Towards Accessibility of Covering Arrays for Practitioners of Combinatorial Testing

Ulrike Grömping
FB II
BHT - Berliner Hochschule für Technik
Berlin, Germany
ulrike.groemping@bht-berlin.de

***Abstract*—Combinatorial testing can be useful not only for large software-related testing efforts. While it seems worthwhile to evaluate its benefits for more general applications, lack of immediate availability of suitable arrays is a major impediment against its wider use by practitioners. This paper discusses requirements for making covering arrays (CAs), a key tool in combinatorial testing, accessible to practitioners of various application areas in a way that satisfies their needs. It is argued that, by fulfilling such requirements, the expert community also improves exchange of results among its members. The paper is thus intended as a call for action for the combinatorial testing community to improve the sharing of CA constructions.**

Keywords—Covering arrays, catalogue, mathematical constructions

I. Introduction

The author first encountered covering arrays (CAs) in a request by an engineering quality manager, who brought up the need to cover all combinations for a set of factors in a large vehicle system. She had to find out that – contrary to orthogonal arrays for which there are various catalogues as well as implemented construction algorithms in a variety of software tools – CAs are not as easily available to non-expert practitioners. This paper is a call for action to improve the availability of good CAs for practitioners who do not have the willingness and/or ability needed for obtaining an array given the current state of available resources. Such practitioners will be called "uninitiated practitioners" (UIPs). It is anticipated that catering for UIPs will also improve exchange and reuse of information within the combinatorial testing community.

CAs are collections of N runs (rows) for k factors (columns) in $v_1, \ldots, v_k$ levels; if $v_1 = \ldots = v_k = v$, a CA is called "uniform", otherwise "mixed level" (MCA). The central property of a CA is its strength: A CA of strength t contains (covers) each conceivable tuple of any set of t columns at least once. Uniform CAs are denoted as CA(N, t, k, v), and the theoretically possible minimum size of a uniform CA of strength t for k columns in v levels is denoted as CAN(t, k, v); CAN(t, k, v) is often unknown. A CA is optimal, if its size N achieves the theoretical minimum, and it is best-known, if there is no currently known CA for the same setting that needs fewer runs. Tables curated by Colbourn ([1]) list the current best-known run sizes (i.e., upper bounds for CAN(t, k, v)) for many uniform settings, grouped by (t, v) combinations for t=2,…,6 and v=2,…,25. Unfortunately, the tables come without the best-known uniform CAs.

There are mathematical constructions, mostly – but not only – for uniform CAs, and search strategies that are much more flexible and can create arbitrary MCAs; unfortunately, the latter are also much slower. Torres-Jiménez et al. [2] review mathematical methods, Leithner et al. [3] provide a comprehensive overview of search methods. Most mathematical constructions and search methods require mathematical and/or computational background and expertise and are thus not easily workable for UIPs.

This paper aims for instigating activities that improve the availability of CAs for practitioners from all fields who are not themselves experts in CA creation. One part of the activities is an R package by the author that will implement mathematical CA constructions and various utilities for creating and using CAs. Comments on all aspects of this paper and the planned R package are invited. Section II describes the current status on CAs from the point of view of an interested non-expert and considers requirements that should be addressed, with special emphasis on supporting UIPs. Section III discusses activities around the accessible implementation of mathematical CA constructions. Section IV calls for community action in order to preserve and enhance the community knowledge base inherent in the Colbourn tables ([1]), including the provision of precomputed CAs, where adequate.

II. Status and Desirable Improvements

The idea of CAs is simple enough. In an *ideal* world, a UIP hears about CAs, thinks about a useful application and runs a web search. The search leads to a place or tool where a good or even optimal CA can be downloaded, or where there are pointers to constructing such an array with little effort and in short time using a familiar and readily available software.

In the real world, the UIP will likely find the NIST [4] library of precomputed CAs – which were obtained via a search algorithm in 2008 – as well as the Colbourn [1] tables of current upper bounds for CAN(t, k, v). Frustration will arise when realising that a further catalogue of CAs referenced from the NIST website is unavailable (but see Section IV), and that the available CAs from the NIST library are often relevantly larger than the best-known CAs, which are neither directly available nor referenced in a way that can be understood by a UIP. The UIP will also find software tools for the creation of CAs through search algorithms, among them CAgen [5] with the possibility of free direct web access and JMPPro [6] as a commercial statistical software with a free academic license. A very persistent UIP may find a documentation page for CAs within version 10.5 of the open source SageMath system ([7]), which offers a limited selection of mathematical CA constructions (see also Section III). None of the finds makes it easy for the UIP to obtain a satisfactory array fast, although tools like CAgen or JMPPro (provided it is available) may cater to the more enthusiastic UIP, perhaps also SageMath for someone who already uses its development version. In summary, for relevant problem sizes, availability of CAs for UIPs is currently quite limited. Moreover, various best-known CAs behind the Colbourn tables ([1]) are not even easily available for experts.

The most obvious immediate benefit for the UIP can be obtained by providing precomputed current-best – or at least "good" – CAs. The necessary actions for providing such precomputed CAs – and for keeping the Colbourn tables ([1]) up-to-date – are discussed in Section IV. It is not possible to store arrays for all conceivable needs, and in some cases, it may be much more efficient to code a mathematical construction than to store all conceivable arrays that can be constructed by it. Thus, implementing mathematical constructions is certainly worthwhile. Where they exist, mathematical constructions can yield much smaller CAs than search algorithms, and do so fast at the same time. For example, a CA(1051, 4, 1051, 3) from a mathematical construction (see Section III: cyclotomy construction type 1 with prime 1051) was matched by a CA(30777, 4, 1051, 3) constructed with JMPPro, the only search tool accessible to the author that yielded a result in reasonable time for that setting (given the available Windows computer). Activities for the implementation of mathematical constructions are discussed in Section III.

Besides the specific aspects to be discussed in Sections III and IV, there are various further relevant aspects that are less universal but nevertheless important. Even a UIP may want a CA for a situation with mixed levels for which a precomputed array is not available and a search algorithm is – at least at present – the only option. It is therefore highly desirable to provide accessible tools with which a tailor-made CA can be created. As long as such a tool is easily available and provides an array with reasonable effort and within reasonable time – perhaps more important, within foreseeable time – a UIP can benefit from using it. The author has experience as a UIP attempting to use CAgen ([5]), JMPPro ([6]) and CTWedge ([8]). CAgen is the only tool that allows the creation of small but relevant designs directly on the web with immediate access for any interested user. CTWedge only runs extremely small requests on the web and defers larger requests to batch queues, which could be a good idea – because it is very annoying to close a browser window by mistake after substantial run time – but did not work well for the author (neither result nor error message, after many weeks). For free (web) versions of CA creation tools – which are of course a very friendly service by the groups who provide them – it would be very desirable to provide information that raises realistic expectations about the size of use cases covered and of run times to be expected. Without those, UIPs are likely to expect results too fast and for too large problems, and might waste resources in vain, or be so disillusioned that they abandon the intention of using CAs altogether. Besides informing UIPs about permissible problem sizes and expected run times, it would also be very useful to make sophisticated search tools like CAgen or CTWedge accessible via software familiar to UIPs, e.g., R or Python. This could be achieved via an API: If such APIs are reasonably standardized, third-party software can create search commands and ideally hand them over to the search tool and also receive the resulting CA. The assessment of the coverage properties of a CA is of substantial interest (see, e.g., [9]) and should be made available in any software that deals with CAs; for large CAs, it can be very computer-intensive. Various further aspects that are very relevant for the practical use of CAs (constraints, variable strength, don't care values, …) are ignored in this paper, for the sake of brevity.

III. Implementation of Mathematical Constructions

The Colbourn tables ([1]) as the go-to place for the smallest known upper bounds for CAN(t, k, v) are a good starting point for choosing algorithms to be implemented. Torrez-Jiménez et al. ([2], their Section IX) give an overview of algorithms behind the table entries: important mathematical (as opposed to search type or metaheuristic / post-optimization) construction elements appear to be covering perfect hash families (CPHF, which themselves need to be constructed or available) for all kinds of strengths and numbers of levels, constructions based on cyclotomy, group-based constructions and constructions based on starter vectors, direct product and power-based constructions and other recursive constructions. It is natural to inspect which constructions either yield optimal CAs or achieve large improvements over available catalogued arrays. The strength 2 construction of [10] and [11] for 2-level CAs is proven to be optimal and easy to implement and thus a natural candidate for high priority implementation, e.g., most likely in JMPPro ([6]).

Mathematical constructions should be implemented in software environments used by UIPs or their advisors. Open source implementations achieve the greatest impact. There are two ongoing activities for implementing mathematical CA constructions in open source software: the author's R package CAs (on GitHub at https://github.com/ugroempi/CAs as an early development version), and SageMath version 10.5 and higher ([7], current activities can be found on GitHub, e.g., https://github.com/sagemath/sage/issues/38603). Both already contain the optimal construction of [10] and [11]. A suitable software environment should also offer the appropriate infrastructure for CA constructions, for example, Galois field arithmetic or covering perfect hash families. The author is not aware of current efforts on implementing mathematical algorithms for CA constructions besides the recent efforts in SageMath and R, neither open source nor commercial (except for the afore-mentioned optimal 2-level CAs). It should be attempted to leverage synergies between the efforts in SageMath and R. Easy availability of CAs from mathematical constructions might also lead to improved search-based designs, if search algorithms can incorporate large math-based arrays.

Beyond uniform CAs, it is desirable to also implement mathematical constructions for MCAs, e.g., by Akhtar et al. [12]. These would be valuable additions to the toolbox. Given that the availability of mathematical MCA constructions is likely limited, methods for obtaining MCAs by combining uniform CAs or by expanding a uniform CA are of interest. A few columns with fewer levels than the others can be obtained by changing selected levels in the respective columns.

The rest of this section exemplifies the implementation of a mathematical construction, the cyclotomy-based CAs by Colbourn [13], which will also serve as an example in the discussion of useful enhancements for the Colbourn tables ([1])

in Section IV. Colbourn table entries with source “Cyclotomy (Colbourn)” or “Cyclotomy (Torres-Jimenez)” refer to “Constructions 4” from [13], which provides a total of eight related constructions: 1, 2, 3, 3a, 3b, 4, 4a, 4b. The conditions stated for coverage are expensive to check, and tables within [13] report check results. All eight cyclotomy constructions use Galois fields GF(q) = {0, 1, …, $q-1$} based on a prime or prime power q with $q \bmod v = 1$, and a primitive $\omega \in$ GF(q). A primitive is an element for which $\{\omega, \omega^2, \ldots, \omega^{q-1}\}=\{1, 2, \ldots, q-1\}$, i.e., for which the powers span all non-zero elements of GF(q); for example, in GF(5), 2 and 3 are primitives whereas 4 is not. According to [13], it does not matter which primitive is used in cases where there is more than one possibility. R package CAs uses the Galois field implementation in the R package lhs [14], which is based on [15]. The cyclotomic vector $\mathbf{x}_{q,v} = (x_0, \ldots, x_{q-1})$ has $x_0 = 0$, and x_i equals the logarithm of i w.r.t. base ω in GF(q), taken modulo v. This cyclotomic vector is then used for constructing a $q \times q$ matrix A, with rows and columns indexed by elements of GF(q), with $a_{ij} = x_{j-i}$, where the difference is taken in GF(q). The constructions vary in how the matrix A is post-processed: Construction 1 uses A itself, construction 2 adds $v-1$ constant rows populated with the nonzero values $1, \ldots, v-1$, constructions 3 and 4 and their variants “develop” A by vertically juxtaposing A, A+1 mod v, …, $A+v-1$ mod v, or variants where specific rows or columns are added to A before this developing step. Constructions imply different run size / column number combinations, except for a few cases which are compatible with constructions 3b and 4b, or even 3b, 4a or 4b for CA(24, 4, 12, 2); for all these, it is safe to use construction 4b, whose conditions are implied by the others. After working out the construction types, selected designs were created and were checked by repeated random sampling for obvious violations of coverage. For each construction, at least two specimens, not both with $v = 2$, were checked brute force for the stated coverage, to make sure that the constructions were correctly programmed. For resource reasons, where possible, small examples from the paper were used for these checks, as the cyclotomy-based CAs from the Colbourn tables ([1]) are often quite large. This was complicated by some mistakes especially in Table 2 of the paper, which appears to make various erroneous claims (e.g., for strength 3 with 4-level columns, that construction 3b works for $q = 41$). For a CA(1051, 4, 1051, 3), the smallest CA with 3 levels by construction 1 in the Colbourn tables, a brute force check was done by Michael Wagner using an internal powerful version of CAMETRICS ([9]); even given this powerful tool, verifying the 4-coverage of the CA took about 3.5 hours parallelized over 30 kernels. For including the cyclotomy constructions into the planned R package, it is sufficient to provide the generating information with the package and to create each CA on the fly, when requested; the run time for the largest design (strength 4 for 10009 variables at 8 levels each in 80072 runs) was about 270 seconds on one kernel on a Windows 10 machine with a 12th Gen Intel Core i7-12700T at 1.4 GHz and 32 GB RAM.

IV. Maintenance of Colbourn Tables, and Storage of Precomputed CAs and CA Construction Details

This section calls for action with the goal of securing the future of the Colbourn tables ([1]) and making them more useful by providing actual arrays and/or construction details. It is important that the community perpetuates and continues Colbourn’s tables beyond his active time as a researcher. The claim of representing the best-known CA sizes at any given point in time requires that the scientific community informs the curator(s) of relevant improvements and that the curator(s) react(s) in a timely fashion. At present, a few arrays in a hard-to-find catalogue (Torres-Jiménez [16], URL on NIST website outdated) are better than their “best-known” counterparts in the Colbourn tables. This illustrates the difficulty of keeping such tables up-to-date. It would be desirable to provide versioning for the full table, and a date for each table entry. Implementing this via a personal website is very difficult. It might be easier to achieve with a platform like GitHub. Zenodo (https://zenodo.org/) could also be an option.

Besides keeping the upper bounds for CAN(t, k, v) up-to-date, it would be extremely useful to provide actual CAs and/or CA constructions, and the rest of this section discusses this goal. For CAs that have been found by a search algorithm after substantial search time, the obvious solution is to store the CA itself. For mathematical constructions, it might suffice to provide information on construction details. UIPs are likely served better by an actual array, while other parts of the community might be served better by construction details. Stored designs and/or construction details should be easily accessible and remain available permanently. At the same time, newly-found smaller arrays should be made available in a timely fashion, ideally without making their predecessors disappear. There should be a standardized strategy regarding file names, file formats and storage locations. It is important to choose a storage strategy that supports easy access to individual CAs. This implies storing individual CAs in separate files, and using a compression that can be handled by standard tools. Like for the table of upper bounds for CAN(t, k, v), GitHub comes to mind; Zenodo is presumably less suitable for a large number of individual files. However, the author is not an expert on permanent storage strategies.

Regardless of stored CAs or stored construction parameters for a mathematical construction, it would often be very resource-intensive to verify the claimed coverage. Trustworthiness of the catalogued CAs or CA construction parameters is therefore important, and the documentation in relation to table entries should contribute to building trust. In the following, desirable documentation components for table entries are discussed, using the cyclotomy-based arrays as examples. Their implementation required working out details that are not mentioned in the Colbourn tables ([1]), and writing code for the constructions. The ground work is in the Colbourn tables, which collect the settings, for which the paper [13] and additional undocumented efforts by Colbourn and Torres-Jiménez verified the assumptions from “Constructions 4” of [13]. The Colbourn table entries could be enhanced for providing better support for array creation, by (from least to most demanding):

1) actual references for each entry, ideally by referring to identifiers from a separate list of references;

2) details needed for applying constructions from that reference, e.g., construction type and prime or prime power;

3) information on *how* the CA was obtained / verified *when* and *by whom*;

4) the actual array, with information on how it was obtained (e.g., the algorithm or the code or the software from which it was created);

5) pseudo code with a small easy to check test case (where possible).

Items 1) and 2) are meant to help experts to construct arrays. Item 3) serves to improve trust that the array fulfils the stated properties, both for experts and UIPs. Item 4) aims to support UIPs who want to work with an array without needing to understand the construction. Item 5) could help experts (and perhaps even UIPs) who want to implement the construction within a tool chain of their own choice.

Trust is now discussed in more detail for the cyclotomy constructions of Section III: All studied constructions are attributed either to Colbourn or to Torres-Jiménez. Many of those attributed to Colbourn are based on computations reported in [13] (e.g., 67, 71, 83 columns in strength 4 for 2-level columns based on Lemma 5.1 1., many CAs of strengths 4 and 5 for 3-level columns based on Lemma 5.2); ideally, the source information under items 1) and 2) should be detailed enough to go back to an individual lemma, but that might be too much to ask. The devil is in the detail, however, and there is, e.g. a small mistake in Lemma 5.1 4. of [13], where the creation of a CA(24, 4, 12, 2) is claimed for construction 4a with prime 11, while in fact only construction 4b yields strength 4. For a small prime like 11, it is easy to observe that coverage is violated (1/9 of the quadruples are not completely covered). For larger situations, a mistake like that might go unnoticed, if sample coverage checks unluckily succeed a number of times. Some of the CAs attributed to Colbourn (e.g., strength 5 CAs for more than 1231 3-level columns) and all the arrays attributed to Torres-Jiménez must have been verified by these researchers; details on when or how the verifications were conducted have not been found. While it can be assumed that both researchers are respected authorities within the community, it would nevertheless contribute to building trust if information in the spirit of item 3) were publicly documented (and linked from the tables), e.g., in terms of a report, or the code that was used.

The previous paragraph focused on the cyclotomy constructions. If one envisions enhanced Colbourn tables, it will be necessary to develop a structure that supports providing suitable information for all kinds of constructions, particularly also for CAs obtained from search algorithms. It would, in the author's view, be of great benefit to the community to provide the Colbourn tables with (links to) precomputed CAs.

The question of continuing the Colbourn tables can also be considered independently from providing collections of actual arrays. Continuing the table of known bounds will be useful even without a direct connection to actual designs, and it is also possible to provide collections of actual arrays separately from the Colbourn tables, like with the NIST tables ([4]). Regardless whether separate or together, such tools should be provided to the entire research and application community, in a stable location, with adequate documentation, and free of charge. While requirements for documentation are important, they should not be so demanding that researchers are deterred from submitting their contributions.

Acknowledgments

The author thanks Rob Carnell for implementing Galois field functionality in the R package lhs. Thanks also go to Michael Wagner for the brute force check of CA(1051, 3, 1051, 3) with the internal version of CAMETRICS, and to further members of the MATRIS research group for helpful discussions. Furthermore, thanks are due to anonymous reviewers for comments that improved the focus of this paper.

References

[1] C. J. Colbourn, "Covering array tables for t = 2, 3, 4, 5, 6". [Online]. Available: https://www.public.asu.edu/~ccolbou/src/tabby/ catable.html. Accessed on 11 Feb. 2025.

[2] J. Torres-Jiménez, I. Izquierdo-Marquez, und H. Avila-George, "Methods to construct uniform covering arrays", *IEEE Access Pract. Innov. Open Solut.*, Vol. 7, pp. 42774–42797, 2019, DOI: 10.1109/ACCESS.2019.2907057.

[3] M. Leithner, A. Bombarda, M. Wagner, A. Gargantini, und D. E. Simos, "State of the CArt: evaluating covering array generators at scale", *Int. J. Softw. Tools Technol. Transf.*, Vol. 26, No. 3, pp. 301–326, 2024, DOI: 10.1007/s10009-024-00745-2.

[4] "NIST Covering Array Tables". 2008. [Online]. Available: https://math.nist.gov/coveringarrays/.

[5] M. Wagner, K. Kleine, D. E. Simos, R. Kuhn, und R. Kacker, "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)*, Oct. 2020, pp. 191–200. DOI: 10.1109/ICSTW50294.2020.00041.

[6] JMP Statistical Discovery LLC, "*Design of experiments guide*", 2024. [Online]. Available: https://www.jmp.com/support/help/en/18.1/ index.shtml#page/jmp/covering-arrays.shtml.

[7] A. Dwyer und B. Stevens, "Covering Arrays (CA)". 2022. [Online]. Available: https://doc.sagemath.org/html/en/reference/combinat/sage/ combinat/designs/covering_array.html#sage.combinat.designs.covering_ array.covering_array.

[8] A. Gargantini und M. Radavelli, "Migrating Combinatorial Interaction Test Modeling and Generation to the Web", in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Vasteras: IEEE, Apr. 2018, pp. 308–317. DOI: 10.1109/ICSTW.2018.00066.

[9] M. Leithner, K. Kleine, und D. E. Simos, "CAMETRICS: A Tool for Advanced Combinatorial Analysis and Measurement of Test Sets", in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2018, pp. 318–327. DOI: 10.1109/ICSTW.2018.00067.

[10] D. J. Kleitman und J. Spencer, "Families of k-independent sets", *Discrete Math.*, Bd. 6, Nr. 3, pp. 255–262, Jan. 1973, DOI: 10.1016/0012-365X(73)90098-8.

[11] G. O. H. Katona, "Two applications (for search theory and truth functions) of Sperner type theorems", *Period. Math. Hung.*, Vol. 3, No. 1, pp. 19–26, March 1973, DOI: 10.1007/BF02018457.

[12] Y. Akhtar, C. J. Colbourn, und V. R. Syrotiuk, "Mixed-level covering, locating, and detecting arrays via cyclotomy", in *Combinatorics, graph theory and computing*, F. Hoffman, S. Holliday, Z. Rosen, F. Shahrokhi, und J. Wierman, Hrsg., Cham: Springer International Publishing, 2024, pp. 37–50. DOI: 10.1007/978-3-031-52969-6_4.

[13] C. J. Colbourn, "Covering arrays from cyclotomy", *Des. Codes Cryptogr.*, Vol. 55, No. 2, pp. 201–219, May 2010, DOI: 10.1007/s10623-009-9333-8.

[14] R. Carnell, "lhs: Latin Hypercube Samples". R package version 1.2.0, 2024. DOI: 10.32614/CRAN.package.lhs.

[15] A. Owen, *Orthogonal Arrays for Computer Experiments, Visualizations, and Integration in high dimensions: A C library*. (1994). [Online]. Avaiable: http://lib.stat.cmu.edu/designs/oa.c.

[16] J. Torres-Jiménez, „Covering Arrays". [Online]. Available: https://www.tamps.cinvestav.mx/~oc/. Accessed on 6 February 2025.