k-1$ columns and distinct values in the k-th and the $k+1$st column. These are accommodated by crossing the best possible $\mathrm{CA}(N_{t-2}, t-2, k-1, v)$ with the $v(v-1) \times 2$ array of distinct pairs for the last two columns, leading to $N_{t-2}v(v-1)$ additional rows as the price for the additional column. For strength 2, the $\mathrm{CA}(N_0, 0, k-1, v)$ can be taken as a row of flexible values, as the only missing pairs are the distinct pairs for the last two columns. In the initial example, the four additional rows are obtained by crossing a CA(2, 1, 1330, 2) (whose columns are all 01) with the 2×2 array that consists of the rows 01 and 10.

For an example with higher strength, the Colbourn tables contain a $\mathrm{CA}(9101, 5, 7014, 3)$ that is obtained from a $\mathrm{CA}(7013, 5, 7013, 3)$ (cyclotomy construction) by adding one factor (source entry "Add 1 factors"). The additional rows are obtained by crossing a CA(348, 3, 7012, 3) from the doubling construction with the $3 \cdot 2$ distinct pairs for the last two columns. (The doubling CA doubles a CA(276, 3, 3584, 3) from the construction "Power CZ3-16.5-14.1", which has so far not been deciphered.) In version 0.21 of the R package **CAs**, the best strength 3 CA for 7012 columns has 365 runs, which leads to a result with 102 more rows than possible with the design from the Colbourn tables, as the excess size 17 of the ingredient CA must be multiplied with 6.

7.6.2 Add more than 1 factor

For strengths 5 and 6, there are also constructions that are based on adding more than one factor at once. For example, the strength 5 CAs for 7015 and 7016 columns are also obtained from the CA$(7013, 5, 7013, 3)$ by adding two or three factors, respectively. This is different from sequentially adding a single column multiple times, and the source for the construction has not yet been found; it may even be search-based.

7.7 Adding a symbol (augmentation)

7.7.1 Construction D (Chateauneuf-Kreher, `CK_constrD`)

The construction uses a CA$(N, 3, k, v-1)$ D_3 (coded from 1 to $v-1$) and a CA$(N, 2, k-1, v-1)$ D_2 (also coded from 1 to $v-1$) and combines them into a CA$(N + kM + k(v-1), 3, k, v)$, by appending to D_3

- a $kM \times k$ matrix made up of the columns of D_2 combined with vectors 0_M
- a $k(v-1) \times k$ matrix made up of the vectors $1 \dots (v-1)$ and 0_{v-1}

in the way shown in Figure Construction D of Chateauneuf and Kreher (2002).

7.7.2 Augmentation by CKRS

Formula (5) of Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010) refers to a greedy form of augmenting each column of a CA with an additional level:

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

The procedure is as follows: Loop over the columns, duplicate the rows with the least frequent level, and replace that level by the new level in those rows. The above formula provides the maximum number of runs that can arise from this approach.

7.7.3 More advanced augmentation

The methods of the previous subsections rarely (if ever) yield competitive run sizes. Colbourn (2014b) describes a more sophisticated approach that is based on quilting arrays obtained from OAs based on focusing on interactions with zero levels. This method deserves being explored and implemented, if possible; this has not yet happened.

7.8 Roux type constructions

7.8.1 Doubling the number of columns (Chateauneuf-Kreher, `CK_doubling`)

Chateauneuf and Kreher (2002) proposed a construction that doubles the number of columns of a strength 3 CA$(N, 3, k, v)$ D_3 by adding $M(v-1)$ rows obtained from a strength 2 CA$(M, 2, k, v)$ D_2, as follows:

- Horizontally concatenate two unmodified copies of D_3 ($N \times 2k$), i.e., use $\mathbf{1}_2 \otimes D_3$.
- Horizontally concatenate $v-1$ vertically concatenated unmodified copies of D_2 with $v-1$ copies of D_2 to which the group action "addition modulo v" has been applied for all elements of the cyclic group $\mathbb{Z}_v$ ($M(v-1) \times 2k$).
- Vertically concatenate both results.

Chateauneuf and Kreher (2002)'s Theorem 4.1 holds the construction for $v = 2$ provided earlier, and their Theorem 4.5 extends it to v levels. The construction is implemented in function `CK_doublingCA` of R package CAs; the underlying workhorse function is `CK_doubling`.

For example, an optimal CA$(64, 3, 6, 4)$ as D_3 (see Section 3.2) can be extended to a CA$(121, 3, 12, 4)$ by using as D_2 a CA$(19, 2, 6, 4)$ obtained from the Meagher and Stevens (2005) construction implemented in function `CS_MS` of R package CAs (see Section 6.3.1). This CA is one run worse than the optimal one, whose construction in the Colbourn tables is given as "Torres Jimenez" (source not identified).

7.8.2 Martirosyan and van Trung (2004) (`MTVTRouxtypeCA`)

Martirosyan and Trung (2004) introduced recursive constructions for strengths 4 and 5, and even general t (their Section 4). They provide detailed how the arrays are to be combined; these are not repeated here. They are implemented in internal workhorse functions of the R package **CAs**. However, the usefulness of this implementation is currently limited, as such constructions are typically competitive for very large situations only, and the package **CAs** is not well-equipped for very large arrays. (Naïve use of the function may even crash R.)

7.8.3 Strength 3 CAs based on Theorem 3 of Cohen, Colbourn and Ling (2008) (`cclCA`)

Theorem 3 of Cohen, Colbourn, and Ling (2008) allows to multiply the number of colums of a CA$(N, 3, \ell, v)$ (A) by k based on a difference covering array D with k columns and R rows and the additional ingredients CA$(M, 2, k, v)$ (B) and CA$(Q, 2, \ell, v)$ (C); the result is a CA$(N + M + QR, 3, k\ell, v)$. The construction juxtaposes the three matrices $E_1 = \mathbf{1}_k \otimes A$ $(N \times k\ell)$ with $E_2 = B \otimes \mathbf{1}_\ell$ and $E_3 = (\mathbf{1}_{R\times k} \otimes C + D \otimes \mathbf{1}_{Q\times\ell})$ mod v $(QR \times k\ell)$; where the DCA has a constant row of zeroes, that row can be omitted without deteriorating coverage. Cohen, Colbourn, and Ling (2008) provided tables of feasible DCA sizes, obtained by simulated annealing, but not the actual DCAs. *As soon as these become available, the construction can be implemented.*

7.8.4 Further Roux-type constructions for strengths 3 and 4

Colbourn, Martirosyan, Van Trung, and Walker (2006) provided further Roux-type constructions for strengths 3 and 4. Their work has not yet been implemented, although it does yield some interesting CAs.

7.9 Covering array extender (CAEX, `CAEX`)

Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021) propose a construction for extending the number of columns of a CA. They obtain the resulting CAs for $v = 3$ and also provide construction details within the paper. The 27 of these arrays, which are optimal according to the Colbourn tables (Nov 2024), were available as actual CAs via download from the Torres-Jiménez website on Feb 6, 2025 (and are now gone). The construction contains a random element and is thus not purely algebraic. It is nevertheless implemented in the R package **CAs** as `CAEX`, and is explained in the following, using the steps outlined in Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021):

1. *Construct* an optimal CA (for prime v, e.g., with the Bose construction; for non-prime v, a latin square will do for strength 2).
2. *Manage* a database Γ that holds all available CAs with their genealogy. It has an *initializer* (t, v, maximum N ($N_{\max}$), maximum k ($k_{\max}$), initial constructed CA in standard format *here I deviate from Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021), who includes this in the* updater) and an *updater*.
3. The *updater* is the workhorse. It conducts design creation and standardisation, including identification of flexible values (also known as redundant elements or don't care values) and maximisation of the k_1 portion of the PCA representation of the array. Furthermore, it extends the number of columns.
   i. *construct* design from prior via a product algorithm (this step is very vague, basically loops over various possibilities)
   ii. isomorphically *standardize* it, including
      - 0 to $v - 1$ coding
      - identification of a flexible set of values
      - maximisation of constant rows (design $\boldsymbol{\alpha}$, number ρ of constant rows)
      - maximisation of columns with permuted values in the first few rows (i.e., the k_1 of a PCA; design $\boldsymbol{\beta}$, number κ of k_1 columns)
   iii. *extend* it by adding columns (*AddFactor*), adding rows (*AddRow*), and filling up missing pairs that arise for added columns (*FillerCA*). The addition of columns starts with adding a copy of the column with the most missing entries. For strength 2, the only problematic pair for this column is its originator; if the missing entries suffice for removing that problem, no row needs to be added. The proposed algorithm uses a step based on simulated annealing in the *FillerCA* component.

Implementation of the construction may be worthwhile for extending the number of columns for further situations, whenever more columns are needed. However, according to the paper, the method requires enormous computational effort.

Appendix A: Cluster analysis of source entries

This appendix documents the initial cluster analysis that was undertaken for getting a start into understanding the constructions implemented in the Colbourn tables.

The distance matrix was obtained based on the Levenshtein distance as implemented in the R function `adist` when using it with default settings. Agglomerative hierarchical clustering with the `ward.D2` distance in function `hclust` yielded 111 clusters at cut height 20. The resulting cluster number, merged with the table information, helps to structure the effort of obtaining appropriate references or locate stored CAs, even though the clustering is far from perfect. Appendix B shows a (long!) listing of all 2287 source entries arranged by cluster number. Note that the number of cluster entries and the number of Colbourn table entries that they represent are not necessarily closely linked: for example, the 43 clusters with up to five elements hold 1.88 percent of the 2287 distinct source entries and contain 45.24 percent of the 13641 not necessarily distinct Colbourn table source entries, whereas the 16 clusters with at least 40 elements make up 53.87 percent of the distinct source entries and contain 20.86 percent of the 13641 not necessarily distinct Colbourn table source entries.

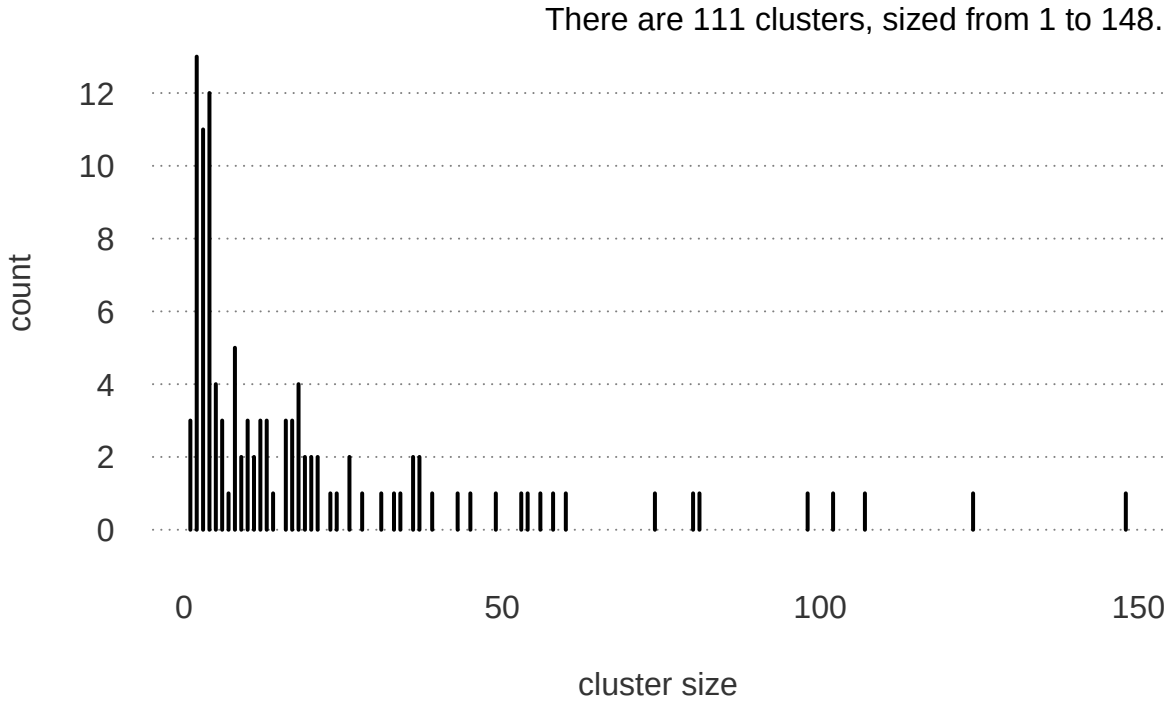

Figure 2: Frequency distribution of cluster sizes

In the following explorations of the clusters, the source entries are explored, understanding their scope and assigning references. The analysis is based on the clusters, which were built solely using the source entries, but nevertheless also differ substantially on their scope regarding strength t, number of levels v and array run size N, as can be seen in Table 13. The detailed cluster content in terms of source entries that are grouped together can be inspected in Appendix B.

Table 13: Property distributions for the clusters. 'size' means the number of Colbourn table entries with a source from this cluster. Labels for N are meant to exclude previous categories, e.g., <=2000 stands for 201 to 2000.

cluster		strength					levels							N			
ID	size	2	3	4	5	6	2	3	4	5	6	7:10	>10	<=200	<=2000	<=20000	>20000
1	824	0	0	262	332	230	0	2	18	11	0	306	487	0	8	63	753
2	497	0	18	135	200	144	0	0	0	1	60	105	331	0	3	17	477
3	134	0	0	44	53	37	0	0	0	0	0	0	134	0	0	0	134
4	56	0	0	17	24	15	0	0	0	0	0	0	56	0	0	0	56
5	4	0	0	0	4	0	0	0	0	0	0	0	4	0	0	0	4
6	581	0	12	19	239	311	5	18	22	19	35	101	381	1	2	20	558
7	7	4	0	0	3	0	0	0	0	0	0	2	5	0	4	0	3
8	567	70	53	174	187	83	0	182	226	0	89	5	65	14	239	271	43

Table 13: Property distributions for the clusters (continued)

cluster		strength					levels							N			
ID	size	2	3	4	5	6	2	3	4	5	6	7:10	>10	<=200	<=2000	<=20000	>20000
9	671	539	9	31	40	52	2	0	29	47	28	190	375	174	370	11	116
10	32	8	15	3	6	0	5	4	0	1	2	3	17	11	10	11	0
11	155	27	57	33	25	13	77	78	0	0	0	0	0	125	30	0	0
12	122	0	122	0	0	0	2	6	1	2	15	10	86	9	20	51	42
13	326	0	326	0	0	0	0	16	2	1	56	20	231	7	43	124	152
14	707	0	28	679	0	0	0	13	0	1	0	95	598	0	7	20	680
15	18	12	6	0	0	0	0	6	0	1	9	2	0	18	0	0	0
16	148	0	12	55	45	36	0	6	18	5	0	59	60	0	12	24	112
17	67	0	1	25	24	17	0	0	0	0	14	13	40	0	0	4	63
18	23	0	0	11	8	4	0	0	0	0	0	0	23	0	0	0	23
19	332	0	2	142	102	86	0	0	1	2	15	84	230	0	1	9	322
20	60	0	0	21	19	20	0	0	0	0	0	1	59	0	0	0	60
21	22	0	13	1	4	4	0	6	2	2	0	6	6	0	9	13	0
22	22	0	2	3	6	11	20	2	0	0	0	0	0	6	16	0	0
23	184	0	26	97	61	0	20	87	12	0	35	15	15	6	64	74	40
24	26	0	0	13	12	1	2	1	0	0	23	0	0	2	0	12	12
25	71	0	20	4	4	43	43	7	5	2	12	2	0	12	51	7	1
26	71	0	0	37	27	7	19	2	0	0	38	3	9	5	16	33	17
27	71	4	8	24	35	0	32	5	3	1	3	4	23	12	32	7	20
28	2118	2118	0	0	0	0	0	0	0	4	65	151	1898	104	2014	0	0
29	2	2	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0
30	139	138	0	0	0	1	0	0	0	0	1	1	137	2	136	0	1
31	8	0	0	2	3	3	0	0	0	0	0	2	6	0	0	0	8
32	92	92	0	0	0	0	0	0	1	2	1	8	80	16	76	0	0
33	29	29	0	0	0	0	0	0	1	0	0	17	11	22	7	0	0
34	4	4	0	0	0	0	0	0	0	0	0	1	3	1	3	0	0
35	15	15	0	0	0	0	15	0	0	0	0	0	0	15	0	0	0
36	527	0	0	0	264	263	0	2	21	15	66	89	334	0	0	0	527
37	99	22	21	16	19	21	3	4	3	5	1	16	67	16	21	20	42
38	207	2	81	74	29	21	0	0	0	0	3	24	180	1	9	83	114
39	46	0	30	10	4	2	0	0	0	0	0	0	46	0	0	24	22
40	6	6	0	0	0	0	0	0	1	2	0	2	1	4	2	0	0
41	47	0	0	0	11	36	0	0	0	1	1	4	41	0	0	0	47
42	31	0	0	0	25	6	0	0	0	0	0	2	29	0	0	0	31
43	20	0	0	0	12	8	0	0	0	0	0	8	12	0	0	0	20
44	60	0	0	0	11	49	0	0	0	14	1	10	35	0	0	0	60
45	26	0	0	0	0	26	0	0	0	17	0	3	6	0	0	0	26
46	10	0	0	0	4	6	0	0	0	0	0	0	10	0	0	0	10
47	7	0	0	0	3	4	0	0	0	0	0	0	7	0	0	0	7
48	2	0	0	0	0	2	0	0	0	0	0	0	2	0	0	0	2
49	8	0	7	0	0	1	0	0	0	1	0	0	7	0	0	6	2
50	16	0	0	0	16	0	0	0	0	0	0	0	16	0	0	0	16
51	36	0	1	0	29	6	0	0	0	6	1	2	27	0	1	0	35
52	23	0	0	0	23	0	0	0	0	0	0	3	20	0	0	0	23
53	34	0	0	0	34	0	0	0	0	0	0	4	30	0	0	0	34
54	34	0	0	0	34	0	0	0	0	0	0	5	29	0	0	0	34
55	14	0	2	0	12	0	0	0	2	2	2	4	4	0	0	2	12
56	102	0	1	0	101	0	0	0	0	0	3	18	81	0	0	1	101
57	39	0	0	0	37	2	0	0	0	2	0	3	34	0	0	0	39
58	12	0	0	0	0	12	0	0	0	0	0	0	12	0	0	0	12
59	23	0	0	0	0	23	0	0	0	1	2	20	0	0	0	0	23
60	33	0	0	0	0	33	0	0	3	0	3	12	15	0	0	0	33
61	31	0	0	0	0	31	0	2	0	4	1	6	18	0	0	0	31
62	22	0	0	0	0	22	0	0	0	0	3	8	11	0	0	0	22
63	8	0	0	0	0	8	0	0	0	0	0	1	7	0	0	0	8
64	5	0	0	5	0	0	0	5	0	0	0	0	0	0	5	0	0
65	179	0	3	1	14	161	14	54	19	0	5	19	68	3	8	4	164
66	26	0	14	2	10	0	26	0	0	0	0	0	0	15	11	0	0
67	54	0	2	0	8	44	6	15	12	0	0	3	18	2	0	4	48
68	52	0	0	0	2	50	2	27	8	0	0	1	14	0	2	0	50
69	49	0	0	0	46	3	0	0	3	0	0	0	46	0	0	0	49
70	58	0	0	0	58	0	0	0	3	0	0	0	55	0	0	0	58
71	39	0	0	0	0	39	0	31	2	0	0	0	6	0	0	0	39
72	14	0	0	0	0	14	0	9	5	0	0	0	0	0	0	0	14
73	33	0	8	11	5	9	21	9	0	0	3	0	0	6	15	8	4
74	9	0	4	2	0	3	3	6	0	0	0	0	0	2	4	1	2
75	28	0	7	12	0	9	21	7	0	0	0	0	0	4	15	9	0
76	13	0	3	0	0	10	10	3	0	0	0	0	0	0	3	10	0
77	31	0	0	0	31	0	0	0	0	0	31	0	0	0	0	0	31
78	18	0	1	0	17	0	0	1	0	0	17	0	0	0	1	0	17
79	56	0	2	0	46	8	8	2	0	0	46	0	0	0	2	8	46
80	13	0	2	0	10	1	1	2	0	0	10	0	0	0	2	1	10
81	14	0	0	0	10	4	4	0	0	0	10	0	0	0	0	4	10
82	47	0	0	0	10	37	0	0	0	4	3	10	30	0	0	0	47
83	64	0	0	0	14	50	0	0	8	0	0	18	38	0	0	0	64
84	165	0	0	0	25	140	0	0	10	0	9	28	118	0	0	0	165

Table 13: Property distributions for the clusters (continued)

cluster		strength					levels							N			
ID	size	2	3	4	5	6	2	3	4	5	6	7:10	>10	<=200	<=2000	<=20000	>20000
85	198	0	0	0	47	151	0	0	6	0	4	53	135	0	0	0	198
86	179	0	0	0	75	104	0	0	0	0	0	73	106	0	0	0	179
87	254	0	0	0	39	215	0	0	6	0	0	71	177	0	0	0	254
88	315	0	0	0	72	243	0	0	9	0	9	75	222	0	0	0	315
89	153	0	0	0	70	83	0	0	4	0	6	40	103	0	0	0	153
90	285	0	0	0	61	224	0	0	1	0	25	80	179	0	0	0	285
91	235	0	0	0	47	188	0	0	2	0	2	48	183	0	0	0	235
92	110	0	0	0	8	102	0	0	1	0	2	17	90	0	0	0	110
93	88	0	0	0	0	88	0	0	0	0	0	3	85	0	0	0	88
94	32	0	32	0	0	0	0	0	1	1	0	5	25	1	5	16	10
95	10	0	10	0	0	0	0	0	0	0	0	0	10	0	0	1	9
96	438	0	438	0	0	0	0	0	32	89	0	195	122	0	142	247	49
97	96	0	96	0	0	0	0	0	0	0	0	31	65	0	0	66	30
98	21	0	21	0	0	0	0	0	0	0	0	0	21	0	0	4	17
99	70	0	70	0	0	0	0	0	49	21	0	0	0	0	70	0	0
100	122	0	16	35	31	40	0	0	5	117	0	0	0	0	21	44	57
101	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0
102	14	0	14	0	0	0	0	0	0	1	0	11	2	0	1	13	0
103	3	0	0	0	3	0	0	0	0	0	1	2	0	0	0	0	3
104	19	0	15	4	0	0	0	0	0	0	2	10	7	0	4	14	1
105	29	0	29	0	0	0	0	0	0	0	0	4	25	0	3	16	10
106	10	0	10	0	0	0	0	0	0	0	0	0	10	0	0	1	9
107	14	0	0	14	0	0	0	0	0	0	6	8	0	0	0	8	6
108	54	11	28	13	1	1	31	12	4	1	2	3	1	48	6	0	0
109	147	146	0	0	1	0	1	0	24	14	23	45	40	95	52	0	0
110	43	0	0	0	0	43	43	0	0	0	0	0	0	0	43	0	0
111	6	5	1	0	0	0	1	2	0	1	2	0	0	6	0	0	0

As there are some source entries that contain numeric specifics for single arrays, some of the clusters are very large; other clusters have few elements that nevertheless cover many instances in the Colbourn tables. Table 13 provides summarizing information.

In the following, actual references are provided for many of the source entries. Note that the references do not necessarily yield to easy array creation, and there will be substantial work involved in making the arrays available even after having the references identified.

- Clusters 1 to 5 relate to "... restricted SCPHF RE ..." constructions, where SCPHF stands for "Sherwood covering perfect hash families" (1515 not necessarily distinct source entries in the Colbourn tables); many entries have one or more occurrences of "fuse". References given are CL for Colbourn and Lanus (2018a), and in a few cases CLS for Colbourn, Lanus, and Sarkar (2018); the constructions are not covered in the R package **CAs**; they have not been prioritized, as almost all of them are large ($N > 20000$ for about 94%) or extremely large ($N > 1000000$ for about 57%).
- Cluster 6 is about adding factors (581 not necessarily distinct source entries in the Colbourn tables); the addition always refers to an obvious preceding row of the table and causes a substantial increase in N. Adding one factor is explained in Theorem 3.2 of Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010). Adding more than one factor occurs for strengths 5 and 6 only; **it could so far not be located** and might also be based on a search construction.
- Cluster 7 is about adding a symbol. A method for this is also explained in Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010) (their Equation (5)); another method for adding one symbol to a strength 3 CA is Construction D of Chateauneuf and Kreher (2002) (but that one is not in the Colbourn tables); according to Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010), neither method is universally better than the other. These constructions do not cover a large number of cases, but seem desirable to implement because of conceptual simplicity.
- Cluster 8 relates to constructions from the Torres-Jimènez group that cover a total of 567 not necessarily distinct source entries; references are "AG-TJ-IM" (Avila-George, Torres-Jimenez, and Izquierdo-Marquez (2018)) for adding a factor with subsequent stochastic optimization (called AFSO by its authors, strength 2 only, non-prime power v), "IM-TJ-AJ-AG" (Izquierdo-Marquez, Torres-Jimenez, Acevedo-Juárez, and Avila-George (2018)) and "TJ-IM" (Torres-Jimenez and Izquierdo-Marquez (2018)) for two variants of creating a CPHF with a 3-stage approach, and "TJ-RC" (Torres-Jimenez and Rodriguez-Cristerna (2017)), for metaheuristic post-optimization of the "NIST Covering Array Tables" (n.d.) arrays (which are based on an IPOG-F search). *Unfortunately, the resulting search-based CAs and CPHFs are nowhere to be found.*

- Cluster 9 is a mixture of various different source entries that have in common that they are short:
  - "Augment OA" (103 source entries) is explained in Colbourn (2014b). It is about increasing the number of levels for a known OA, and it makes use of quilted arrays in the process.
  - "composition" (10 source entries) refers to a product construction of v and w levels into vw levels. Its implementation in function `compositCA` works for general t. All such entries in the Colbourn tables are for strength $t = 6$, but some OAs for lower strength that can be obtained by composition are contained in the Colbourn tables under "Derive from strength 6" or "Derive from strength 5".
  - "PCAx2PCA" and "PCAxPCA" refer to 320 and 219 source entries, resp.; the concept of partitioned CAs (PCA, see also Section 7.4.3) is introduced in Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006); it can be serially applied and produces many good strength 2 CAs. "PCAx2PCA" is short for applying Theorem 3.3 of the paper, in which a particular block of the second PCA has its top left rows as permutations of the levels; this is beneficial in recursive application (e.g., `recBoseCA` or `recursiveBose` in R package **CAs**), because it permits to allocate more columns to the k_1 part of the resulting PCA. "PCAxPCA" is about PCA multiplication without that sub structure (Theorem 3.2 of the paper). Although the construction is understood, its implementation in many cases does not recover the run sizes stated in the Colbourn tables.
  - The other cluster members are author names that are relevant for single-digit source entries each, among them Ji and Yin (2010) and some more cases for which some or all CAs are included in the package.
- Cluster 10 again holds short source entries for relatively few cases, covered by CKRS (Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010)) and further authors.
  - The CKRS arrays are downloadable at http://www.sztaki.hu/~keri/arrays/CA_listings.zip; it seems, however, that only one of the three alleged Steiner constructions is actually found there.
  - Twelve constructions by Cohen, Colbourn, and Ling (2008) (CCL) are based on ordered designs; these are all strength 3 for 10 or more levels. They have been implemented in function `ODbasedCA`.
  - Seven constructions refer to projection (C) constructions, presumably by Colbourn (2008); there are many further of those that were subjected to postop, mostly by the NCK method (see cluster 30); they all have strength 2 and a large number of levels (minimum: 9). The projection construction has been automated for Bose OAs only in function `projBoseCA`.
  - The two remaining constructions are singletons and have not been implemented.
- Cluster 11 holds the CA extender by Torres-Jimenez, Acevedo-Juárez, and Avila-George (2021) (27 entries; very powerful construction for strength 2 with 3 levels, implemented in function `CAEX`), SBSTT (search-based software testing tool) by Torres-Jimenez and Avila-George (2016) (76 entries), tower by Torres-Jimenez, Izquierdo-Marquez, Kacker, and Richard Kuhn (2015) (one entry) and a two-stage simulated annealing algorithm by Torres-Jimenez, Avila-George, and Izquierdo-Marquez (2017) (51 entries). All these are search-based. Except for `CAEX`, whose necessary ingredients were downloadable from Torres-Jimenez (n.d.) in early 2025, the other CAs are unavailable, as the Torres-Jiménez repository does no longer exist (and did no longer hold the arrays before either).
- Cluster 12 holds Chateauneuf-Kreher doubling (Chateauneuf and Kreher (2002)) without (114 entries) and with (5 entries) NCK postprocessing, and Chateauneuf-Kreher NRB (3 entries, Chateauneuf, Colbourn, and Kreher (1999), Chateauneuf and Kreher (2002)). These are available in functions `CK_doublingCA` and `CK_NRB`; the postprocessing is available as well, but may not be powerful enough.
- Cluster 13 likely refers to Cohen, Colbourn, and Ling (2008) (114 entries, plus a few with fuse once or twice; all strength 3). While it was initially thought that it is stated Theorem 3 of that paper and is based on a difference covering array (DCA) and several CAs as ingredients, it is not yet clear whether this is really the case. Thus, the construction was not yet implemented.
- Cluster 14 covers source entries for Roux type constructions according to Colbourn, Martirosyan, Van Trung, and Walker (2006). (not yet implemented)
- Cluster 15 holds the two "cover starter" source entries, combined with names Colbourn and Torres-Jimémez (two different spellings); these were not located, the known cover starter constructions do not yield the respective CAs. Colbourn and Torres-Jimenez (2013) has a section that refers to cover starters, but without providing any explicit constructions.
- Clusters 16 to 18 have CPHF IPO constructions (238 not necessarily distinct source entries), with reference WCS (Wagner, Colbourn, and Simos (2022)). Many of the CPHFs are downloadable at

https://srd.sba-research.org/data/cphfipo/cphfs.zip and have been included into function `cphfCA` of the package **CAs**.

- Clusters 19 and 20 have CPHF and SCPHF constructions, with reference CLS (Colbourn, Lanus, and Sarkar (2018)), and different amounts of "fuse". Some of the CPHFs are available for download at Dwyer (2024).
- Cluster 21 has simulated annealing of CPHF and SCPHF, with reference Torres-Jimenez and Izquierdo-Marquez (2018). This work appears to be unavailable, like most work by this group.
- Cluster 22 has cyclic constructions by Colbourn and Kéri (2009) for $v = 2$ and by Colbourn for $v = 3$ (reference unknown). The binary CAs are implemented in function `CS_CK` (cover starter Colbourn Kéri).
- Cluster 23 has cyclotomy-related constructions by Colbourn (Colbourn (2010)), without or with ("fuse", "mult", "shift") modifications, and a few constructions with "shift" by Avila-George, Torres-Jimenez, and Hernández (2012).
- Cluster 24 has several constructions involving Colbourn (most relevant "density cyclic", presumably from Colbourn (2014a)), all with NCK postprocessing; the arrays with "density" in the source are available at Dwyer (2024), if they are not too large.
- Cluster 25 has Cyclotomy constructions by Torres-Jimènez (i.e., assuming that he is supposed to have checked the conditions for coverage from the cyclotomy construction), and two further source entries that only list the name Torres-Jimènez with or without a hyphen. The latter might refer to his uploaded arrays; for $v = 2$ these were available at Torres-Jimenez (n.d.) in February 2025. Note that the two strength 4 CAs did not quite match the sizes (177 and 178 instead of 173 and 174 runs); of course, this may happen, if an initial lucky search cannot be repeated. The non-cyclotomy CAs for $v > 2$ are unavailable.
- Cluster 26 has "Density Cyclic" (presumably of Colbourn (2014a)) without "postop NCK", some with "Mult" inserted; these are again available at Dwyer (2024), if not too large. There are also "Paley type (Colbourn)" with 19 instances of strength 5 in 2 levels (Colbourn (2015)), and large "extended OA (Colbourn)" (minimum run size 12312, strengths 4 to 6, $v \geq 9$; these are all extensions to a Bush construction, or fusions thereof, somewhat smaller than by the method of `add1CA` (e.g., 12312 instead of 12393 from `add1CA`)), and two-stage Sarkar-Colbourn (Sarkar and Colbourn (2019), *very* large, minimum run size >500000; therefore not pursued).
- Cluster 27 has "Derive" instances. These are provided in the package **CAs** via function `derive`, however not in an automated way. In several cases, where OAs are derived, both the original OA and the derived OA are obtainable via `compositCA`.
- Clusters 28 and 29 refer to direct product constructions, presumably according to a technique introduced in Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) and generalized in Colbourn and Torres-Jimenez (2013), in cluster 29 postprocessed by "TJ-SA". All these CAs are for strength 2, with numbers of levels starting at 5. The frequencies are particularly high where the number of levels is not a prime power. The direct product construction, including the generalized one, is implemented in R package **CAs** in function `productCA` for two specified ingredients, and in functions `recBoseCA` and `dpCA` for specified k and v. Note, however, that the claimed eCANs are in many cases not matched by the implementation of those constructions within the R package **CAs**, and for various of the instances, the current best implemented CA is not from the direct product construction but from the PCA product construction.
- Cluster 30 refers to three projection constructions (Colbourn (2008)) and one MIPOG construction (presumably, Younis and Zamli (2011)), all with "postop NCK" or "TJ-SA" (see cluster 29).
- Cluster 31 has "extended OA (Colbourn) special fuse" with or without "postop NCK" (see also Cluster 26).
- Cluster 32 has "Fix ... symbol", cluster 33 has the same, but with "postop NCK" or "postop TJ-SA"; authors are LCDST (Lobb, Colbourn, Danziger, Stevens, and Torres-Jimenez (2012)), Colbourn, Martirosyan, Mullen, Shasha, Sherwood, and Yucas (2006) and Meagher and Stevens (2005) (with some details only available in Meagher (2005a) or Meagher (2005b)). According to Torres-Jimenez, Izquierdo-Marquez, and Avila-George (2019), these are extensions of the Chateauneuf and Kreher (2002) construction. The constructions are implemented in R package **CAs** in functions `CS_MS` (for Cover Starter Meagher Stevens), `CS_LCDST` and `CS_CMMSSY`.
- Cluster 34 refers to incomplete or holey transversal designs (Colbourn and Torres-Jimenez (2013)); these have not been prioritized, since they yield only few entries ($t = 2$, $v = 10, 14, 22, 22$, $k = 6, 50, 6, 7$).
- Cluster 35 holds the Kleitman and Spencer (1973); Katona (1973) optimal strength 2 CAs for

2-level columns (closed form analytical construction, function `KSK`).

- Cluster 36 refers to constructions by Martirosyan and Van Trung (2008), and "Martirosyan-Colbourn" (Martirosyan, Sosina and Colbourn, Charles (2005)), partly with "postop NCK" or "variant". These have not been automated yet.
- Clusters 37 to 39 refer to constructions based on orthogonal arrays (one by Ji and Yin (2010)), with varying numbers of "fuse" post-processings and partly with further post processing. The OAs from the Bush construction and fusions thereof are available in the package **CAs** (functions `SCA_Busht` or `fuseBushtCA`). Some orthogonal arrays for non-prime-power numbers of levels are available among the miscellaneous CAs (function `miscCA`), others (for $v > 12$) are not available in the package at present.
- Cluster 40 has constructions that involve partitioned covering arrays (PCAs, see also Cluster 9) with post-processing.
- Clusters 41 to 63 deal with "perfect hash family" constructions, specifying some parameters of a PHF (or PHHF) without specifying a construction or a source. There are a total of 643 not necessarily distinct source entries of these types in the Colbourn tables. Only 15 of the CAs have fewer than 100.000 runs, and 90% have more than 1.860.000 runs. It will require some research to find out how to construct each PH(H)F, as there is no place where the ingredient PHFs or their construction rule can be conveniently found. It should suffice to constrain the implementation of this construction to strength 3 CAs (all below 100000 runs) and smaller ones of the other strengths. An example is "perfect hash family3,2906,122T18". This describes a PHF with 3 rows and 2906 columns in 122 levels, for which one row is reduced to 122-18=104 levels, which also reduces the number of columns of the resulting PHHF. The CA construction is the same as for the power constructions of Colbourn and Torres-Jimenez (2010) (see Section 7.1.1), once the PH(H)F has been figured out.
- Cluster 64 has construction PGL with and without postop NCK, according to Maity, Akhtar, Chandrasekharan, and Colbourn (2018). There are two PGL constructions for 72 and 58 3-level columns at strength 4 each, and these can be reduced in run size for fewer columns by Nayeri, Colbourn, and Konjevod (2013) postprocessing. The construction has not yet been implemented.
- Clusters 65 to 72 relate to "Power" constructions whose labels start with CT for Colbourn and Torres-Jimenez (2010) (342 distinct source entries).
- Clusters 73 to 81 relate to "Power" constructions whose labels start with CZ for Colbourn and Zhou (2012) 215 distinct source entries.
- Clusters 82 to 93 relate to "Power" constructions whose labels start with "N-CZ" for (818 distinct source entries); a source for these was not found. All these CAs have at least strength 5. The search was not prioritized, as the smallest eCAN from this construction is 221556 and thus likely not interesting for the target user of the R package.
- Clusters 94 and 95 refer to constructions by Raaphorst, Moura, and Stevens (2014) based on linear feedback shift registers (LFSR), with "fuse" steps added.
- Clusters 96 to 99 refer to restricted covering perfect hash family constructions as discussed in Wagner, Colbourn, and Simos (2022); they are from the same paper as Clusters 16 to 18, but unfortunately not offered for download anywhere.
- Cluster 100 refers to restricted CPHF with simulated annealing according to "TJ-IM" (Torres-Jimenez and Izquierdo-Marquez (2018)). These have not been found anywhere.
- Cluster 101 refers to a single entry according to Hnich, Prestwich, Selensky, and Smith (2006), for which the CA$(40, 3, 7, 3)$ (Figure 5 of the paper) does not live up to the claimed run size 39 in the Colbourn tables. The CA$(39, 3, 7, 3)$ is available from the Dwyer repository (Dwyer (2024)). There likely was a subsequent postprocessing that is not mentioned in the source entry; the postprocessing of Nayeri, Colbourn, and Konjevod (2013) did not yield a run size reduction on the Hnich CA.
- Clusters 102 to 107 refer to unrestricted Sherwood covering perfect hash families (SCPHF). Cluster 102 is based on Sherwood, Martirosyan, and Colbourn (2006), mostly with fuse and "postop NCK". Clusters 103 and 104 are based on Colbourn, Lanus, and Sarkar (2018), clusters 105 and 106 on Torres-Jimenez and Izquierdo-Marquez (2020), whereas cluster 107 refers to tabu search (Walker II and Colbourn (2009)). The method is implemented in function `scphfCA` with SCPHFs from Sherwood, Martirosyan, and Colbourn (2006) and Erin Lanus (as provided in Dwyer (2024)).
- Cluster 108 contains four simulated annealing source entries (Avila-George, Torres-Jimenez, and Hernández (2012) (AG-TJ-H), Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010) (CKRS), Cohen, Colbourn, and Ling (2003); Cohen, Colbourn, and Ling (2008), Torres-Jimenez and Rodriguez-Tello (2012) (TJ-RT)). Most of the latter TJ-RT arrays are available at Dwyer (2024),

the CKRS arrays are available in the package (downloaded from Kéris website); Myra Cohen provided some CAs. The AG-TJ-H arrays are unavailable.

- Cluster 109 holds two further simulated annealing source entries attributed to Torres-Jimènez (147 source entries). Except for the one with $t = 5$ and $v = 2$, which is available in the package, these are unavailable, as Torres-Jimenez (n.d.) no longer exists (and they were not on their when it last existed).
- Cluster 110 holds a single entry for the 43 strength 6 2-level designs by Wagner, Kampel, and Simos (2021) that can be downloaded at https://srd.sba-research.org/data/sipo/ and are available in the package via function `WKS_CAs`.
- Cluster 111 holds three tabu search based entries, referencing Nurmela (2004), Rouse-Lamarre (2009, private communication by e-mail to the authors of Colbourn, Kéri, Rivas Soriano, and Schlage-Puchta (2010)) and Zekaoui (2006). Except for the CA$(42, 2, 8, 6)$, all of them can be made with the package **CAs**, though in most cases from a different construction. The CA$(42, 2, 8, 6)$ was not found, the setting currently needs 46 runs from fusing a Bose CA for 7 levels.

Appendix B: The clustered source entries

Table 14: The source entries arranged by clusters

	clusterId	frequency
2-Restricted SCPHF RE (CL)	1	153
2,2-Restricted SCPHF RE (CL)	1	47
3-Restricted SCPHF RE (CL)	1	176
3,2-restricted SCPHF RE (CL)	1	6
3,2-Restricted SCPHF RE (CL)	1	45
3,3-Restricted SCPHF RE (CL)	1	41
3,3,3-Restricted SCPHF RE (CL)	1	11
4-Restricted SCPHF RE (CL)	1	139
4,2-restricted SCPHF RE (CL)	1	2
4,3-restricted SCPHF RE (CL)	1	8
4,4-Restricted SCPHF RE (CL)	1	61
4,4,4-Restricted SCPHF RE (CL)	1	31
5-Restricted SCPHF RE (CL)	1	52
5,5-Restricted SCPHF RE (CL)	1	37
5,5,5-Restricted SCPHF RE (CL)	1	14
5+-Restricted SCPHF RE (CL)	1	1
2-Restricted SCPHF RE (CL) fuse	2	94
2,2-Restricted SCPHF RE (CL) fuse	2	23
3-Restricted SCPHF RE (CL) fuse	2	105
3,2-restricted SCPHF RE (CL) fuse	2	3
3,2-Restricted SCPHF RE (CL) fuse	2	19
3,3-Restricted SCPHF RE (CL) fuse	2	21
3,3,3-Restricted SCPHF RE (CL) fuse	2	7
4-Restricted SCPHF RE (CL) fuse	2	82
4,3-restricted SCPHF RE (CL) fuse	2	2
4,4-Restricted SCPHF RE (CL) fuse	2	38
4,4,4-Restricted SCPHF RE (CL) fuse	2	21
5-Restricted SCPHF RE (CL) fuse	2	30
5,5-Restricted SCPHF RE (CL) fuse	2	24
5,5,5-Restricted SCPHF RE (CL) fuse	2	10
Path-Restricted SCPHF RE (CLS)	2	16
Path-Restricted SCPHF RE (CLS) fuse	2	2
2-Restricted SCPHF RE (CL) fuse fuse	3	30
2,2-Restricted SCPHF RE (CL) fuse fuse	3	7
3-Restricted SCPHF RE (CL) fuse fuse	3	27
3,2-Restricted SCPHF RE (CL) fuse fuse	3	6
3,3-Restricted SCPHF RE (CL) fuse fuse	3	6
3,3,3-Restricted SCPHF RE (CL) fuse fuse	3	3
4-Restricted SCPHF RE (CL) fuse fuse	3	21
4,4-Restricted SCPHF RE (CL) fuse fuse	3	9
4,4,4-Restricted SCPHF RE (CL) fuse fuse	3	7
5-Restricted SCPHF RE (CL) fuse fuse	3	8
5,5-Restricted SCPHF RE (CL) fuse fuse	3	7
5,5,5-Restricted SCPHF RE (CL) fuse fuse	3	3
2-Restricted SCPHF RE (CL) fuse fuse fuse	4	14
2,2-Restricted SCPHF RE (CL) fuse fuse fuse	4	3
3-Restricted SCPHF RE (CL) fuse fuse fuse	4	11
3,2-Restricted SCPHF RE (CL) fuse fuse fuse	4	2
3,3-Restricted SCPHF RE (CL) fuse fuse fuse	4	2
3,3,3-Restricted SCPHF RE (CL) fuse fuse fuse	4	1
4-Restricted SCPHF RE (CL) fuse fuse fuse	4	9

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
4,4-Restricted SCPHF RE (CL) fuse fuse fuse	4	4
4,4,4-Restricted SCPHF RE (CL) fuse fuse fuse	4	3
5-Restricted SCPHF RE (CL) fuse fuse fuse	4	3
5,5-Restricted SCPHF RE (CL) fuse fuse fuse	4	3
5,5,5-Restricted SCPHF RE (CL) fuse fuse fuse	4	1
2-Restricted SCPHF RE (CL) fuse fuse fuse fuse	5	1
2-Restricted SCPHF RE (CL) fuse fuse fuse fuse fuse	5	1
3-Restricted SCPHF RE (CL) fuse fuse fuse fuse	5	1
3-Restricted SCPHF RE (CL) fuse fuse fuse fuse fuse	5	1
Add 1 factors	6	133
Add 11 factors	6	17
Add 13 factors	6	34
Add 16 factors	6	6
Add 17 factors	6	10
Add 19 factors	6	17
Add 2 factors	6	76
Add 23 factors	6	32
Add 25 factors	6	18
Add 25 factors fuse	6	2
Add 3 factors	6	62
Add 4 factors	6	37
Add 5 factors	6	47
Add 6 factors	6	19
Add 7 factors	6	13
Add 8 factors	6	15
Add 9 factors	6	12
Add a factor	6	31
Add a symbol	7	1
Add a symbol (binary)	7	4
Add a symbol postop NCK	7	2
Add Factor SO (AG-TJ-IM)	8	70
CPHF 3-stage (IM-TJ-AJ-AG)	8	226
CPHF 3-stage (TJ-IM)	8	180
IPOGF postop MPO (TJ-RC)	8	91
Augment OA	9	103
composition	9	10
Ji-Li-Yin	9	6
Ji-Yin	9	4
Johnson-Entringer	9	1
Li-Ji-Yin	9	4
PCAx2PCA	9	320
PCAxPCA	9	219
Sloane	9	1
Xu-Ji-Dong	9	3
Brick by Brick (TATA)	10	1
cross-sum (CKRS)	10	8
ordered design (CCL)	10	12
projection (C)	10	7
Special (Yan Jun)	10	1
Steiner system (CKRS)	10	3
CA Extender (TJ-AJ-AG)	11	27
SBSTT (TJ-AG)	11	76

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
tower (TJ-IM-K-K)	11	1
two-stage SA (TJ-AG-I)	11	51
Chateauneuf-Kreher doubling	12	114
Chateauneuf-Kreher doubling postop NCK	12	5
Chateauneuf-Kreher NRB	12	3
Cohen-Colbourn-Ling	13	318
Cohen-Colbourn-Ling fuse	13	7
Cohen-Colbourn-Ling fuse fuse	13	1
Colbourn-Martirosyan-TVT-Walker	14	611
Colbourn-Martirosyan-TVT-Walker fuse	14	80
Colbourn-Martirosyan-TVT-Walker fuse fuse	14	15
Colbourn-Martirosyan-TVT-Walker fuse fuse fuse	14	1
Cover starter (Colbourn, Torres-Jimenez)	15	6
cover starter (Colbourn,Torres-Jimenez)	15	12
CPHF IPO 10 (WCS)	16	4
CPHF IPO 11 (WCS)	16	4
CPHF IPO 13 (WCS)	16	3
CPHF IPO 14 (WCS)	16	2
CPHF IPO 19 (WCS)	16	1
CPHF IPO 2 (WCS)	16	3
CPHF IPO 20 (WCS)	16	1
CPHF IPO 21 (WCS)	16	1
CPHF IPO 22 (WCS)	16	1
CPHF IPO 23 (WCS)	16	1
CPHF IPO 24 (WCS)	16	1
CPHF IPO 3 (WCS)	16	26
CPHF IPO 4 (WCS)	16	28
CPHF IPO 5 (WCS)	16	23
CPHF IPO 6 (WCS)	16	20
CPHF IPO 7 (WCS)	16	16
CPHF IPO 8 (WCS)	16	8
CPHF IPO 9 (WCS)	16	5
CPHF IPO 2 (WCS) fuse	17	2
CPHF IPO 3 (WCS) fuse	17	20
CPHF IPO 4 (WCS) fuse	17	17
CPHF IPO 5 (WCS) fuse	17	13
CPHF IPO 6 (WCS) fuse	17	9
CPHF IPO 7 (WCS) fuse	17	3
CPHF IPO 8 (WCS) fuse	17	2
CPHF IPO 9 (WCS) fuse	17	1
CPHF IPO 2 (WCS) fuse fuse	18	1
CPHF IPO 3 (WCS) fuse fuse	18	7
CPHF IPO 3 (WCS) fuse fuse fuse	18	3
CPHF IPO 4 (WCS) fuse fuse	18	5
CPHF IPO 4 (WCS) fuse fuse fuse	18	2
CPHF IPO 5 (WCS) fuse fuse	18	3
CPHF IPO 5 (WCS) fuse fuse fuse	18	1
CPHF IPO 6 (WCS) fuse fuse	18	1
CPHF Random Extension (CLS)	19	51
CPHF Random Extension (CLS) fuse	19	35
SCPHF Random Extension (CLS)	19	151

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
SCPHF Random Extension (CLS) fuse	19	95
CPHF Random Extension (CLS) fuse fuse	20	8
CPHF Random Extension (CLS) fuse fuse fuse	20	4
SCPHF Random Extension (CLS) fuse fuse	20	31
SCPHF Random Extension (CLS) fuse fuse fuse	20	15
SCPHF Random Extension (CLS) fuse fuse fuse fuse	20	1
SCPHF Random Extension (CLS) fuse fuse fuse fuse fuse	20	1
CPHF Sim Annealing (TJ-IM)	21	9
CPHF Sim Annealing (TJ-IM) fuse	21	2
CPHF Sim Annealing (TJ-IM) fuse fuse	21	1
SCPHF Sim Annealing (TJ-IM)	21	10
Cyclic (Colbourn-Keri)	22	15
Cyclic (Colbourn)	22	2
Cyclic, derived (Colbourn-Keri)	22	5
Cyclotomy (Colbourn)	23	169
Cyclotomy (Colbourn) fuse	23	1
Cyclotomy mult (Colbourn)	23	1
Cyclotomy shift (AG+TJ+H)	23	3
Cyclotomy shift (Colbourn)	23	10
Cyclotomy (Colbourn) postop NCK	24	1
Density (Bryce-Colbourn) postop NCK	24	3
Density (Colbourn) postop NCK	24	1
Density Cyclic (Colbourn) postop NCK	24	20
Hadamard (Colbourn-Keri) postop NCK	24	1
Cyclotomy (Torres-Jimenez)	25	44
Torres-Jimenez	25	11
Torres Jimenez	25	16
Density Cyclic (Colbourn)	26	36
Density Mult Cyclic (Colbourn)	26	2
extended OA (Colbourn)	26	12
Paley type (Colbourn)	26	19
Two-stage (Sarkar-Colbourn)	26	2
Derive from strength 3	27	4
Derive from strength 4	27	8
Derive from strength 5	27	24
Derive from strength 6	27	35
Direct product	28	545
Direct product generalized	28	1573
Direct product generalized postop TJ-SA	29	1
Direct product postop TJ-SA	29	1
double proj (C) postop NCK	30	1
MIPOG (Younis) postop NCK	30	1
projection (C) postop NCK	30	119
projection (C) postop TJ-SA	30	18
extended OA (Colbourn) special fuse	31	7
extended OA (Colbourn) special fuse postop NCK	31	1
Fix 1 symbols (Colbourn)	32	7
Fix 1 symbols (LCDST)	32	13
Fix 1 symbols (Meagher-Stevens)	32	15

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Fix 3 symbols (LCDST)	32	2
Fix 4 symbols (LCDST)	32	15
Fix 5 symbols (LCDST)	32	21
Fix 6 symbols (LCDST)	32	10
Fix 7 symbols (LCDST)	32	9
Fix 2 symbols (LCDST) postop NCK	33	9
Fix 2 symbols (LCDST) postop TJ-SA	33	5
Fix 3 symbols (LCDST) postop TJ-SA	33	11
Fix 4 symbols (LCDST) postop TJ-SA	33	2
Fix 5 symbols (LCDST) postop TJ-SA	33	2
Holey Transversal Design	34	2
Incomplete Transversal Design	34	2
Kleitman and Spencer, or Katona	35	15
Martirosyan-Colbourn	36	155
Martirosyan-TVT	36	206
Martirosyan-TVT postop NCK	36	6
Martirosyan-TVT variant	36	160
orthogonal array	37	73
orthogonal array (Ji-Yin)	37	5
orthogonal array fuse	37	21
orthogonal array fuse fuse	38	17
orthogonal array fuse fuse fuse	38	11
orthogonal array fuse fuse fuse fuse	38	11
orthogonal array fuse fuse fuse fuse fuse	38	9
orthogonal array fuse fuse fuse postop NCK	38	21
orthogonal array fuse fuse postop NCK	38	46
orthogonal array fuse postop NCK	38	90
orthogonal array fuse postop TJ-SA	38	2
orthogonal array fuse fuse fuse fuse fuse fuse	39	4
orthogonal array fuse fuse fuse fuse fuse fuse fuse	39	3
orthogonal array fuse fuse fuse fuse fuse fuse fuse postop NCK	39	4
orthogonal array fuse fuse fuse fuse fuse fuse postop NCK	39	3
orthogonal array fuse fuse fuse fuse fuse postop NCK	39	14
orthogonal array fuse fuse fuse fuse postop NCK	39	18
PCAx2PCA postop TJ-SA	40	1
PCAxPCA postop TJ-SA	40	5
perfect hash family13,10609,103T10	41	2
perfect hash family13,10609,103T14	41	3
perfect hash family13,10609,103T16	41	4
perfect hash family13,10609,103T18	41	4
perfect hash family13,10609,103T21	41	3
perfect hash family13,10609,103T25	41	4
perfect hash family13,10609,103T37	41	3
perfect hash family13,10609,103T39	41	3
perfect hash family13,10609,103T4	41	1
perfect hash family13,10609,103T5	41	4
perfect hash family13,10609,103T7	41	1
perfect hash family13,11449,107T11	41	1
perfect hash family13,11449,107T15	41	1
perfect hash family13,11449,107T41	41	1
perfect hash family13,11449,107T42	41	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family9,10609,103T11	41	3
perfect hash family9,10609,103T30	41	3
perfect hash family9,10609,103T37	41	3
perfect hash family9,11449,107T10	41	1
perfect hash family9,11449,107T12	41	1
perfect hash family13,11881,109T33	42	2
perfect hash family13,11881,109T35	42	2
perfect hash family13,11881,109T39	42	2
perfect hash family9,11881,109T16	42	1
perfect hash family9,11881,109T17	42	2
perfect hash family9,11881,109T20	42	2
perfect hash family9,11881,109T21	42	2
perfect hash family9,11881,109T25	42	2
perfect hash family9,11881,109T29	42	1
perfect hash family9,11881,109T34	42	2
perfect hash family9,11881,109T36	42	3
perfect hash family9,11881,109T47	42	2
perfect hash family9,11881,109T54	42	2
perfect hash family9,11881,109T56	42	2
perfect hash family9,11881,109T57	42	2
perfect hash family9,11881,109T58	42	2
perfect hash family13,12769,113T26	43	4
perfect hash family13,12769,113T28	43	4
perfect hash family9,12769,113T22	43	1
perfect hash family9,12769,113T24	43	2
perfect hash family9,12769,113T25	43	1
perfect hash family9,12769,113T31	43	1
perfect hash family9,12769,113T35	43	1
perfect hash family9,12769,113T36	43	1
perfect hash family9,12769,113T40	43	4
perfect hash family9,12769,113T43	43	1
perfect hash family13,5329,73T14	44	1
perfect hash family13,5329,73T21	44	1
perfect hash family13,5329,73T23	44	1
perfect hash family13,5329,73T9	44	1
perfect hash family13,5331,75	44	1
perfect hash family13,5333,77	44	1
perfect hash family13,5334,78	44	5
perfect hash family13,6241,79T4	44	1
perfect hash family13,6244,82	44	1
perfect hash family13,6890,84	44	1
perfect hash family13,6893,87	44	4
perfect hash family13,6894,88	44	2
perfect hash family13,7921,89	44	1
perfect hash family13,7921,89T5	44	1
perfect hash family13,7921,89T7	44	1
perfect hash family13,7925,93	44	2
perfect hash family13,7926,94	44	1
perfect hash family13,9410,98	44	1
perfect hash family14,2212,50	44	1
perfect hash family14,2213,51	44	4
perfect hash family14,2214,52	44	4
perfect hash family14,3725,65	44	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family14,3726,66	44	4
perfect hash family15,965,35	44	2
perfect hash family15,966,36	44	2
perfect hash family16,289,17,c	44	1
perfect hash family16,361,19,c	44	1
perfect hash family16,529,23,c	44	1
perfect hash family16,625,25,c	44	1
perfect hash family3,849,567	44	2
perfect hash family3,850,568	44	1
perfect hash family3,871,582	44	3
perfect hash family9,2213,51	44	2
perfect hash family9,5333,77	44	1
perfect hash family9,7925,93	44	1
perfect hash family9,7927,95	44	1
perfect hash family13,6889,83S2	45	1
perfect hash family13,6889,83S3	45	1
perfect hash family13,6889,83S4	45	1
perfect hash family13,6889,83S5	45	1
perfect hash family13,6889,83S6	45	1
perfect hash family13,6889,83S7	45	1
perfect hash family13,6889,83S8	45	1
perfect hash family13,6889,83T1	45	1
perfect hash family13,9409,97T13	45	1
perfect hash family13,9409,97T15	45	1
perfect hash family13,9409,97T8	45	1
perfect hash family14,1107,40S11	45	3
perfect hash family14,1107,40S13	45	3
perfect hash family14,1107,40S14	45	3
perfect hash family14,2209,47S2	45	1
perfect hash family14,2209,47T1	45	1
perfect hash family14,3481,59	45	1
perfect hash family14,3481,59T13	45	1
perfect hash family14,3481,59T7	45	1
perfect hash family14,3481,59T9	45	1
perfect hash family16,361,19,c fuse	46	1
perfect hash family16,529,23,c fuse	46	1
perfect hash family16,625,25,c fuse	46	1
perfect hash family20,2116,23,c fuse	46	1
perfect hash family20,2500,25,c fuse	46	1
perfect hash family21,12167,23,c	46	1
perfect hash family21,12167,23,c fuse	46	1
perfect hash family32,512,16,c fuse	46	1
perfect hash family32,512,16,c fuse fuse	46	1
perfect hash family32,515,19,c fuse	46	1
perfect hash family16,529,23,c fuse fuse	47	1
perfect hash family16,529,23,c fuse fuse fuse	47	1
perfect hash family16,625,25,c fuse fuse	47	1
perfect hash family16,625,25,c fuse fuse fuse	47	1
perfect hash family20,2116,23,c fuse fuse	47	1
perfect hash family20,2116,23,c fuse fuse fuse	47	1
perfect hash family21,12167,23,c fuse fuse	47	1
perfect hash family16,625,25,c fuse fuse fuse fuse	48	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family16,625,25,c fuse fuse fuse fuse fuse	48	1
perfect hash family2,34,18T12	49	1
perfect hash family2,364,183T177	49	1
perfect hash family2,760,381T361	49	1
perfect hash family2,760,381T363	49	1
perfect hash family2,760,381T375	49	1
perfect hash family2,760,381T376	49	1
perfect hash family2,760,381T377	49	1
perfect hash family4,884,547T137	49	1
perfect hash family20,2116,23,c	50	1
perfect hash family20,2116,23T10	50	1
perfect hash family20,2116,23T3	50	1
perfect hash family20,2116,23T4	50	1
perfect hash family20,2116,23T5	50	1
perfect hash family20,2116,23T6	50	1
perfect hash family20,2116,23T7	50	1
perfect hash family20,2116,23T8	50	1
perfect hash family20,2116,23T9	50	1
perfect hash family20,2117,24	50	4
perfect hash family20,2500,25,c	50	1
perfect hash family20,2501,26	50	2
perfect hash family3,1005,671	51	2
perfect hash family3,1005,671T74	51	2
perfect hash family3,1018,680	51	1
perfect hash family3,1033,690	51	1
perfect hash family3,1075,718	51	1
perfect hash family3,1080,721	51	1
perfect hash family3,1080,721T101	51	1
perfect hash family3,1080,721T3	51	1
perfect hash family3,1080,721T58	51	1
perfect hash family3,1083,723	51	2
perfect hash family3,1083,723T52	51	2
perfect hash family3,1093,730	51	1
perfect hash family3,1123,750	51	2
perfect hash family3,1144,764	51	2
perfect hash family3,1147,766	51	2
perfect hash family3,1149,767	51	2
perfect hash family3,1225,818	51	1
perfect hash family3,1255,838	51	2
perfect hash family3,1275,851	51	2
perfect hash family3,1855,104	51	1
perfect hash family4,1034,640	51	1
perfect hash family4,1034,640T90	51	1
perfect hash family4,1168,720	51	1
perfect hash family4,1168,720T80	51	1
perfect hash family5,2000,720	51	1
perfect hash family5,2000,720T80	51	1
perfect hash family3,1017,679T146	52	1
perfect hash family3,1017,679T150	52	1
perfect hash family3,1017,679T155	52	1
perfect hash family3,1017,679T65	52	1
perfect hash family3,1032,689T121	52	1
perfect hash family3,1032,689T143	52	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family3,1032,689T147	52	1
perfect hash family3,1092,729T196	52	1
perfect hash family3,1092,729T200	52	1
perfect hash family3,1092,729T205	52	1
perfect hash family3,1092,729T49	52	1
perfect hash family3,1122,749T226	52	2
perfect hash family3,1122,749T295	52	2
perfect hash family3,1122,749T297	52	2
perfect hash family3,1122,749T299	52	2
perfect hash family3,1122,749T307	52	2
perfect hash family3,1122,749T96	52	2
perfect hash family3,1074,717T54	53	1
perfect hash family3,1074,717T97	53	1
perfect hash family3,1224,817T249	53	1
perfect hash family3,1224,817T271	53	1
perfect hash family3,1224,817T275	53	1
perfect hash family3,1224,817T62	53	1
perfect hash family3,1239,827	53	1
perfect hash family3,1239,827T294	53	1
perfect hash family3,1239,827T298	53	1
perfect hash family3,1239,827T303	53	1
perfect hash family3,1239,827T322	53	1
perfect hash family3,1239,827T97	53	1
perfect hash family3,1254,837T114	53	2
perfect hash family3,1254,837T166	53	2
perfect hash family3,1254,837T364	53	2
perfect hash family3,1254,837T367	53	2
perfect hash family3,1254,837T371	53	2
perfect hash family3,1254,837T376	53	2
perfect hash family3,1275,851T128	53	2
perfect hash family3,1275,851T13	53	2
perfect hash family3,1275,851T180	53	2
perfect hash family3,1275,851T378	53	2
perfect hash family3,1275,851T381	53	2
perfect hash family3,1131,755	54	1
perfect hash family3,1131,755T187	54	1
perfect hash family3,1131,755T209	54	1
perfect hash family3,1131,755T213	54	1
perfect hash family3,1131,755T65	54	1
perfect hash family3,1143,763T110	54	2
perfect hash family3,1143,763T13	54	2
perfect hash family3,1143,763T240	54	2
perfect hash family3,1143,763T309	54	2
perfect hash family3,1143,763T311	54	2
perfect hash family3,1143,763T313	54	2
perfect hash family3,1146,765T109	54	3
perfect hash family3,1146,765T115	54	3
perfect hash family3,1146,765T124	54	3
perfect hash family3,1149,767T114	54	2
perfect hash family3,1149,767T17	54	2
perfect hash family3,1149,767T3	54	2
perfect hash family3,1149,767T313	54	2
perfect hash family3,2906,122T18	55	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family3,2978,123S3	55	1
perfect hash family4,7776,1296	55	2
perfect hash family4,7776,1296T296	55	1
perfect hash family4,7776,1296T382	55	1
perfect hash family9,14986,126T46	55	1
perfect hash family9,14986,126T53	55	3
perfect hash family9,8536,96S4	55	1
perfect hash family9,8536,96S5	55	1
perfect hash family9,8536,96S6	55	1
perfect hash family9,8536,96S7	55	1
perfect hash family3,3125,125T21	56	1
perfect hash family4,3125,625	56	13
perfect hash family4,3125,625T101	56	2
perfect hash family4,3125,625T102	56	2
perfect hash family4,3125,625T11	56	2
perfect hash family4,3125,625T110	56	3
perfect hash family4,3125,625T120	56	2
perfect hash family4,3125,625T142	56	1
perfect hash family4,3125,625T145	56	1
perfect hash family4,3125,625T152	56	2
perfect hash family4,3125,625T153	56	2
perfect hash family4,3125,625T155	56	2
perfect hash family4,3125,625T159	56	2
perfect hash family4,3125,625T164	56	2
perfect hash family4,3125,625T171	56	5
perfect hash family4,3125,625T173	56	2
perfect hash family4,3125,625T175	56	2
perfect hash family4,3125,625T179	56	2
perfect hash family4,3125,625T180	56	1
perfect hash family4,3125,625T183	56	2
perfect hash family4,3125,625T217	56	1
perfect hash family4,3125,625T218	56	1
perfect hash family4,3125,625T220	56	1
perfect hash family4,3125,625T225	56	1
perfect hash family4,3125,625T246	56	5
perfect hash family4,3125,625T249	56	3
perfect hash family4,3125,625T251	56	1
perfect hash family4,3125,625T252	56	2
perfect hash family4,3125,625T265	56	3
perfect hash family4,3125,625T267	56	1
perfect hash family4,3125,625T276	56	2
perfect hash family4,3125,625T28	56	2
perfect hash family4,3125,625T289	56	3
perfect hash family4,3125,625T301	56	2
perfect hash family4,3125,625T33	56	2
perfect hash family4,3125,625T341	56	2
perfect hash family4,3125,625T353	56	1
perfect hash family4,3125,625T354	56	1
perfect hash family4,3125,625T364	56	1
perfect hash family4,3125,625T388	56	1
perfect hash family4,3125,625T43	56	3
perfect hash family4,3125,625T5	56	1
perfect hash family4,3125,625T57	56	1
perfect hash family4,3125,625T58	56	2

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family4,3125,625T70	56	1
perfect hash family4,3125,625T79	56	2
perfect hash family4,3125,625T83	56	1
perfect hash family4,3125,625T92	56	2
perfect hash family4,3125,625T96	56	2
perfect hash family3,928,620	57	1
perfect hash family3,942,629T61	57	1
perfect hash family3,942,629T83	57	1
perfect hash family3,942,629T87	57	1
perfect hash family3,960,641	57	3
perfect hash family3,960,641T59	57	3
perfect hash family3,972,649T67	57	3
perfect hash family3,972,649T8	57	3
perfect hash family3,973,650	57	3
perfect hash family3,978,653	57	2
perfect hash family3,978,653T86	57	2
perfect hash family3,981,655T14	57	3
perfect hash family3,981,655T5	57	3
perfect hash family3,981,655T73	57	3
perfect hash family3,982,656	57	3
perfect hash family3,993,663	57	1
perfect hash family3,993,663T43	57	1
perfect hash family5,1775,639T89	57	1
perfect hash family6,2536,639T89	57	1
perfect hash family32,512,16,c	58	1
perfect hash family32,512,16T2	58	1
perfect hash family32,512,16T3	58	1
perfect hash family32,512,16T4	58	1
perfect hash family32,513,17	58	3
perfect hash family32,513,17,c	58	1
perfect hash family32,514,18	58	1
perfect hash family32,515,19,c	58	1
perfect hash family32,516,20	58	2
perfect hash family4,243,156	59	1
perfect hash family4,283,180	59	1
perfect hash family4,285,181	59	1
perfect hash family4,285,181T1	59	1
perfect hash family4,305,193	59	2
perfect hash family4,307,195	59	2
perfect hash family5,1358,550	59	1
perfect hash family5,335,156	59	1
perfect hash family5,345,157S3	59	1
perfect hash family5,345,157S4	59	1
perfect hash family5,395,180	59	1
perfect hash family5,405,181	59	1
perfect hash family5,405,181S2	59	1
perfect hash family5,405,181S3	59	1
perfect hash family5,405,181S4	59	1
perfect hash family5,405,181T1	59	1
perfect hash family5,429,193	59	1
perfect hash family5,431,195	59	1
perfect hash family5,440,196S2	59	1
perfect hash family5,440,196S3	59	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family5,440,196S4	59	1
perfect hash family4,315,200	60	2
perfect hash family4,317,202	60	2
perfect hash family4,328,208	60	2
perfect hash family4,328,208T58	60	2
perfect hash family4,328,208T61	60	2
perfect hash family4,328,208T63	60	1
perfect hash family4,328,208T67	60	1
perfect hash family4,332,210	60	2
perfect hash family4,332,210T54	60	1
perfect hash family4,353,224	60	1
perfect hash family4,353,224T14	60	1
perfect hash family4,375,237	60	1
perfect hash family4,375,237T56	60	1
perfect hash family4,375,237T57	60	1
perfect hash family4,388,244	60	2
perfect hash family4,388,244T49	60	2
perfect hash family4,388,244T51	60	2
perfect hash family4,396,250T48	60	1
perfect hash family4,396,250T50	60	1
perfect hash family5,475,207T51	60	1
perfect hash family5,475,207T57	60	1
perfect hash family5,475,207T60	60	1
perfect hash family5,475,207T62	60	1
perfect hash family5,475,207T66	60	1
perfect hash family4,396,250	61	2
perfect hash family4,560,350	61	1
perfect hash family4,887,550	61	1
perfect hash family5,1776,640	61	1
perfect hash family5,444,200	61	2
perfect hash family5,450,202	61	2
perfect hash family5,450,202T2	61	2
perfect hash family5,476,208	61	2
perfect hash family5,478,210	61	1
perfect hash family5,530,236	61	1
perfect hash family5,540,237	61	1
perfect hash family5,556,244	61	2
perfect hash family5,574,250	61	2
perfect hash family5,620,268	61	1
perfect hash family5,866,350	61	1
perfect hash family6,2086,550	61	1
perfect hash family6,2560,640	61	1
perfect hash family6,541,181	61	1
perfect hash family6,561,201	61	2
perfect hash family6,568,208	61	2
perfect hash family6,736,250	61	2
perfect hash family5,515,223T42	62	1
perfect hash family5,515,223T43	62	1
perfect hash family5,550,238S4	62	1
perfect hash family5,550,238T43	62	1
perfect hash family5,550,238T45	62	1
perfect hash family5,570,246T44	62	2
perfect hash family5,570,246T46	62	2

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
perfect hash family6,730,244	62	2
perfect hash family6,730,244T42	62	2
perfect hash family6,730,244T44	62	2
perfect hash family6,730,244T49	62	2
perfect hash family6,730,244T51	62	2
perfect hash family6,735,249	62	3
perfect hash familyD16,1250,50^11 25^5	63	4
perfect hash familyD16,1458,54^11 27^5	63	2
perfect hash familyD16,578,34^11 17^5	63	2
PGL (Maity-Akhtar-Chandrasekharan-Colbourn)	64	2
PGL (Maity-Akhtar-Chandrasekharan-Colbourn) postop NCK	64	3
Power CT100^2	65	1
Power CT100^2,c	65	1
Power CT11^2	65	1
Power CT11^3	65	1
Power CT12^2,c	65	1
Power CT13^3+1	65	1
Power CT16^2	65	1
Power CT17^2	65	4
Power CT17^2+1	65	2
Power CT17^2T1	65	1
Power CT19^2	65	3
Power CT19^2+1	65	6
Power CT19^2+2	65	2
Power CT19^2+3	65	1
Power CT19^2T1	65	3
Power CT19^2T8	65	1
Power CT19^3	65	2
Power CT19^3+1	65	1
Power CT19^3T1	65	2
Power CT19^3T2	65	1
Power CT19^3T5	65	1
Power CT23^2	65	2
Power CT23^2+1	65	4
Power CT23^2T1	65	2
Power CT23^2T2	65	2
Power CT23^2T3	65	1
Power CT23^2T4	65	1
Power CT23^2T5	65	1
Power CT23^3	65	2
Power CT25^2	65	8
Power CT25^2+1	65	6
Power CT25^2T1	65	4
Power CT25^2T14	65	1
Power CT25^2T2	65	5
Power CT25^2T3	65	1
Power CT27^2	65	1
Power CT27^2+1	65	9
Power CT27^2T1	65	9
Power CT27^2T2	65	4
Power CT29^2	65	1
Power CT29^2+1	65	1
Power CT29^2T1	65	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CT29^2T2	65	1
Power CT29^2T3	65	2
Power CT29^2T4	65	1
Power CT31^2	65	1
Power CT31^2T1	65	1
Power CT31^2T2	65	1
Power CT32^2	65	1
Power CT32^2+1	65	1
Power CT32^2+2	65	2
Power CT32^2+3	65	4
Power CT32^2T1	65	5
Power CT32^2T2	65	1
Power CT37^2	65	1
Power CT37^2+1	65	1
Power CT37^2+2	65	1
Power CT37^2+3	65	2
Power CT37^2T1	65	1
Power CT37^2T2	65	2
Power CT41^2	65	1
Power CT41^2+1	65	1
Power CT41^2T1	65	1
Power CT41^2T3	65	1
Power CT41^2T4	65	1
Power CT41^2T6	65	1
Power CT43^2	65	1
Power CT43^2+1	65	1
Power CT43^2+2	65	2
Power CT43^2+3	65	1
Power CT43^2T1	65	1
Power CT43^2T3	65	1
Power CT47^2	65	1
Power CT47^2T2	65	1
Power CT47^2T6	65	1
Power CT49^2+1	65	2
Power CT49^2+2	65	1
Power CT49^2+3	65	1
Power CT49^2T10	65	1
Power CT49^2T11	65	1
Power CT49^2T2	65	1
Power CT49^2T3	65	1
Power CT49^2T4	65	2
Power CT49^2T5	65	1
Power CT49^2T6	65	1
Power CT49^2T7	65	1
Power CT49^2T8	65	1
Power CT49^2T9	65	2
Power CT53^2 +5	65	1
Power CT53^2+1	65	1
Power CT53^2+2	65	2
Power CT53^2T1	65	1
Power CT53^2T3	65	1
Power CT61^2	65	1
Power CT61^2+1	65	1
Power CT61^2T3	65	1
Power CT68^2,c	65	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CT72^2,c	65	1
Power CT80^2,c	65	1
Power CT84^2,c	65	1
Power CT90^2,c	65	1
Power CT98^2,c	65	1
Power CT101^2,cT3c	66	1
Power CT11^2,T3c	66	1
Power CT11^3,+1c	66	1
Power CT11^3,T3c	66	1
Power CT11^3,T6c	66	1
Power CT11^3,T7c	66	1
Power CT11^3T3T3c	66	1
Power CT11^3T7T3c	66	1
Power CT11^4,cT9c	66	1
Power CT11^4,T3c	66	1
Power CT11^4,T6c	66	1
Power CT11^4,T7c	66	1
Power CT11^4T3cT3cT7c	66	1
Power CT11^4T3T3c	66	1
Power CT11^4T7T3c	66	1
Power CT11^4T7T7c	66	1
Power CT13^4S13c	66	1
Power CT68^2,cT33c	66	1
Power CT72^2,cT4c	66	1
Power CT80^2,cT12c	66	1
Power CT80^2,cT8c	66	1
Power CT84^2,cT4c	66	1
Power CT90^2,cT10c	66	1
Power CT90^2,cT6c	66	1
Power CT98^2,cT14c	66	1
Power CT98^2,cT8c	66	1
Power CT12^3,cT1	67	1
Power CT12^3,cT1T1	67	1
Power CT19^2T1T1	67	2
Power CT19^3T1T1	67	2
Power CT19^3T1T1T1	67	1
Power CT19^3T3cS17	67	1
Power CT19^3T3cS18	67	1
Power CT19^3T4S3	67	1
Power CT19^3T4T1	67	1
Power CT19^3T4T1T1	67	1
Power CT19^3T4T2	67	1
Power CT19^3T4T2T1	67	1
Power CT19^3T4T3	67	1
Power CT19^3T5T2	67	1
Power CT23^2T1T1	67	2
Power CT25^2T1T1	67	4
Power CT27^2T1T1	67	9
Power CT27^2T2T1	67	4
Power CT29^2T1T1	67	1
Power CT29^2T2T1	67	1
Power CT31^2T1T1	67	2
Power CT31^2T2T1	67	1
Power CT37^2T1T1	67	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CT37^2T2T1	67	1
Power CT37^2T2T2	67	2
Power CT41^2T1T1	67	2
Power CT41^2T3T1	67	1
Power CT41^2T4T1	67	1
Power CT41^2T6T1	67	1
Power CT43^2T1T1	67	1
Power CT47^2T2T2	67	1
Power CT47^2T5T2	67	1
Power CT53^2T1T1	67	1
Power CT59^2T1T1	67	1
Power CT19^2Arc(3)	68	1
Power CT19^2Arc(4)	68	1
Power CT25^2Arc(3)	68	4
Power CT27^2Arc(2)T2	68	4
Power CT27^2Arc(3)	68	5
Power CT31^2Arc(3)	68	2
Power CT31^2Arc(4)	68	1
Power CT37^2Arc(2)T2	68	1
Power CT41^2Arc(2)T3	68	1
Power CT41^2Arc(2)T4	68	1
Power CT41^2Arc(3)	68	2
Power CT41^2Arc(3)T3	68	1
Power CT41^2Arc(4)	68	2
Power CT41^2Arc(5)	68	2
Power CT41^2Arc(6)	68	2
Power CT41^2Arc(7)	68	1
Power CT41^2Arc(8)	68	1
Power CT43^2Arc(3)	68	1
Power CT43^2Arc(4)	68	1
Power CT53^2Arc(3)	68	1
Power CT53^2Arc(4)	68	1
Power CT53^2Arc(5)	68	1
Power CT53^2Arc(6)	68	1
Power CT53^2Arc(7)	68	1
Power CT53^2Arc(8)	68	1
Power CT59^2Arc(10)	68	1
Power CT59^2Arc(11)	68	1
Power CT59^2Arc(12)	68	1
Power CT59^2Arc(3)	68	1
Power CT59^2Arc(4)	68	1
Power CT59^2Arc(5)	68	1
Power CT59^2Arc(6)	68	1
Power CT59^2Arc(7)	68	1
Power CT59^2Arc(8)	68	1
Power CT59^2Arc(9)	68	1
Power CT73^2Arc(7)T2	68	1
Power CT81^2Arc(7)T2	68	1
Power CT23^2T3T3	69	1
Power CT23^2T4T3	69	1
Power CT23^3T10	69	1
Power CT23^3T11	69	1
Power CT23^3T14	69	1
Power CT23^3T15	69	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CT23^3T16	69	2
Power CT23^3T17	69	2
Power CT23^3T2T2	69	1
Power CT23^3T3T1	69	2
Power CT23^3T3T2	69	3
Power CT23^3T3T3	69	2
Power CT23^3T4	69	1
Power CT23^3T5	69	1
Power CT23^3T5T5	69	1
Power CT23^3T6	69	1
Power CT23^3T6T4	69	1
Power CT23^3T6T5	69	1
Power CT23^3T6T6	69	1
Power CT23^3T7	69	1
Power CT23^3T7T4	69	1
Power CT23^3T7T5	69	1
Power CT23^3T7T6	69	1
Power CT23^3T7T7	69	1
Power CT23^3T8	69	1
Power CT23^3T8T3	69	1
Power CT23^3T8T4	69	1
Power CT23^3T8T5	69	1
Power CT23^3T8T6	69	1
Power CT23^3T8T7	69	1
Power CT23^3T8T8	69	1
Power CT23^3T9	69	1
Power CT23^3T9T3	69	1
Power CT23^3T9T4	69	1
Power CT23^3T9T5	69	1
Power CT23^3T9T6	69	1
Power CT23^3T9T8	69	1
Power CT23^3T9T9	69	1
Power CT25^3T16	69	1
Power CT25^3T17	69	1
Power CT25^3T18	69	1
Power CT25^3T19	69	1
Power CT53^2T3T3	69	1
Power CT23^3T10T10	70	1
Power CT23^3T10T10T10	70	1
Power CT23^3T10T10T3	70	1
Power CT23^3T10T10T4	70	1
Power CT23^3T10T10T9	70	1
Power CT23^3T10T3	70	1
Power CT23^3T10T4	70	1
Power CT23^3T10T5	70	1
Power CT23^3T10T9	70	1
Power CT23^3T10T9T3	70	1
Power CT23^3T10T9T4	70	1
Power CT23^3T10T9T5	70	1
Power CT23^3T10T9T9	70	1
Power CT23^3T11T10T4	70	1
Power CT23^3T11T4	70	1
Power CT23^3T2T2T1	70	1
Power CT23^3T2T2T2	70	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CT23^3T3T1T1	70	2
Power CT23^3T3T2T1	70	2
Power CT23^3T3T2T2	70	1
Power CT23^3T3T3T1	70	2
Power CT23^3T3T3T2	70	2
Power CT23^3T3T3T3	70	2
Power CT23^3T6T6T6	70	1
Power CT23^3T7T5T5	70	1
Power CT23^3T7T6T5	70	1
Power CT23^3T7T7T4	70	1
Power CT23^3T7T7T5	70	1
Power CT23^3T7T7T7	70	1
Power CT23^3T8T6T4	70	1
Power CT23^3T8T6T5	70	1
Power CT23^3T8T7T4	70	1
Power CT23^3T8T7T5	70	1
Power CT23^3T8T7T6	70	1
Power CT23^3T8T7T7	70	1
Power CT23^3T8T8T4	70	1
Power CT23^3T8T8T5	70	1
Power CT23^3T8T8T6	70	1
Power CT23^3T8T8T7	70	1
Power CT23^3T8T8T8	70	1
Power CT23^3T9T6T4	70	1
Power CT23^3T9T6T5	70	1
Power CT23^3T9T8T3	70	1
Power CT23^3T9T8T4	70	1
Power CT23^3T9T8T5	70	1
Power CT23^3T9T8T6	70	1
Power CT23^3T9T8T8	70	1
Power CT23^3T9T9T3	70	1
Power CT23^3T9T9T4	70	1
Power CT23^3T9T9T5	70	1
Power CT23^3T9T9T6	70	1
Power CT23^3T9T9T8	70	1
Power CT23^3T9T9T9	70	1
Power CT25^3S11	71	1
Power CT25^3S12	71	1
Power CT25^3S13	71	1
Power CT25^3S14	71	1
Power CT25^3S15	71	1
Power CT25^3S16	71	1
Power CT25^3S17	71	1
Power CT25^3S18	71	1
Power CT25^3S19	71	1
Power CT25^3S20	71	1
Power CT25^3S21	71	1
Power CT25^3S22	71	1
Power CT25^3S23	71	1
Power CT25^3S24	71	1
Power CT25^3S25	71	1
Power CT25^3T10S23	71	1
Power CT25^3T10S24	71	1
Power CT25^3T11S24	71	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CT25^3T12S22	71	1
Power CT25^3T12S23	71	1
Power CT25^3T12S24	71	1
Power CT25^3T2S24	71	1
Power CT25^3T3S24	71	1
Power CT25^3T4S24	71	1
Power CT25^3T5S24	71	1
Power CT25^3T6S24	71	1
Power CT25^3T7S24	71	1
Power CT25^3T8S24	71	1
Power CT25^3T9S22	71	1
Power CT25^3T9S23	71	1
Power CT25^3T9S24	71	1
Power CT27^3S20	71	1
Power CT27^3S21	71	1
Power CT31^3T3S30	71	2
Power CT31^3T5S29	71	2
Power CT31^3T5S30	71	2
Power CT37^2Arc(10):3 twos,2 ones	72	1
Power CT37^2Arc(10):4 twos,1 ones	72	1
Power CT37^2Arc(10):5 twos,0 ones	72	2
Power CT37^2Arc(12):4 twos,2 ones	72	1
Power CT37^2Arc(6):2 twos,1 ones	72	1
Power CT37^2Arc(6):3 twos,0 ones	72	2
Power CT37^2Arc(8):2 twos,2 ones	72	1
Power CT37^2Arc(8):3 twos,1 ones	72	1
Power CT37^2Arc(8):4 twos,0 ones	72	2
Power CT47^2Arc(6):3 twos,0 ones	72	1
Power CT47^2Arc(8):4 twos,0 ones	72	1
Power CZ2-12.4-10.1	73	1
Power CZ2-17.4-16.1	73	1
Power CZ2-27.12-26.1	73	1
Power CZ3-11.8-10.1	73	1
Power CZ3-11.8-8.1	73	1
Power CZ3-13.12-10.1	73	1
Power CZ3-13.12-11.1	73	1
Power CZ3-13.3-12.6	73	1
Power CZ3-13.4-12.5	73	1
Power CZ3-13.5-10.1	73	1
Power CZ3-13.5-12.1	73	1
Power CZ3-13.5-12.4	73	1
Power CZ3-13.5-6.1	73	1
Power CZ3-13.5-8.1	73	1
Power CZ3-13.5-9.1	73	1
Power CZ3-13.6-12.3	73	1
Power CZ3-13.8-12.1	73	1
Power CZ3-16.5-14.1	73	1
Power CZ3-19.15-15.4	73	1
Power CZ3-19.18-15.1	73	1
Power CZ3-19.18-16.1	73	1
Power CZ3-19.18-17.1	73	1
Power CZ3-19.5-16.1	73	1
Power CZ3-19.5-18.1	73	1
Power CZ3-23.18-15.1	73	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CZ3-23.18-16.1	73	1
Power CZ3-23.18-17.1	73	1
Power CZ3-23.18-18.1	73	1
Power CZ3-23.20-17.1	73	1
Power CZ3-23.20-18.1	73	1
Power CZ3-23.20-19.1	73	1
Power CZ3-9.8-5.1	73	1
Power CZ3-9.8-6.1	73	1
Power CZ2-16.5	74	1
Power CZ2-19.13	74	1
Power CZ2-27.13	74	1
Power CZ3-14.6	74	1
Power CZ3-16.6	74	1
Power CZ3-19.6	74	1
Power CZ3-20.6	74	1
Power CZ3-21.19	74	1
Power CZ3-9.9	74	1
Power CZ3-11.7-10.1-10.1	75	1
Power CZ3-11.7-8.1-10.1	75	1
Power CZ3-11.7-8.1-9.1	75	1
Power CZ3-11.7-9.1-10.1	75	1
Power CZ3-13.4-6.1-9.1	75	1
Power CZ3-13.7-12.1-12.1	75	1
Power CZ3-16.4-14.1-14.1	75	1
Power CZ3-16.7-12.1-12.1	75	1
Power CZ3-16.7-12.1-13.1	75	1
Power CZ3-17.4-16.1-16.1	75	1
Power CZ3-19.11-18.0-17.8	75	1
Power CZ3-19.17-15.1-15.1	75	1
Power CZ3-19.17-15.1-16.1	75	1
Power CZ3-19.17-15.1-18.1	75	1
Power CZ3-19.17-16.1-16.1	75	1
Power CZ3-19.17-16.1-17.1	75	1
Power CZ3-19.17-16.1-18.1	75	1
Power CZ3-19.17-17.1-17.1	75	1
Power CZ3-19.17-17.1-18.1	75	1
Power CZ3-19.4-14.1-18.1	75	1
Power CZ3-19.4-16.1-16.1	75	1
Power CZ3-19.4-16.1-18.1	75	1
Power CZ3-19.4-17.1-18.1	75	1
Power CZ4-13.0-12.12-10.1	75	1
Power CZ4-13.0-12.12-4.1	75	1
Power CZ4-13.0-12.12-5.1	75	1
Power CZ4-13.0-12.12-8.1	75	1
Power CZ4-13.0-12.12-9.1	75	1
Power CZ3-16.3-14.1-14.1-14.1	76	1
Power CZ3-19.16-15.1-15.1-15.1	76	1
Power CZ3-19.16-15.1-15.1-16.1	76	1
Power CZ3-19.16-15.1-15.1-18.1	76	1
Power CZ3-19.16-15.1-16.1-16.1	76	1
Power CZ3-19.16-15.1-16.1-18.1	76	1
Power CZ3-19.16-16.1-16.1-16.1	76	1
Power CZ3-19.16-16.1-16.1-17.1	76	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CZ3-19.16-16.1-16.1-18.1	76	1
Power CZ3-19.16-16.1-17.1-17.1	76	1
Power CZ3-19.16-16.1-17.1-18.1	76	1
Power CZ3-19.3-16.1-16.1-18.1	76	1
Power CZ3-19.3-16.1-17.1-18.1	76	1
Power CZ3-23.0-22.10-21.11	77	1
Power CZ3-23.0-22.11-21.10	77	1
Power CZ3-23.0-22.12-21.9	77	1
Power CZ3-23.0-22.13-21.8	77	1
Power CZ3-23.0-22.14-21.7	77	1
Power CZ3-23.0-22.15-21.6	77	1
Power CZ3-23.0-22.16-21.5	77	1
Power CZ3-23.0-22.17-21.4	77	1
Power CZ3-23.0-22.18-21.3	77	1
Power CZ3-23.0-22.19-21.2	77	1
Power CZ3-23.0-22.20-21.1	77	1
Power CZ3-23.1-22.10-21.10	77	1
Power CZ3-23.1-22.11-21.9	77	1
Power CZ3-23.1-22.12-21.8	77	1
Power CZ3-23.1-22.13-21.7	77	1
Power CZ3-23.1-22.14-21.6	77	1
Power CZ3-23.1-22.15-21.5	77	1
Power CZ3-23.1-22.16-21.4	77	1
Power CZ3-23.1-22.17-21.3	77	1
Power CZ3-23.1-22.18-21.2	77	1
Power CZ3-23.1-22.19-21.1	77	1
Power CZ3-23.10-22.10-21.1	77	1
Power CZ3-23.2-22.18-21.1	77	1
Power CZ3-23.4-22.16-21.1	77	1
Power CZ3-23.5-22.15-21.1	77	1
Power CZ3-23.6-22.14-21.1	77	1
Power CZ3-23.7-22.13-21.1	77	1
Power CZ3-23.8-22.11-21.2	77	1
Power CZ3-23.8-22.12-21.1	77	1
Power CZ3-23.9-22.10-21.2	77	1
Power CZ3-23.9-22.11-21.1	77	1
Power CZ3-23.0-22.21	78	1
Power CZ3-23.1-22.20	78	1
Power CZ3-23.1-22.5	78	1
Power CZ3-23.10-22.11	78	1
Power CZ3-23.11-22.10	78	1
Power CZ3-23.12-22.9	78	1
Power CZ3-23.13-22.8	78	1
Power CZ3-23.14-22.7	78	1
Power CZ3-23.15-22.6	78	1
Power CZ3-23.16-22.5	78	1
Power CZ3-23.2-22.19	78	1
Power CZ3-23.3-22.18	78	1
Power CZ3-23.4-22.17	78	1
Power CZ3-23.5-22.16	78	1
Power CZ3-23.6-22.15	78	1
Power CZ3-23.7-22.14	78	1
Power CZ3-23.8-22.13	78	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CZ3-23.9-22.12	78	1
Power CZ3-23.0-22.9-21.12	79	1
Power CZ3-23.1-22.4-20.1	79	1
Power CZ3-23.1-22.7-21.13	79	1
Power CZ3-23.1-22.8-21.12	79	1
Power CZ3-23.1-22.9-21.11	79	1
Power CZ3-23.10-22.8-21.3	79	1
Power CZ3-23.10-22.9-21.2	79	1
Power CZ3-23.11-22.7-21.3	79	1
Power CZ3-23.11-22.8-21.2	79	1
Power CZ3-23.11-22.9-21.1	79	1
Power CZ3-23.12-22.5-21.4	79	1
Power CZ3-23.12-22.6-21.3	79	1
Power CZ3-23.12-22.7-21.2	79	1
Power CZ3-23.12-22.8-21.1	79	1
Power CZ3-23.13-22.3-21.5	79	1
Power CZ3-23.13-22.4-21.4	79	1
Power CZ3-23.13-22.5-21.3	79	1
Power CZ3-23.13-22.6-21.2	79	1
Power CZ3-23.13-22.7-21.1	79	1
Power CZ3-23.14-22.2-21.5	79	1
Power CZ3-23.14-22.3-21.4	79	1
Power CZ3-23.14-22.4-21.1	79	1
Power CZ3-23.14-22.4-21.3	79	1
Power CZ3-23.14-22.5-21.2	79	1
Power CZ3-23.14-22.6-18.1	79	1
Power CZ3-23.14-22.6-21.1	79	1
Power CZ3-23.15-22.0-21.6	79	1
Power CZ3-23.15-22.1-21.5	79	1
Power CZ3-23.15-22.2-21.2	79	1
Power CZ3-23.15-22.2-21.4	79	1
Power CZ3-23.15-22.3-15.1	79	1
Power CZ3-23.15-22.3-21.1	79	1
Power CZ3-23.15-22.3-21.3	79	1
Power CZ3-23.15-22.4-21.2	79	1
Power CZ3-23.15-22.5-18.1	79	1
Power CZ3-23.15-22.5-21.1	79	1
Power CZ3-23.16-22.0-21.3	79	1
Power CZ3-23.16-22.0-21.5	79	1
Power CZ3-23.16-22.1-21.2	79	1
Power CZ3-23.16-22.1-21.4	79	1
Power CZ3-23.16-22.2-15.1	79	1
Power CZ3-23.16-22.2-21.3	79	1
Power CZ3-23.16-22.3-21.2	79	1
Power CZ3-23.16-22.4-18.1	79	1
Power CZ3-23.16-22.4-21.1	79	1
Power CZ3-23.17-21.1-21.1	79	1
Power CZ3-23.17-22.0-21.4	79	1
Power CZ3-23.17-22.1-21.3	79	1
Power CZ3-23.17-22.2-21.2	79	1
Power CZ3-23.17-22.3-18.1	79	1
Power CZ3-23.17-22.3-21.1	79	1
Power CZ3-23.18-22.0-21.3	79	1
Power CZ3-23.18-22.1-21.2	79	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power CZ3-23.18-22.2-17.1	79	1
Power CZ3-23.18-22.2-18.1	79	1
Power CZ3-23.3-22.2-20.1	79	1
Power CZ3-23.16-15.1-21.1-22.1	80	1
Power CZ3-23.18-18.1-21.1-21.1	80	1
Power CZ3-23.18-18.1-21.1-22.1	80	1
Power CZ3-23.18-19.1-19.1-21.1	80	1
Power CZ3-23.18-19.1-20.1-21.1	80	1
Power CZ3-23.18-19.1-21.1-21.1	80	1
Power CZ3-23.18-19.1-21.1-22.1	80	1
Power CZ3-23.18-20.1-20.1-20.1	80	1
Power CZ3-23.18-20.1-20.1-21.1	80	1
Power CZ3-23.18-20.1-21.1-21.1	80	1
Power CZ3-23.18-20.1-21.1-22.1	80	1
Power CZ3-23.3-20.1-20.1-20.1	80	1
Power CZ3-23.3-20.1-20.1-22.1	80	1
Power CZ3-23.17-15.1-21.1	81	1
Power CZ3-23.17-15.1-22.1	81	1
Power CZ3-23.17-16.1-22.1	81	1
Power CZ3-23.17-17.1-22.1	81	1
Power CZ3-23.19-17.1-21.1	81	1
Power CZ3-23.19-17.1-22.1	81	1
Power CZ3-23.19-18.1-20.1	81	1
Power CZ3-23.19-18.1-21.1	81	1
Power CZ3-23.19-18.1-22.1	81	1
Power CZ3-23.19-19.1-20.1	81	1
Power CZ3-23.19-19.1-21.1	81	1
Power CZ3-23.19-19.1-22.1	81	1
Power CZ3-23.19-20.1-20.1	81	1
Power CZ3-23.19-20.1-21.1	81	1
Power N-CT101^2	82	13
Power N-CT101^2T10	82	3
Power N-CT101^2T12T1	82	2
Power N-CT101^2T12T3	82	3
Power N-CT101^2T13	82	1
Power N-CT101^2T16	82	1
Power N-CT101^2T18	82	1
Power N-CT101^2T19	82	2
Power N-CT101^2T19T3	82	3
Power N-CT101^2T1T1	82	4
Power N-CT101^2T20	82	2
Power N-CT101^2T24	82	1
Power N-CT101^2T25T1	82	2
Power N-CT101^2T3T3	82	3
Power N-CT101^2T6	82	2
Power N-CT101^2T7	82	1
Power N-CT101^2T8	82	3
Power N-CT101^2Arc(13)T11	83	1
Power N-CT101^2Arc(13)T13	83	2
Power N-CT101^2Arc(13)T18	83	2
Power N-CT101^2Arc(13)T2	83	5
Power N-CT101^2Arc(13)T4	83	2
Power N-CT101^2Arc(13)T6	83	2

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT101^2Arc(2)T12	83	2
Power N-CT101^2Arc(2)T25	83	2
Power N-CT101^2Arc(3)	83	7
Power N-CT101^2Arc(3)T12	83	2
Power N-CT101^2Arc(3)T25	83	2
Power N-CT101^2Arc(4)	83	7
Power N-CT101^2Arc(4)T12	83	2
Power N-CT101^2Arc(4)T25	83	2
Power N-CT101^2Arc(5)	83	7
Power N-CT101^2Arc(6)	83	5
Power N-CT101^2Arc(7)	83	5
Power N-CT101^2Arc(8)	83	5
Power N-CT101^2Arc(9)T12	83	2
Power N-CT101^2Trin2,2,2	84	1
Power N-CT101^2Trin2,2,6	84	1
Power N-CT41^2Trin1,2,13	84	3
Power N-CT41^2Trin1,2,8	84	3
Power N-CT41^2Trin1,4,4	84	2
Power N-CT41^2Trin1,5,7	84	1
Power N-CT41^2Trin1,6,13	84	3
Power N-CT41^2Trin1,7,7	84	2
Power N-CT41^2Trin1,8,13	84	3
Power N-CT41^2Trin2,2,13	84	3
Power N-CT41^2Trin2,2,2	84	3
Power N-CT41^2Trin2,2,7	84	1
Power N-CT41^2Trin2,2,8	84	3
Power N-CT41^2Trin2,4,7	84	1
Power N-CT41^2Trin2,6,13	84	3
Power N-CT41^2Trin2,7,7	84	3
Power N-CT41^2Trin2,8,13	84	3
Power N-CT41^2Trin3,3,12	84	1
Power N-CT41^2Trin3,3,3	84	1
Power N-CT41^2Trin3,6,12	84	1
Power N-CT41^2Trin3,7,7	84	2
Power N-CT41^2Trin4,4,10	84	2
Power N-CT41^2Trin4,4,12	84	2
Power N-CT41^2Trin6,6,6	84	2
Power N-CT47^2Trin1,2,8	84	1
Power N-CT47^2Trin1,3,5	84	2
Power N-CT47^2Trin2,2,2	84	4
Power N-CT53^2Trin1,2,4	84	2
Power N-CT53^2Trin1,4,4	84	4
Power N-CT53^2Trin2,2,4	84	2
Power N-CT53^2Trin2,2,7	84	3
Power N-CT53^2Trin2,4,4	84	4
Power N-CT53^2Trin4,4,4	84	6
Power N-CT59^2Trin1,3,13	84	3
Power N-CT59^2Trin1,3,3	84	3
Power N-CT59^2Trin2,2,2	84	1
Power N-CT59^2Trin3,3,13	84	3
Power N-CT59^2Trin3,3,3	84	3
Power N-CT67^2Trin1,3,14	84	3
Power N-CT67^2Trin2,2,2	84	1
Power N-CT67^2Trin2,2,7	84	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT67^2Trin2,7,7	84	1
Power N-CT67^2Trin3,3,14	84	3
Power N-CT67^2Trin6,7,7	84	1
Power N-CT67^2Trin7,7,12	84	1
Power N-CT67^2Trin7,7,7	84	1
Power N-CT71^2Trin1,3,15	84	1
Power N-CT71^2Trin2,2,2	84	2
Power N-CT73^2Trin1,4,4	84	1
Power N-CT73^2Trin4,4,4	84	1
Power N-CT79^2Trin1,4,34	84	2
Power N-CT79^2Trin2,2,13	84	1
Power N-CT79^2Trin2,2,14	84	1
Power N-CT79^2Trin2,2,2	84	2
Power N-CT79^2Trin2,2,23	84	2
Power N-CT79^2Trin2,2,28	84	2
Power N-CT79^2Trin2,2,4	84	1
Power N-CT79^2Trin2,2,7	84	1
Power N-CT79^2Trin2,4,4	84	1
Power N-CT79^2Trin3,3,23	84	2
Power N-CT79^2Trin3,3,3	84	2
Power N-CT79^2Trin3,3,5	84	2
Power N-CT79^2Trin4,4,4	84	1
Power N-CT89^2Trin1,5,5	84	2
Power N-CT89^2Trin2,2,13	84	2
Power N-CT89^2Trin2,2,2	84	4
Power N-CT89^2Trin2,2,4	84	4
Power N-CT89^2Trin2,4,4	84	4
Power N-CT89^2Trin3,3,23	84	1
Power N-CT89^2Trin3,3,3	84	1
Power N-CT89^2Trin4,4,4	84	5
Power N-CT97^2Trin2,2,2	84	3
Power N-CT97^2Trin2,2,4	84	2
Power N-CT97^2Trin2,4,4	84	2
Power N-CT97^2Trin3,3,3	84	1
Power N-CT97^2Trin3,3,6	84	1
Power N-CT97^2Trin3,3,9	84	1
Power N-CT97^2Trin3,6,6	84	1
Power N-CT97^2Trin4,4,4	84	2
Power N-CT97^2Trin5,5,5	84	1
Power N-CT29^2	85	8
Power N-CT29^2+1	85	4
Power N-CT29^2T1	85	6
Power N-CT31^2	85	11
Power N-CT37^2+1	85	2
Power N-CT37^2+2	85	5
Power N-CT41^2	85	11
Power N-CT43^2	85	6
Power N-CT43^2+1	85	3
Power N-CT43^2+2	85	9
Power N-CT43^2+3	85	5
Power N-CT47^2	85	2
Power N-CT47^2+1	85	4
Power N-CT47^2+2	85	7
Power N-CT47^2+3	85	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT53^2	85	3
Power N-CT53^2+1	85	7
Power N-CT53^2+2	85	2
Power N-CT53^2+3	85	7
Power N-CT59^2	85	1
Power N-CT59^2+1	85	3
Power N-CT61^2	85	7
Power N-CT67^2	85	5
Power N-CT67^2+1	85	4
Power N-CT67^2+3	85	2
Power N-CT73^2	85	7
Power N-CT73^2+1	85	5
Power N-CT73^2+2	85	4
Power N-CT73^2+3	85	5
Power N-CT79^2	85	1
Power N-CT79^2+1	85	2
Power N-CT79^2+2	85	2
Power N-CT79^2+3	85	7
Power N-CT83^2	85	4
Power N-CT83^2+1	85	2
Power N-CT83^2+2	85	5
Power N-CT83^2+3	85	1
Power N-CT89^2	85	7
Power N-CT89^2+1	85	1
Power N-CT89^2+2	85	3
Power N-CT89^2+3	85	4
Power N-CT97^2	85	3
Power N-CT97^2+1	85	3
Power N-CT97^2+2	85	2
Power N-CT97^2+3	85	5
Power N-CT29^2Arc(2)T6	86	2
Power N-CT29^2Arc(3)T6	86	2
Power N-CT29^2Arc(4)T6	86	2
Power N-CT31^2Arc(2)T3	86	2
Power N-CT31^2Arc(2)T5	86	3
Power N-CT31^2Arc(2)T8	86	2
Power N-CT37^2Arc(2)T4	86	2
Power N-CT41^2Arc(2)T13	86	3
Power N-CT41^2Arc(2)T2	86	3
Power N-CT41^2Arc(2)T4	86	2
Power N-CT41^2Arc(2)T8	86	3
Power N-CT41^2Arc(3)T4	86	2
Power N-CT41^2Arc(4)T4	86	2
Power N-CT43^2Arc(2)T14	86	1
Power N-CT43^2Arc(2)T4	86	1
Power N-CT47^2Arc(2)T17	86	1
Power N-CT47^2Arc(2)T3	86	2
Power N-CT47^2Arc(2)T4	86	2
Power N-CT47^2Arc(2)T5	86	2
Power N-CT47^2Arc(2)T8	86	1
Power N-CT47^2Arc(3)T5	86	2
Power N-CT53^2Arc(2)T2	86	2
Power N-CT53^2Arc(2)T4	86	2
Power N-CT53^2Arc(9)T10	86	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT53^2Arc(9)T11	86	1
Power N-CT53^2Arc(9)T2	86	1
Power N-CT53^2Arc(9)T6	86	1
Power N-CT53^2Arc(9)T8	86	1
Power N-CT59^2Arc(2)T13	86	3
Power N-CT59^2Arc(2)T3	86	3
Power N-CT59^2Arc(3)T13	86	3
Power N-CT59^2Arc(3)T3	86	3
Power N-CT61^2Arc(9)T11	86	1
Power N-CT61^2Arc(9)T13	86	1
Power N-CT61^2Arc(9)T16	86	1
Power N-CT61^2Arc(9)T2	86	1
Power N-CT61^2Arc(9)T24	86	1
Power N-CT61^2Arc(9)T25	86	1
Power N-CT61^2Arc(9)T6	86	1
Power N-CT61^2Arc(9)T7	86	1
Power N-CT61^2Arc(9)T8	86	1
Power N-CT67^2Arc(2)T14	86	3
Power N-CT67^2Arc(2)T7	86	3
Power N-CT67^2Arc(3)T14	86	3
Power N-CT67^2Arc(3)T7	86	3
Power N-CT67^2Arc(4)T7	86	3
Power N-CT67^2Arc(5)T7	86	3
Power N-CT67^2Arc(6)T7	86	1
Power N-CT71^2Arc(2)T15	86	1
Power N-CT71^2Arc(3)T15	86	1
Power N-CT71^2Arc(9)T10	86	1
Power N-CT71^2Arc(9)T11	86	1
Power N-CT71^2Arc(9)T16	86	1
Power N-CT71^2Arc(9)T2	86	1
Power N-CT71^2Arc(9)T20	86	1
Power N-CT71^2Arc(9)T9	86	1
Power N-CT73^2Arc(2)T12	86	1
Power N-CT73^2Arc(2)T4	86	1
Power N-CT73^2Arc(2)T9	86	1
Power N-CT73^2Arc(3)T12	86	1
Power N-CT73^2Arc(3)T4	86	1
Power N-CT73^2Arc(4)T12	86	1
Power N-CT73^2Arc(4)T4	86	1
Power N-CT79^2Arc(2)T27	86	4
Power N-CT79^2Arc(3)T27	86	4
Power N-CT79^2Arc(4)T27	86	4
Power N-CT79^2Arc(5)T27	86	4
Power N-CT79^2Arc(6)T27	86	4
Power N-CT79^2Arc(9)T10	86	1
Power N-CT79^2Arc(9)T11	86	1
Power N-CT79^2Arc(9)T12	86	1
Power N-CT79^2Arc(9)T18	86	1
Power N-CT79^2Arc(9)T2	86	1
Power N-CT79^2Arc(9)T20	86	1
Power N-CT79^2Arc(9)T25	86	1
Power N-CT79^2Arc(9)T27	86	1
Power N-CT83^2Arc(13)T2	86	2
Power N-CT83^2Arc(2)T17	86	3
Power N-CT83^2Arc(2)T27	86	2

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT83^2Arc(2)T5	86	1
Power N-CT83^2Arc(2)T9	86	1
Power N-CT83^2Arc(3)T27	86	2
Power N-CT83^2Arc(3)T5	86	1
Power N-CT83^2Arc(4)T27	86	2
Power N-CT83^2Arc(4)T5	86	1
Power N-CT83^2Arc(9)T2	86	1
Power N-CT83^2Arc(9)T9	86	1
Power N-CT89^2Arc(2)T19	86	1
Power N-CT89^2Arc(2)T27	86	2
Power N-CT89^2Arc(2)T6	86	1
Power N-CT89^2Arc(3)T19	86	1
Power N-CT89^2Arc(3)T27	86	2
Power N-CT89^2Arc(3)T6	86	1
Power N-CT89^2Arc(4)T19	86	1
Power N-CT89^2Arc(4)T6	86	1
Power N-CT89^2Arc(5)T19	86	1
Power N-CT89^2Arc(5)T6	86	1
Power N-CT89^2Arc(6)T19	86	1
Power N-CT89^2Arc(6)T6	86	1
Power N-CT89^2Arc(9)T12	86	1
Power N-CT89^2Arc(9)T19	86	1
Power N-CT89^2Arc(9)T2	86	1
Power N-CT97^2Arc(12)T2	86	2
Power N-CT97^2Arc(13)T15	86	2
Power N-CT97^2Arc(13)T2	86	3
Power N-CT97^2Arc(13)T4	86	2
Power N-CT97^2Arc(13)T6	86	2
Power N-CT29^2Arc(3)	87	4
Power N-CT29^2Arc(4)	87	2
Power N-CT31^2Arc(3)	87	4
Power N-CT37^2Arc(3)	87	2
Power N-CT37^2Arc(4)	87	2
Power N-CT43^2Arc(3)	87	3
Power N-CT43^2Arc(4)	87	3
Power N-CT47^2Arc(3)	87	5
Power N-CT47^2Arc(4)	87	3
Power N-CT47^2Arc(5)	87	3
Power N-CT47^2Arc(6)	87	3
Power N-CT47^2Arc(7)	87	5
Power N-CT47^2Arc(8)	87	5
Power N-CT53^2Arc(10)	87	4
Power N-CT53^2Arc(11)	87	4
Power N-CT53^2Arc(12)	87	4
Power N-CT53^2Arc(6)	87	1
Power N-CT53^2Arc(7)	87	5
Power N-CT53^2Arc(8)	87	5
Power N-CT53^2Arc(9)	87	5
Power N-CT61^2Arc(3)	87	3
Power N-CT61^2Arc(4)	87	3
Power N-CT67^2Arc(11)	87	4
Power N-CT67^2Arc(12)	87	4
Power N-CT67^2Arc(3)	87	3
Power N-CT67^2Arc(4)	87	3

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT67^2Arc(5)	87	3
Power N-CT67^2Arc(6)	87	3
Power N-CT67^2Arc(7)	87	1
Power N-CT71^2Arc(3)	87	2
Power N-CT71^2Arc(4)	87	2
Power N-CT71^2Arc(5)	87	2
Power N-CT71^2Arc(6)	87	2
Power N-CT71^2Arc(7)	87	2
Power N-CT71^2Arc(8)	87	2
Power N-CT73^2Arc(10)	87	1
Power N-CT73^2Arc(11)	87	1
Power N-CT73^2Arc(12)	87	1
Power N-CT73^2Arc(13)	87	1
Power N-CT73^2Arc(3)	87	2
Power N-CT73^2Arc(4)	87	2
Power N-CT73^2Arc(5)	87	1
Power N-CT73^2Arc(6)	87	1
Power N-CT73^2Arc(7)	87	1
Power N-CT73^2Arc(8)	87	1
Power N-CT73^2Arc(9)	87	1
Power N-CT79^2Arc(10)	87	5
Power N-CT79^2Arc(11)	87	5
Power N-CT79^2Arc(12)	87	5
Power N-CT79^2Arc(3)	87	6
Power N-CT79^2Arc(4)	87	6
Power N-CT79^2Arc(5)	87	6
Power N-CT79^2Arc(6)	87	6
Power N-CT79^2Arc(7)	87	6
Power N-CT79^2Arc(8)	87	6
Power N-CT79^2Arc(9)	87	6
Power N-CT83^2Arc(3)	87	3
Power N-CT83^2Arc(4)	87	5
Power N-CT83^2Arc(5)	87	5
Power N-CT83^2Arc(6)	87	5
Power N-CT83^2Arc(7)	87	5
Power N-CT83^2Arc(8)	87	6
Power N-CT89^2Arc(10)	87	2
Power N-CT89^2Arc(11)	87	2
Power N-CT89^2Arc(12)	87	2
Power N-CT89^2Arc(3)	87	1
Power N-CT89^2Arc(4)	87	1
Power N-CT89^2Arc(5)	87	1
Power N-CT89^2Arc(6)	87	1
Power N-CT89^2Arc(9)	87	1
Power N-CT97^2Arc(10)	87	3
Power N-CT97^2Arc(11)	87	3
Power N-CT97^2Arc(12)	87	3
Power N-CT97^2Arc(13)	87	3
Power N-CT97^2Arc(3)	87	3
Power N-CT97^2Arc(4)	87	3
Power N-CT97^2Arc(5)	87	3
Power N-CT97^2Arc(6)	87	3
Power N-CT97^2Arc(7)	87	3
Power N-CT97^2Arc(8)	87	3

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT97^2Arc(9)	87	3
Power N-CT29^2T1T1	88	4
Power N-CT29^2T2T2	88	1
Power N-CT29^2T4T2	88	1
Power N-CT29^2T6T1	88	2
Power N-CT29^2T6T2	88	1
Power N-CT29^2T8T5	88	1
Power N-CT29^2T8T6	88	1
Power N-CT31^2T1T1	88	5
Power N-CT31^2T2T1	88	1
Power N-CT31^2T3T1	88	2
Power N-CT31^2T5T1	88	3
Power N-CT31^2T8T1	88	2
Power N-CT37^2T12T3	88	2
Power N-CT37^2T1T1	88	2
Power N-CT37^2T4T1	88	2
Power N-CT37^2T4T3	88	2
Power N-CT37^2T8T3	88	2
Power N-CT37^2T8T4	88	2
Power N-CT41^2T10T3	88	1
Power N-CT41^2T10T4	88	3
Power N-CT41^2T10T5	88	1
Power N-CT41^2T10T6	88	3
Power N-CT41^2T10T8	88	1
Power N-CT41^2T12T3	88	1
Power N-CT41^2T12T4	88	2
Power N-CT41^2T12T6	88	1
Power N-CT41^2T13T1	88	3
Power N-CT41^2T13T13	88	3
Power N-CT41^2T13T2	88	3
Power N-CT41^2T13T6	88	3
Power N-CT41^2T13T8	88	3
Power N-CT41^2T14T3	88	1
Power N-CT41^2T1T1	88	2
Power N-CT41^2T2T1	88	3
Power N-CT41^2T2T2	88	8
Power N-CT41^2T3T2	88	1
Power N-CT41^2T3T3	88	2
Power N-CT41^2T4T1	88	2
Power N-CT41^2T4T2	88	1
Power N-CT41^2T4T4	88	4
Power N-CT41^2T6T6	88	2
Power N-CT41^2T7T2	88	1
Power N-CT41^2T7T3	88	2
Power N-CT41^2T7T4	88	2
Power N-CT41^2T7T5	88	1
Power N-CT41^2T7T7	88	3
Power N-CT41^2T8T1	88	3
Power N-CT41^2T8T2	88	3
Power N-CT43^2T13T1	88	1
Power N-CT43^2T14T1	88	1
Power N-CT43^2T1T1	88	3
Power N-CT43^2T8T1	88	1
Power N-CT43^2T9T1	88	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT47^2T14T3	88	1
Power N-CT47^2T17T1	88	1
Power N-CT47^2T1T1	88	7
Power N-CT47^2T2T1	88	2
Power N-CT47^2T2T2	88	5
Power N-CT47^2T3T1	88	4
Power N-CT47^2T3T3	88	1
Power N-CT47^2T4T1	88	2
Power N-CT47^2T4T2	88	2
Power N-CT47^2T5T1	88	2
Power N-CT47^2T5T3	88	2
Power N-CT47^2T6T1	88	2
Power N-CT47^2T8T1	88	1
Power N-CT47^2T8T2	88	1
Power N-CT53^2T1T1	88	2
Power N-CT53^2T2T1	88	2
Power N-CT53^2T2T2	88	2
Power N-CT53^2T4T1	88	4
Power N-CT53^2T4T2	88	2
Power N-CT53^2T4T4	88	6
Power N-CT53^2T5T5	88	3
Power N-CT53^2T7T2	88	3
Power N-CT53^2T9T2	88	1
Power N-CT59^2T10T4	88	3
Power N-CT59^2T13T1	88	3
Power N-CT59^2T13T3	88	3
Power N-CT59^2T15T4	88	1
Power N-CT59^2T15T5	88	1
Power N-CT59^2T2T2	88	1
Power N-CT59^2T3T1	88	3
Power N-CT59^2T3T3	88	3
Power N-CT59^2T5T5	88	2
Power N-CT59^2T6T2	88	1
Power N-CT59^2T8T3	88	3
Power N-CT61^2T1T1	88	3
Power N-CT67^2T14T1	88	3
Power N-CT67^2T14T3	88	3
Power N-CT67^2T14T5	88	3
Power N-CT67^2T1T1	88	3
Power N-CT67^2T2T2	88	1
Power N-CT67^2T3T3	88	3
Power N-CT67^2T5T5	88	2
Power N-CT67^2T6T1	88	2
Power N-CT67^2T7T1	88	3
Power N-CT67^2T7T2	88	1
Power N-CT67^2T7T7	88	1
Power N-CT71^2T2T2	88	2
Power N-CT71^2T3T3	88	2
Power N-CT73^2T3T3	88	6
Power N-CT73^2T4T1	88	1
Power N-CT73^2T4T4	88	1
Power N-CT73^2T5T5	88	2
Power N-CT73^2T6T3	88	2
Power N-CT73^2T6T5	88	4
Power N-CT73^2T7T7	88	4

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT73^2T8T3	88	2
Power N-CT73^2T8T7	88	1
Power N-CT73^2T9T1	88	1
Power N-CT73^2T9T7	88	3
Power N-CT79^2T23T2	88	2
Power N-CT79^2T23T3	88	2
Power N-CT79^2T24T3	88	2
Power N-CT79^2T25T3	88	2
Power N-CT79^2T28T2	88	2
Power N-CT79^2T2T2	88	4
Power N-CT79^2T34T4	88	2
Power N-CT79^2T3T3	88	2
Power N-CT79^2T4T2	88	1
Power N-CT79^2T4T4	88	1
Power N-CT79^2T5T3	88	2
Power N-CT79^2T5T5	88	1
Power N-CT79^2T7T2	88	1
Power N-CT79^2T9T2	88	1
Power N-CT83^2T16T1	88	1
Power N-CT83^2T17T1	88	3
Power N-CT83^2T1T1	88	3
Power N-CT83^2T3T3	88	2
Power N-CT83^2T5T1	88	1
Power N-CT83^2T9T1	88	1
Power N-CT89^2T1T1	88	1
Power N-CT89^2T23T3	88	1
Power N-CT89^2T24T3	88	1
Power N-CT89^2T2T2	88	4
Power N-CT89^2T3T3	88	1
Power N-CT89^2T4T4	88	1
Power N-CT89^2T5T5	88	2
Power N-CT89^2T6T1	88	1
Power N-CT97^2T1T1	88	1
Power N-CT97^2T24T5	88	3
Power N-CT97^2T2T2	88	3
Power N-CT97^2T3T3	88	1
Power N-CT97^2T4T2	88	2
Power N-CT97^2T4T4	88	2
Power N-CT97^2T5T5	88	1
Power N-CT97^2T7T7	88	1
Power N-CT29^2T2	89	6
Power N-CT29^2T3	89	3
Power N-CT29^2T4	89	3
Power N-CT29^2T6	89	3
Power N-CT29^2T9	89	2
Power N-CT59^2T11	89	2
Power N-CT59^2T4	89	1
Power N-CT59^2T5	89	4
Power N-CT59^2T6	89	1
Power N-CT79^2T1	89	2
Power N-CT79^2T10	89	2
Power N-CT79^2T11	89	1
Power N-CT79^2T13	89	3
Power N-CT79^2T15	89	3

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT79^2T16	89	3
Power N-CT79^2T18	89	3
Power N-CT79^2T2	89	3
Power N-CT79^2T23	89	5
Power N-CT79^2T24	89	2
Power N-CT79^2T25	89	2
Power N-CT79^2T38	89	1
Power N-CT79^2T4	89	1
Power N-CT79^2T9	89	1
Power N-CT89^2T1	89	1
Power N-CT89^2T10	89	3
Power N-CT89^2T11	89	4
Power N-CT89^2T12	89	6
Power N-CT89^2T13	89	2
Power N-CT89^2T14	89	4
Power N-CT89^2T15	89	1
Power N-CT89^2T16	89	3
Power N-CT89^2T19	89	2
Power N-CT89^2T20	89	2
Power N-CT89^2T21	89	4
Power N-CT89^2T22	89	3
Power N-CT89^2T23	89	8
Power N-CT89^2T24	89	1
Power N-CT89^2T25	89	6
Power N-CT89^2T26	89	3
Power N-CT89^2T27	89	3
Power N-CT89^2T28	89	2
Power N-CT89^2T29	89	3
Power N-CT89^2T3	89	1
Power N-CT89^2T38	89	2
Power N-CT89^2T4	89	1
Power N-CT89^2T42	89	2
Power N-CT89^2T43	89	2
Power N-CT89^2T44	89	2
Power N-CT89^2T45	89	2
Power N-CT89^2T47	89	2
Power N-CT89^2T48	89	2
Power N-CT89^2T49	89	2
Power N-CT89^2T52	89	2
Power N-CT89^2T53	89	2
Power N-CT89^2T6	89	3
Power N-CT89^2T7	89	7
Power N-CT89^2T8	89	2
Power N-CT89^2T9	89	1
Power N-CT31^2+3	90	3
Power N-CT31^2T1	90	7
Power N-CT31^2T10	90	3
Power N-CT31^2T11	90	3
Power N-CT31^2T12	90	1
Power N-CT31^2T13	90	1
Power N-CT31^2T2	90	6
Power N-CT31^2T3	90	4
Power N-CT31^2T4	90	4
Power N-CT31^2T5	90	5

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT31^2T6	90	5
Power N-CT31^2T7	90	4
Power N-CT31^2T8	90	3
Power N-CT31^2T9	90	1
Power N-CT37^2T1	90	2
Power N-CT37^2T10	90	1
Power N-CT37^2T11	90	1
Power N-CT37^2T12	90	2
Power N-CT37^2T14	90	2
Power N-CT37^2T3	90	2
Power N-CT37^2T4	90	5
Power N-CT37^2T5	90	1
Power N-CT37^2T6	90	1
Power N-CT37^2T7	90	2
Power N-CT37^2T8	90	3
Power N-CT41^2+1	90	4
Power N-CT41^2T1	90	2
Power N-CT41^2T10	90	8
Power N-CT41^2T11	90	2
Power N-CT41^2T12	90	3
Power N-CT41^2T14	90	1
Power N-CT41^2T2	90	6
Power N-CT41^2T3	90	1
Power N-CT41^2T4	90	6
Power N-CT41^2T5	90	1
Power N-CT41^2T6	90	3
Power N-CT41^2T7	90	7
Power N-CT47^2T1	90	7
Power N-CT47^2T11	90	4
Power N-CT47^2T12	90	1
Power N-CT47^2T13	90	2
Power N-CT47^2T14	90	2
Power N-CT47^2T16	90	3
Power N-CT47^2T17	90	1
Power N-CT47^2T18	90	3
Power N-CT47^2T2	90	7
Power N-CT47^2T20	90	3
Power N-CT47^2T21	90	1
Power N-CT47^2T22	90	2
Power N-CT47^2T27	90	1
Power N-CT47^2T28	90	1
Power N-CT47^2T29	90	1
Power N-CT47^2T3	90	5
Power N-CT47^2T4	90	4
Power N-CT47^2T5	90	3
Power N-CT47^2T6	90	7
Power N-CT47^2T7	90	2
Power N-CT47^2T8	90	5
Power N-CT47^2T9	90	5
Power N-CT61^2+1	90	3
Power N-CT61^2+2	90	3
Power N-CT61^2+3	90	3
Power N-CT61^2T1	90	3
Power N-CT61^2T10	90	4
Power N-CT61^2T12	90	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT61^2T13	90	1
Power N-CT61^2T15	90	3
Power N-CT61^2T17	90	1
Power N-CT61^2T3	90	3
Power N-CT61^2T4	90	1
Power N-CT61^2T5	90	7
Power N-CT61^2T6	90	3
Power N-CT61^2T7	90	1
Power N-CT61^2T8	90	7
Power N-CT61^2T9	90	2
Power N-CT67^2T1	90	3
Power N-CT67^2T11	90	1
Power N-CT67^2T12	90	2
Power N-CT67^2T15	90	2
Power N-CT67^2T16	90	4
Power N-CT67^2T18	90	2
Power N-CT67^2T2	90	1
Power N-CT67^2T20	90	2
Power N-CT67^2T26	90	2
Power N-CT67^2T28	90	2
Power N-CT67^2T29	90	2
Power N-CT67^2T31	90	2
Power N-CT67^2T6	90	2
Power N-CT67^2T7	90	5
Power N-CT97^2T15	90	3
Power N-CT97^2T17	90	1
Power N-CT97^2T2	90	3
Power N-CT97^2T21	90	2
Power N-CT97^2T23	90	2
Power N-CT97^2T27	90	3
Power N-CT97^2T29	90	1
Power N-CT97^2T5	90	1
Power N-CT97^2T8	90	5
Power N-CT43^2T1	91	3
Power N-CT43^2T10	91	6
Power N-CT43^2T12	91	4
Power N-CT43^2T13	91	1
Power N-CT43^2T14	91	1
Power N-CT43^2T15	91	3
Power N-CT43^2T2	91	10
Power N-CT43^2T23	91	3
Power N-CT43^2T24	91	3
Power N-CT43^2T25	91	3
Power N-CT43^2T26	91	1
Power N-CT43^2T3	91	5
Power N-CT43^2T4	91	4
Power N-CT43^2T5	91	2
Power N-CT43^2T6	91	3
Power N-CT43^2T7	91	4
Power N-CT43^2T8	91	4
Power N-CT43^2T9	91	5
Power N-CT53^2T1	91	2
Power N-CT53^2T10	91	2
Power N-CT53^2T11	91	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT53^2T14	91	1
Power N-CT53^2T17	91	2
Power N-CT53^2T2	91	2
Power N-CT53^2T23	91	2
Power N-CT53^2T25	91	2
Power N-CT53^2T4	91	6
Power N-CT53^2T5	91	3
Power N-CT53^2T6	91	1
Power N-CT53^2T7	91	6
Power N-CT53^2T8	91	3
Power N-CT53^2T9	91	3
Power N-CT73^2T1	91	2
Power N-CT73^2T10	91	3
Power N-CT73^2T11	91	6
Power N-CT73^2T12	91	1
Power N-CT73^2T14	91	2
Power N-CT73^2T17	91	7
Power N-CT73^2T18	91	4
Power N-CT73^2T19	91	2
Power N-CT73^2T20	91	5
Power N-CT73^2T21	91	4
Power N-CT73^2T22	91	4
Power N-CT73^2T24	91	2
Power N-CT73^2T27	91	2
Power N-CT73^2T28	91	2
Power N-CT73^2T3	91	8
Power N-CT73^2T30	91	2
Power N-CT73^2T32	91	2
Power N-CT73^2T36	91	2
Power N-CT73^2T4	91	1
Power N-CT73^2T42	91	2
Power N-CT73^2T5	91	2
Power N-CT73^2T6	91	5
Power N-CT73^2T7	91	10
Power N-CT73^2T8	91	3
Power N-CT73^2T9	91	6
Power N-CT83^2T1	91	1
Power N-CT83^2T11	91	1
Power N-CT83^2T13	91	3
Power N-CT83^2T15	91	2
Power N-CT83^2T16	91	2
Power N-CT83^2T17	91	3
Power N-CT83^2T18	91	1
Power N-CT83^2T21	91	3
Power N-CT83^2T3	91	1
Power N-CT83^2T31	91	4
Power N-CT83^2T32	91	4
Power N-CT83^2T4	91	1
Power N-CT83^2T49	91	4
Power N-CT83^2T5	91	5
Power N-CT83^2T6	91	3
Power N-CT83^2T7	91	3
Power N-CT83^2T9	91	4
Power N-CT71^2T15T1	92	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT71^2T15T3	92	1
Power N-CT71^2T1T1	92	2
Power N-CT73^2T10T10	92	3
Power N-CT73^2T10T7	92	3
Power N-CT73^2T10T9	92	3
Power N-CT73^2T11T7	92	1
Power N-CT73^2T11T8	92	1
Power N-CT73^2T12S12	92	1
Power N-CT73^2T12T1	92	1
Power N-CT73^2T12T4	92	1
Power N-CT73^2T12T9	92	1
Power N-CT73^2T14T14	92	2
Power N-CT73^2T14T3	92	2
Power N-CT73^2T14T8	92	2
Power N-CT73^2T15S12	92	1
Power N-CT73^2T17T14	92	2
Power N-CT73^2T17T17	92	4
Power N-CT73^2T17T2	92	2
Power N-CT73^2T17T8	92	2
Power N-CT73^2T18T17	92	2
Power N-CT73^2T18T18	92	2
Power N-CT73^2T19S11	92	1
Power N-CT73^2T19S12	92	1
Power N-CT73^2T19T17	92	2
Power N-CT73^2T19T18	92	2
Power N-CT73^2T19T19	92	2
Power N-CT73^2T1T1	92	2
Power N-CT73^2T21T21	92	4
Power N-CT73^2T21T3	92	4
Power N-CT73^2T22T21	92	4
Power N-CT73^2T22T3	92	4
Power N-CT73^2T26S12	92	1
Power N-CT73^2T27S12	92	1
Power N-CT73^2T27T14	92	2
Power N-CT73^2T28T14	92	2
Power N-CT73^2T28T28	92	2
Power N-CT73^2T28T3	92	2
Power N-CT73^2T28T8	92	2
Power N-CT73^2T32T14	92	2
Power N-CT73^2T42T3	92	2
Power N-CT73^2T42T8	92	2
Power N-CT73^2T8S12	92	1
Power N-CT73^2T8T8	92	1
Power N-CT79^2T12T3	92	2
Power N-CT79^2T12T4	92	2
Power N-CT79^2T13T2	92	1
Power N-CT79^2T14T2	92	1
Power N-CT79^2T17T4	92	2
Power N-CT79^2T19T1	92	2
Power N-CT79^2T1T1	92	6
Power N-CT79^2T27T1	92	4
Power N-CT83^2T27T1	92	2
Power N-CT89^2T13T2	92	2
Power N-CT73^2Trin1,4,12	93	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Power N-CT73^2Trin1,9,12	93	1
Power N-CT73^2Trin14,14,27	93	2
Power N-CT73^2Trin2,17,17	93	2
Power N-CT73^2Trin2,17,19	93	2
Power N-CT73^2Trin2,17,20	93	2
Power N-CT73^2Trin2,17,21	93	2
Power N-CT73^2Trin2,2,17	93	2
Power N-CT73^2Trin2,2,19	93	2
Power N-CT73^2Trin2,2,20	93	2
Power N-CT73^2Trin2,8,17	93	2
Power N-CT73^2Trin2,8,19	93	2
Power N-CT73^2Trin2,8,20	93	2
Power N-CT73^2Trin3,14,28	93	2
Power N-CT73^2Trin3,21,21	93	4
Power N-CT73^2Trin3,21,22	93	4
Power N-CT73^2Trin3,22,22	93	4
Power N-CT73^2Trin3,3,14	93	2
Power N-CT73^2Trin3,3,21	93	4
Power N-CT73^2Trin3,3,22	93	4
Power N-CT73^2Trin3,3,28	93	6
Power N-CT73^2Trin3,3,3	93	4
Power N-CT73^2Trin3,3,42	93	2
Power N-CT73^2Trin3,3,8	93	2
Power N-CT73^2Trin3,8,14	93	2
Power N-CT73^2Trin3,8,28	93	2
Power N-CT73^2Trin3,8,42	93	2
Power N-CT73^2Trin4,4,12	93	1
Power N-CT73^2Trin7,7,7	93	4
Power N-CT73^2Trin7,7,8	93	1
Power N-CT73^2Trin7,7,9	93	3
Power N-CT73^2Trin7,8,8	93	1
Power N-CT73^2Trin8,14,27	93	2
Power N-CT73^2Trin8,14,28	93	2
Power N-CT73^2Trin8,17,17	93	2
Power N-CT73^2Trin8,8,17	93	2
Power N-CT73^2Trin8,8,19	93	2
Raaphorst-Moura-Stevens	94	12
Raaphorst-Moura-Stevens fuse	94	8
Raaphorst-Moura-Stevens fuse fuse	94	6
Raaphorst-Moura-Stevens fuse fuse fuse	94	6
Raaphorst-Moura-Stevens fuse fuse fuse fuse	95	5
Raaphorst-Moura-Stevens fuse fuse fuse fuse fuse	95	3
Raaphorst-Moura-Stevens fuse fuse fuse fuse fuse fuse	95	2
Restricted CPHF Ext 4(0, 0) WCS	96	8
Restricted CPHF Ext 4(0, 2) WCS	96	8
Restricted CPHF Ext 4(0, 3) WCS	96	6
Restricted CPHF Ext 4(0, 4) WCS	96	6
Restricted CPHF Ext 4(1, 2) WCS	96	3
Restricted CPHF Ext 4(1, 3) WCS	96	5
Restricted CPHF Ext 4(1, 4) WCS	96	5
Restricted CPHF Ext 4(2, 3) WCS	96	3
Restricted CPHF Ext 4(2, 4) WCS	96	3
Restricted CPHF Ext 5(0, 0) WCS	96	7

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Restricted CPHF Ext 5(0, 2) WCS	96	7
Restricted CPHF Ext 5(0, 3) WCS	96	8
Restricted CPHF Ext 5(0, 4) WCS	96	8
Restricted CPHF Ext 5(0, 5) WCS	96	8
Restricted CPHF Ext 5(1, 2) WCS	96	3
Restricted CPHF Ext 5(1, 3) WCS	96	4
Restricted CPHF Ext 5(1, 4) WCS	96	3
Restricted CPHF Ext 5(1, 5) WCS	96	3
Restricted CPHF Ext 6(0, 0) WCS	96	1
Restricted CPHF Ext 6(0, 2) WCS	96	5
Restricted CPHF Ext 6(0, 3) WCS	96	6
Restricted CPHF Ext 6(0, 4) WCS	96	6
Restricted CPHF Ext 6(0, 5) WCS	96	6
Restricted CPHF Ext 6(0, 6) WCS	96	6
Restricted CPHF Ext 6(1, 2) WCS	96	5
Restricted CPHF Ext 6(1, 3) WCS	96	6
Restricted CPHF Ext 6(1, 4) WCS	96	6
Restricted CPHF Ext 6(1, 5) WCS	96	6
Restricted CPHF Ext 6(1, 6) WCS	96	6
Restricted CPHF Ext 6(2, 3) WCS	96	6
Restricted CPHF Ext 6(2, 4) WCS	96	6
Restricted CPHF Ext 6(2, 5) WCS	96	6
Restricted CPHF Ext 6(2, 6) WCS	96	7
Restricted CPHF Ext 6(3, 4) WCS	96	6
Restricted CPHF Ext 6(3, 5) WCS	96	7
Restricted CPHF Ext 6(3, 6) WCS	96	8
Restricted CPHF Ext 6(4, 5) WCS	96	6
Restricted CPHF Ext 6(4, 6) WCS	96	8
Restricted CPHF Ext 6(5, 6) WCS	96	2
Restricted CPHF Ext 7(0, 0) WCS	96	1
Restricted CPHF Ext 7(0, 2) WCS	96	5
Restricted CPHF Ext 7(0, 3) WCS	96	5
Restricted CPHF Ext 7(0, 4) WCS	96	5
Restricted CPHF Ext 7(0, 5) WCS	96	4
Restricted CPHF Ext 7(0, 6) WCS	96	4
Restricted CPHF Ext 7(0, 7) WCS	96	4
Restricted CPHF Ext 7(1, 2) WCS	96	3
Restricted CPHF Ext 7(1, 3) WCS	96	4
Restricted CPHF Ext 7(1, 4) WCS	96	5
Restricted CPHF Ext 7(1, 5) WCS	96	4
Restricted CPHF Ext 7(1, 6) WCS	96	4
Restricted CPHF Ext 7(1, 7) WCS	96	4
Restricted CPHF Ext 7(2, 3) WCS	96	5
Restricted CPHF Ext 7(2, 4) WCS	96	4
Restricted CPHF Ext 7(2, 5) WCS	96	4
Restricted CPHF Ext 7(2, 6) WCS	96	4
Restricted CPHF Ext 7(2, 7) WCS	96	4
Restricted CPHF Ext 7(3, 4) WCS	96	5
Restricted CPHF Ext 7(3, 5) WCS	96	4
Restricted CPHF Ext 7(3, 6) WCS	96	5
Restricted CPHF Ext 7(3, 7) WCS	96	5
Restricted CPHF Ext 7(4, 5) WCS	96	5
Restricted CPHF Ext 7(4, 6) WCS	96	4
Restricted CPHF Ext 7(4, 7) WCS	96	4
Restricted CPHF Ext 8(0, 0) WCS	96	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Restricted CPHF Ext 8(0, 2) WCS	96	3
Restricted CPHF Ext 8(0, 3) WCS	96	3
Restricted CPHF Ext 8(0, 4) WCS	96	3
Restricted CPHF Ext 8(0, 5) WCS	96	3
Restricted CPHF Ext 8(0, 6) WCS	96	2
Restricted CPHF Ext 8(0, 7) WCS	96	3
Restricted CPHF Ext 8(0, 8) WCS	96	3
Restricted CPHF Ext 8(1, 2) WCS	96	1
Restricted CPHF Ext 8(1, 3) WCS	96	1
Restricted CPHF Ext 8(1, 4) WCS	96	2
Restricted CPHF Ext 8(1, 5) WCS	96	3
Restricted CPHF Ext 8(1, 6) WCS	96	2
Restricted CPHF Ext 8(1, 7) WCS	96	2
Restricted CPHF Ext 8(1, 8) WCS	96	1
Restricted CPHF Ext 8(2, 3) WCS	96	3
Restricted CPHF Ext 8(2, 4) WCS	96	2
Restricted CPHF Ext 8(2, 5) WCS	96	2
Restricted CPHF Ext 8(2, 6) WCS	96	2
Restricted CPHF Ext 8(2, 7) WCS	96	2
Restricted CPHF Ext 8(2, 8) WCS	96	3
Restricted CPHF Ext 8(3, 4) WCS	96	3
Restricted CPHF Ext 8(3, 5) WCS	96	3
Restricted CPHF Ext 8(3, 6) WCS	96	3
Restricted CPHF Ext 8(3, 7) WCS	96	3
Restricted CPHF Ext 8(3, 8) WCS	96	3
Restricted CPHF Ext 8(4, 5) WCS	96	3
Restricted CPHF Ext 8(4, 6) WCS	96	3
Restricted CPHF Ext 8(4, 7) WCS	96	2
Restricted CPHF Ext 8(4, 8) WCS	96	2
Restricted CPHF Ext 9(0, 0) WCS	96	1
Restricted CPHF Ext 9(0, 2) WCS	96	2
Restricted CPHF Ext 9(0, 3) WCS	96	2
Restricted CPHF Ext 9(0, 4) WCS	96	2
Restricted CPHF Ext 9(0, 5) WCS	96	1
Restricted CPHF Ext 9(0, 6) WCS	96	2
Restricted CPHF Ext 9(0, 7) WCS	96	2
Restricted CPHF Ext 9(0, 8) WCS	96	2
Restricted CPHF Ext 9(0, 9) WCS	96	2
Restricted CPHF Ext 9(1, 5) WCS	96	1
Restricted CPHF Ext 9(1, 6) WCS	96	1
Restricted CPHF Ext 9(1, 7) WCS	96	1
Restricted CPHF Ext 9(1, 8) WCS	96	1
Restricted CPHF Ext 9(1, 9) WCS	96	1
Restricted CPHF Ext 9(2, 3) WCS	96	1
Restricted CPHF Ext 9(2, 4) WCS	96	1
Restricted CPHF Ext 9(2, 5) WCS	96	2
Restricted CPHF Ext 9(2, 6) WCS	96	2
Restricted CPHF Ext 9(2, 7) WCS	96	1
Restricted CPHF Ext 9(2, 8) WCS	96	1
Restricted CPHF Ext 9(3, 4) WCS	96	1
Restricted CPHF Ext 9(3, 5) WCS	96	2
Restricted CPHF Ext 9(3, 6) WCS	96	2
Restricted CPHF Ext 9(3, 7) WCS	96	2
Restricted CPHF Ext 9(3, 8) WCS	96	1
Restricted CPHF Ext 9(3, 9) WCS	96	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Restricted CPHF Ext 9(4, 5) WCS	96	1
Restricted CPHF Ext 9(4, 6) WCS	96	1
Restricted CPHF Ext 9(5, 7) WCS	96	1
Restricted CPHF Ext 9(6, 7) WCS	96	1
Restricted CPHF Ext 4(0, 0) WCS fuse	97	6
Restricted CPHF Ext 4(0, 2) WCS fuse	97	6
Restricted CPHF Ext 4(0, 3) WCS fuse	97	5
Restricted CPHF Ext 4(0, 4) WCS fuse	97	5
Restricted CPHF Ext 4(1, 2) WCS fuse	97	2
Restricted CPHF Ext 4(1, 3) WCS fuse	97	3
Restricted CPHF Ext 4(1, 4) WCS fuse	97	3
Restricted CPHF Ext 4(2, 3) WCS fuse	97	2
Restricted CPHF Ext 4(2, 4) WCS fuse	97	2
Restricted CPHF Ext 5(0, 0) WCS fuse	97	3
Restricted CPHF Ext 5(0, 2) WCS fuse	97	2
Restricted CPHF Ext 5(0, 3) WCS fuse	97	2
Restricted CPHF Ext 5(0, 4) WCS fuse	97	2
Restricted CPHF Ext 5(0, 5) WCS fuse	97	2
Restricted CPHF Ext 6(0, 2) WCS fuse	97	1
Restricted CPHF Ext 6(0, 3) WCS fuse	97	2
Restricted CPHF Ext 6(0, 4) WCS fuse	97	2
Restricted CPHF Ext 6(0, 5) WCS fuse	97	2
Restricted CPHF Ext 6(0, 6) WCS fuse	97	2
Restricted CPHF Ext 6(1, 2) WCS fuse	97	2
Restricted CPHF Ext 6(1, 3) WCS fuse	97	2
Restricted CPHF Ext 6(1, 4) WCS fuse	97	2
Restricted CPHF Ext 6(1, 5) WCS fuse	97	2
Restricted CPHF Ext 6(1, 6) WCS fuse	97	2
Restricted CPHF Ext 6(2, 3) WCS fuse	97	2
Restricted CPHF Ext 6(2, 4) WCS fuse	97	2
Restricted CPHF Ext 6(2, 5) WCS fuse	97	2
Restricted CPHF Ext 6(2, 6) WCS fuse	97	3
Restricted CPHF Ext 6(3, 4) WCS fuse	97	3
Restricted CPHF Ext 6(3, 5) WCS fuse	97	3
Restricted CPHF Ext 6(3, 6) WCS fuse	97	3
Restricted CPHF Ext 6(4, 5) WCS fuse	97	3
Restricted CPHF Ext 6(4, 6) WCS fuse	97	4
Restricted CPHF Ext 6(5, 6) WCS fuse	97	2
Restricted CPHF Ext 7(3, 6) WCS fuse	97	1
Restricted CPHF Ext 7(3, 7) WCS fuse	97	1
Restricted CPHF Ext 7(4, 5) WCS fuse	97	1
Restricted CPHF Ext 7(4, 6) WCS fuse	97	1
Restricted CPHF Ext 7(4, 7) WCS fuse	97	1
Restricted CPHF Ext 4(0, 0) WCS fuse fuse	98	2
Restricted CPHF Ext 4(0, 0) WCS fuse fuse fuse	98	1
Restricted CPHF Ext 4(0, 2) WCS fuse fuse	98	2
Restricted CPHF Ext 4(0, 2) WCS fuse fuse fuse	98	1
Restricted CPHF Ext 4(0, 3) WCS fuse fuse	98	2
Restricted CPHF Ext 4(0, 3) WCS fuse fuse fuse	98	1
Restricted CPHF Ext 4(0, 4) WCS fuse fuse	98	2
Restricted CPHF Ext 4(0, 4) WCS fuse fuse fuse	98	1
Restricted CPHF Ext 4(2, 3) WCS fuse fuse	98	1
Restricted CPHF Ext 4(2, 4) WCS fuse fuse	98	1
Restricted CPHF Ext 5(0, 0) WCS fuse fuse	98	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Restricted CPHF Ext 6(2, 6) WCS fuse fuse	98	1
Restricted CPHF Ext 6(3, 4) WCS fuse fuse	98	1
Restricted CPHF Ext 6(3, 5) WCS fuse fuse	98	1
Restricted CPHF Ext 6(3, 6) WCS fuse fuse	98	1
Restricted CPHF Ext 6(4, 5) WCS fuse fuse	98	1
Restricted CPHF Ext 6(4, 6) WCS fuse fuse	98	1
Restricted CPHF Ext 10(0, 2) WCS	99	1
Restricted CPHF Ext 10(0, 3) WCS	99	1
Restricted CPHF Ext 10(0, 4) WCS	99	2
Restricted CPHF Ext 10(0, 5) WCS	99	1
Restricted CPHF Ext 10(0, 6) WCS	99	1
Restricted CPHF Ext 10(0, 7) WCS	99	1
Restricted CPHF Ext 10(0, 8) WCS	99	2
Restricted CPHF Ext 10(0, 9) WCS	99	2
Restricted CPHF Ext 10(0,10) WCS	99	2
Restricted CPHF Ext 10(1, 6) WCS	99	1
Restricted CPHF Ext 10(1, 7) WCS	99	1
Restricted CPHF Ext 10(2, 3) WCS	99	1
Restricted CPHF Ext 10(2, 4) WCS	99	2
Restricted CPHF Ext 10(2, 5) WCS	99	2
Restricted CPHF Ext 10(2, 6) WCS	99	2
Restricted CPHF Ext 10(2, 7) WCS	99	2
Restricted CPHF Ext 10(2, 8) WCS	99	1
Restricted CPHF Ext 10(2, 9) WCS	99	1
Restricted CPHF Ext 10(3, 5) WCS	99	2
Restricted CPHF Ext 10(3, 6) WCS	99	1
Restricted CPHF Ext 10(3, 7) WCS	99	2
Restricted CPHF Ext 10(3, 8) WCS	99	1
Restricted CPHF Ext 10(3, 9) WCS	99	1
Restricted CPHF Ext 10(5, 6) WCS	99	1
Restricted CPHF Ext 10(6, 7) WCS	99	1
Restricted CPHF Ext 10(7, 8) WCS	99	1
Restricted CPHF Ext 10(7, 9) WCS	99	1
Restricted CPHF Ext 11(0, 2) WCS	99	1
Restricted CPHF Ext 11(0, 3) WCS	99	1
Restricted CPHF Ext 11(0, 4) WCS	99	1
Restricted CPHF Ext 11(0, 5) WCS	99	1
Restricted CPHF Ext 11(0, 6) WCS	99	1
Restricted CPHF Ext 11(0, 8) WCS	99	1
Restricted CPHF Ext 11(0, 9) WCS	99	1
Restricted CPHF Ext 11(0,10) WCS	99	1
Restricted CPHF Ext 11(0,11) WCS	99	1
Restricted CPHF Ext 11(1,11) WCS	99	1
Restricted CPHF Ext 11(2, 4) WCS	99	1
Restricted CPHF Ext 11(2, 5) WCS	99	1
Restricted CPHF Ext 11(2, 6) WCS	99	1
Restricted CPHF Ext 11(3, 4) WCS	99	1
Restricted CPHF Ext 11(3, 5) WCS	99	1
Restricted CPHF Ext 11(5, 6) WCS	99	1
Restricted CPHF Ext 11(5, 7) WCS	99	1
Restricted CPHF Ext 11(5, 8) WCS	99	1
Restricted CPHF Ext 11(6, 8) WCS	99	1
Restricted CPHF Ext 11(7, 9) WCS	99	1
Restricted CPHF Ext 11(7,11) WCS	99	1

Table 14: The source entries arranged by clusters (continued)

	clusterId	frequency
Restricted CPHF Ext 11(8, 9) WCS	99	1
Restricted CPHF Ext 12(0, 7) WCS	99	1
Restricted CPHF Ext 12(0, 8) WCS	99	1
Restricted CPHF Ext 12(0, 9) WCS	99	1
Restricted CPHF Ext 12(0,12) WCS	99	1
Restricted CPHF Ext 12(2, 3) WCS	99	1
Restricted CPHF Ext 12(2, 5) WCS	99	1
Restricted CPHF Ext 12(2, 6) WCS	99	1
Restricted CPHF Ext 12(2, 7) WCS	99	1
Restricted CPHF Ext 12(3, 4) WCS	99	1
Restricted CPHF Ext 12(4,12) WCS	99	1
Restricted CPHF Ext 13(6, 7) WCS	99	1
Restricted CPHF Sim Annealing (TJ-IM)	100	117
Restricted CPHF Sim Annealing (TJ-IM) fuse	100	5
SAT Local Search (Hnich et al.)	101	1
SCPHF (Sherwood-Martirosyan-Colbourn)	102	1
SCPHF (Sherwood-Martirosyan-Colbourn) fuse postop NCK	102	13
SCPHF CLS - other	103	2
SCPHF CLS - other fuse	103	1
SCPHF Conditional Expectation (CLS)	104	13
SCPHF Conditional Expectation (CLS) fuse	104	5
SCPHF Conditional Expectation (CLS) postop NCK	104	1
SCPHF LFSR (TJ-IM)	105	9
SCPHF LFSR (TJ-IM) fuse	105	8
SCPHF LFSR (TJ-IM) fuse fuse	105	6
SCPHF LFSR (TJ-IM) fuse fuse fuse	105	6
SCPHF LFSR (TJ-IM) fuse fuse fuse fuse	106	5
SCPHF LFSR (TJ-IM) fuse fuse fuse fuse fuse	106	3
SCPHF LFSR (TJ-IM) fuse fuse fuse fuse fuse fuse	106	2
SCPHF Tabu Search (Walker-Colbourn)	107	2
SCPHF Tabu Search (Walker-Colbourn) fuse postop NCK	107	12
simulated annealing (AG-TJ-H)	108	9
simulated annealing (CKRS)	108	8
simulated annealing (Cohen)	108	7
simulated annealing (TJ-RT)	108	30
simulated annealing (Torres-Jimenez)	109	145
simulated annealing (Torres-Jimenez) postop NCK	109	2
SIPO (Wagner-Kampel-Simos)	110	43
tabu search (Nurmela)	111	4
tabu search (Rouse-Lamarre)	111	1
tabu search (Zekaoui)	111	1

References

Avila-George, H., Torres-Jimenez, J., Gonzalez-Hernandez, L., and Hernández, V. (2013), "Metaheuristic approach for constructing functional test-suites," *IET Software*, John Wiley & Sons, Ltd, 7, 104–117. https://doi.org/10.1049/iet-sen.2012.0074.

Avila-George, H., Torres-Jimenez, J., and Hernández, V. (2012), "New Bounds for Ternary Covering Arrays Using a Parallel Simulated Annealing," *Mathematical Problems in Engineering*, (J. G. Carlsson,

ed.), 2012, 897027. https://doi.org/10.1155/2012/897027.
Avila-George, H., Torres-Jimenez, J., and Izquierdo-Marquez, I. (2018), "Improved pairwise test suites for non-prime-power orders," *IET Software*, 12, 215–224. https://doi.org/10.1049/iet-sen.2017.0107.
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.
Brent, R. P. (2004), "Note on Marsaglia's Xorshift Random Number Generators," *Journal of Statistical Software*, 11. https://doi.org/10.18637/jss.v011.i05.
Bush, K. A. (1952), "Orthogonal Arrays of Index Unity," *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, 23, 426–434. https://doi.org/10.1214/aoms/1177729387.
Calvagna, A., and Gargantini, A. (2012), "T-wise combinatorial interaction test suites construction based on coverage inheritance," *Software Testing, Verification and Reliability*, Wiley, 22, 507–526. https://doi.org/10.1002/stvr.466.
Carnell, R. (2024), "lhs: Latin Hypercube Samples." https://doi.org/10.32614/CRAN.package.lhs.
Chateauneuf, M. A., Colbourn, C. J., and Kreher, D. L. (1999), "Covering Arrays of Strength Three," 16, 235–242.
Chateauneuf, M., and Kreher, D. L. (2002), "On the state of strength-three covering arrays," *Journal of Combinatorial Designs*, 10, 217–238. https://doi.org/10.1002/jcd.10002.
Cohen, M. B., Colbourn, C. J., and Ling, A. C. H. (2003), "Augmenting simulated annealing to build interaction test suites," in *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, Denver, Colorado, USA: IEEE, pp. 394–405. https://doi.org/10.1109/ISSRE.2003.1251061.
Cohen, M. B., Colbourn, C. J., and Ling, A. C. H. (2008), "Constructing strength three covering arrays with augmented annealing," *Discrete Mathematics*, 308, 2709–2722. https://doi.org/10.1016/j.disc.2006.06.036.
Colbourn, C. (2009), "Distributing hash families and covering arrays," *J. Combin. Inf. Syst. Sci.*, 34, 113–126.
Colbourn, C. J. (2004), "Combinatorial aspects of covering arrays," *Matematiche (Catania)*, 59, 125–172.
Colbourn, C. J. (2008), "Strength two covering arrays: Existence tables and projection," *Discrete Mathematics*, 308, 772–786. https://doi.org/10.1016/j.disc.2007.07.050.
Colbourn, C. J. (2010), "Covering arrays from cyclotomy," *Designs, Codes and Cryptography*, 55, 201–219. https://doi.org/10.1007/s10623-009-9333-8.
Colbourn, C. J. (2011), "Covering arrays and hash families," in *Information security and related combinatorics, in: NATO science for peace and security series -D: Information and communication security*, pp. 99–136.
Colbourn, C. J. (2014a), "Conditional expectation algorithms for covering arrays," *J. Combin. Math. Combin. Comput.*, 90, 97–115.
Colbourn, C. J. (2014b), "Covering arrays, augmentation, and quilting arrays," *Discrete Mathematics, Algorithms and Applications*, 06, 1450034. https://doi.org/10.1142/S1793830914500347.
Colbourn, C. J. (2015), "Suitable Permutations, Binary Covering Arrays, and Paley Matrices," in *Algebraic Design Theory and Hadamard Matrices*, Springer Proceedings in Mathematics & Statistics, ed. C. J. Colbourn, Cham: Springer International Publishing, pp. 29–42. https://doi.org/10.1007/978-3-319-17729-8_3.
Colbourn, C. J. (n.d.). "Covering array tables: 2<=v<=25, 2 <=t<=6, t<=k<=10000, 2005–23," Available athttps://www.public.asu .edu /~ccolbou/src /tabby.
Colbourn, C. J., and Dinitz, J. H. (eds.) (2007), *Handbook of combinatorial designs*, Discrete mathematics and its applications, Boca Raton, FL: Chapman & Hall/Taylor & Francis.
Colbourn, C. J., Dougherty, R. E., and Horsley, D. (2019), "Distributing hash families with few rows," *Theoretical Computer Science*, 800, 31–41. https://doi.org/10.1016/j.tcs.2019.10.014.
Colbourn, C. J., and Kéri, G. (2009), "Binary Covering Arrays and Existentially Closed Graphs," in *Coding and Cryptology*, Lecture Notes in Computer Science, eds. Y. M. Chee, C. Li, S. Ling, H. Wang, and C. Xing, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 22–33. https://doi.org/10.1007/978-3-642-01877-0_3.
Colbourn, C. J., 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.
Colbourn, C. J., and Lanus, E. (2018a), "Subspace restrictions and affine composition for covering perfect hash families," *The Art of Discrete and Applied Mathematics*, 1, #P2.03. https://doi.org/10.26493/2590-9770.1220.3a1.

Colbourn, C. J., Lanus, E., and Sarkar, K. (2018), "Asymptotic and constructive methods for covering perfect hash families and covering arrays," *Designs, Codes and Cryptography*, 86, 907–937. https://doi.org/10.1007/s10623-017-0369-x.

Colbourn, C. J., Martirosyan, S. S., Mullen, G. L., Shasha, D., Sherwood, G. B., and Yucas, J. L. (2006), "Products of mixed covering arrays of strength two," *Journal of Combinatorial Designs*, 14, 124–138. https://doi.org/10.1002/jcd.20065.

Colbourn, C. J., Martirosyan, S. S., Van Trung, T., and Walker, R. A. (2006), "Roux-type constructions for covering arrays of strengths three and four," *Designs, Codes and Cryptography*, 41, 33–57. https://doi.org/10.1007/s10623-006-0020-8.

Colbourn, C. J., and Torres-Jimenez, J. (2010), "Heterogeneous hash families and covering arrays," in *Contemporary Mathematics*, eds. A. A. Bruen and D. L. Wehlau, Providence, Rhode Island: American Mathematical Society, pp. 3–15. https://doi.org/10.1090/conm/523/10309.

Colbourn, C. J., and Torres-Jimenez, J. (2013), "Profiles of covering arrays of strength two," *Journal of Algorithms and Computation*, 44, 31–59. https://doi.org/10.22059/jac.2013.7914.

Colbourn, C. J., and Zhou, J. (2012), "Improving Two Recursive Constructions for Covering Arrays," *Journal of Statistical Theory and Practice*, 6, 30–47. https://doi.org/10.1080/15598608.2012.647489.

Colbourn, C., and Lanus, E. (2018b), "Subspace restrictions and affine composition for covering perfect hash families," *Art Discrete Appl. Math*, 1, 2 03. https://doi.org/10.26493/2590-9770.1220.3a1.

Dwyer, A. (2024), "aadwyer/CA_Database."

Hartman, A. (2005), "Software and Hardware Testing Using Combinatorial Covering Suites," in *Graph Theory, Combinatorics and Algorithms*, Operations Research/Computer Science Interfaces Series, eds. M. C. Golumbic and I. B.-A. Hartman, New York: Springer-Verlag, pp. 237–266. https://doi.org/10.1007/0-387-25036-0_10.

Hedayat, A. S., Sloane, N. J. A., and Stufken, J. (1999), *Orthogonal Arrays*, Springer Series in Statistics, New York, NY: Springer New York. https://doi.org/10.1007/978-1-4612-1478-6.

Hnich, B., Prestwich, S. D., Selensky, E., and Smith, B. M. (2006), "Constraint Models for the Covering Test Problem," *Constraints*, 11, 199–219. https://doi.org/10.1007/s10601-006-7094-9.

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

Ji, L., and Yin, J. (2010), "Constructions of new orthogonal arrays and covering arrays of strength three," *Journal of Combinatorial Theory, Series A*, 117, 236–247. https://doi.org/10.1016/j.jcta.2009.06.002.

JMP Statistical Discovery LLC (2024), "Design of experiments guide," Section "Covering Arrays."

Katona, G. O. H. (1973), "Two applications (for search theory and truth functions) of Sperner type theorems," *Periodica Mathematica Hungarica*, 3, 19–26. https://doi.org/10.1007/BF02018457.

Kleitman, D. J., and Spencer, J. (1973), "Families of k-independent sets," *Discrete Mathematics*, 6, 255–262. https://doi.org/10.1016/0012-365X(73)90098-8.

Kokkala, J. I., Meagher, K., Naserasr, R., Nurmela, K. J., Östergård, P. R. J., and Stevens, B. (2018), "Classification of small strength-2 covering arrays," Zenodo. https://doi.org/10.5281/zenodo.1476059.

Kokkala, J. I., Meagher, K., Naserasr, R., Nurmela, K. J., Östergård, P. R. J., and Stevens, B. (2020), "On the structure of small strength-2 covering arrays," *Journal of Combinatorial Designs*, 28, 5–24. https://doi.org/10.1002/jcd.21671.

Lobb, J. R., Colbourn, C. J., Danziger, P., Stevens, B., and Torres-Jimenez, J. (2012), "Cover starters for covering arrays of strength two," *Discrete Mathematics*, 312, 943–956. https://doi.org/10.1016/j.disc.2011.10.026.

Maity, S., Akhtar, Y., Chandrasekharan, R. C., and Colbourn, C. J. (2018), "Improved Strength Four Covering Arrays with Three Symbols," *Graphs and Combinatorics*, 34, 223–239. https://doi.org/10.1007/s00373-017-1861-9.

Martirosyan, Sosina, and Colbourn, Charles (2005), "Recursive constructions of covering arrays," in *ALCOMA '05*, Bayreuther Mathematische Schriften, Thurnau, Germany: Universität Bayreuth, pp. 266–275.

Martirosyan, S., and Trung, T. V. (2004), "On t-Covering Arrays," *Designs, Codes and Cryptography*, 32, 323–339. https://doi.org/10.1023/B:DESI.0000029232.40302.6d.

Martirosyan, S., and Van Trung, T. (2008), "Explicit constructions for perfect hash families," *Designs, Codes and Cryptography*, 46, 97–112. https://doi.org/10.1007/s10623-007-9138-6.

Meagher, K. (2005b), "Covering Arrays on Graphs: Qualitative Independence Graphs and Extremal Set Partition Theory," Ottawa: University of Ottawa.

Meagher, K. (2005a), *Group Construction of Covering Arrays | Part 2*, Unpublished, Ottawa.

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.

Moura, L., Stardom, J., Stevens, B., and Williams, A. (2003), "Covering arrays with mixed alphabet sizes," *Journal of Combinatorial Designs*, Wiley, 11, 413–432. https://doi.org/10.1002/jcd.10059.

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.

"NIST Covering Array Tables" (n.d.). Available athttps://math.nist.gov/coveringarrays/.

Nurmela, K. J. (2004), "Upper bounds for covering arrays by tabu search," *Discrete Applied Mathematics*, 138, 143–152. https://doi.org/10.1016/S0166-218X(03)00291-9.

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

Sarkar, K., and Colbourn, C. J. (2019), "Two-stage algorithms for covering array construction," *Journal of Combinatorial Designs*, 27, 475–505. https://doi.org/10.1002/jcd.21657.

Sherwood, G. B., Martirosyan, S. S., and Colbourn, C. J. (2006), "Covering arrays of higher strength from permutation vectors," 14, 202–213. https://doi.org/10.1002/jcd.20067.

Shokri, K., and Moura, L. (2025), "New Families of Strength-3 Covering Arrays Using Linear Feedback Shift Register Sequences," *Journal of Combinatorial Designs*, 33, 156–171. https://doi.org/10.1002/jcd.21963.

Sloane, N. J. A. (1993), "Covering arrays and intersecting codes," *Journal of Combinatorial Designs*, 1, 51–63. https://doi.org/10.1002/jcd.3180010106.

Torres-Jimenez, J. (n.d.). "Jose Torres-Jimenez Homepage," *Covering Arrays*, Available athttps://www.tamps.cinvestav.mx/~oc/.

Torres-Jimenez, J., Acevedo-Juárez, B., and Avila-George, H. (2021), "Covering array EXtender," *Applied Mathematics and Computation*, 402, 126122. https://doi.org/10.1016/j.amc.2021.126122.

Torres-Jimenez, J., and Avila-George, H. (2016), "Search-Based Software Engineering to Construct Binary Test-Suites," in *Trends and Applications in Software Engineering*, Advances in Intelligent Systems and Computing, eds. J. Mejia, M. Munoz, Á. Rocha, and J. Calvo-Manzano, Cham: Springer International Publishing, pp. 201–212. https://doi.org/10.1007/978-3-319-26285-7_17.

Torres-Jimenez, J., Avila-George, H., and Izquierdo-Marquez, I. (2017), "A two-stage algorithm for combinatorial testing," *Optimization Letters*, 11, 457–469. https://doi.org/10.1007/s11590-016-1012-x.

Torres-Jimenez, J., and Izquierdo-Marquez, I. (2018), "Covering arrays of strength three from extended permutation vectors, Des," *Codes Cryptogr*, 86, 2629–2643. https://doi.org/10.1007/s10623-018-0465-6.

Torres-Jimenez, J., and Izquierdo-Marquez, I. (2020), "Improved covering arrays using covering perfect hash families with groups of restricted entries," *Applied Mathematics and Computation*, 369, 124826. https://doi.org/10.1016/j.amc.2019.124826.

Torres-Jimenez, J., Izquierdo-Marquez, I., and Avila-George, H. (2019), "Methods to construct uniform covering arrays," *IEEE access : practical innovations, open solutions*, 7, 42774–42797. https://doi.org/10.1109/ACCESS.2019.2907057.

Torres-Jimenez, J., Izquierdo-Marquez, I., Kacker, R. N., and Richard Kuhn, D. (2015), "Tower of covering arrays," *Discrete Applied Mathematics*, 190-191, 141–146. https://doi.org/10.1016/j.dam.2015.03.010.

Torres-Jimenez, J., and Rodriguez-Cristerna, A. (2017), "Metaheuristic post-optimization of the NIST repository of covering arrays," *CAAI Transactions on Intelligence Technology*, 2, 31–38. https://doi.org/10.1016/j.trit.2016.12.006.

Torres-Jimenez, J., and Rodriguez-Tello, E. (2012), "New bounds for binary covering arrays using simulated annealing," *Information Sciences*, 185, 137–152. https://doi.org/10.1016/j.ins.2011.09.020.

Wagner, M., Colbourn, C. J., and Simos, D. E. (2022), "In-Parameter-Order strategies for covering perfect hash families," *Applied Mathematics and Computation*, 421, 126952. https://doi.org/10.1016/j.amc.2022.126952.

Wagner, M., Kampel, L., and Simos, D. E. (2021), "Heuristically Enhanced IPO Algorithms for Covering Array Generation," in *Combinatorial Algorithms*, Lecture Notes in Computer Science, eds. P. Flocchini and L. Moura, Cham: Springer International Publishing, pp. 571–586. https://doi.org/10.1007/978-3-030-79987-8_40.

Wagner, M., Kleine, K., Simos, D. E., Kuhn, R., and Kacker, R. (2020), "CAGEN: A fast combinatorial test generation tool with support for constraints and higher-index arrays," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 191–200. https://doi.org/10.1109/ICSTW50294.2020.00041.

Walker II, R. A., and Colbourn, C. J. (2009), "Tabu search for covering arrays using permutation vectors," *Journal of Statistical Planning and Inference*, 139, 69–80. https://doi.org/10.1016/j.jspi.2008.05.020.

Younis, M. I., and Zamli, K. Z. (2011), "MIPOG - An Efficient t-Way Minimization Strategy for Combinatorial Testing," *International Journal of Computer Theory and Engineering*, 388–397. https://doi.org/10.7763/IJCTE.2011.V3.337.

Zekaoui, L. (2006), "Mixed covering arrays on graphs and tabu search algorithms," Université d'Ottawa / University of Ottawa.

Zhang, J., Zhang, Z., and Ma, F. (2014), *Automatic Generation of Combinatorial Test Data*, SpringerBriefs in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-43429-1.