TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Diploma Thesis

Combinatorial and Algorithmic Constructions of Covering Arrays

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Technical Mathematics

submitted by

Ludwig Kampel

Registration Number: 0826015

submitted to the Institute of Discrete Mathematics and Geometry
of the Faculty of Mathematics and Geoinformation
of the Vienna University of Technology

Advisor: Univ. Lektor Dr. Dimitrios E. Simos

Vienna, 13.02.2018

(Signature of Author) (Signature of Advisor)

Diplomarbeit

Kombinatorische und Algorithmische Konstruktionen von Covering Arrays

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Ludwig Kampel

Matrikelnummer 0826015

eingereicht am Institut für Diskrete Mathematik und Geometrie
der Fakultät für Mathematik und Geoinformation
der Technischen Universität Wien

Betreuer: Univ. Lektor Dr. Dimitrios E. Simos

Vienna, 13.02.2018 ______________________ ______________________

(Unterschrift Verfasser) (Unterschrift Betreuer)

Contents

Abstract

Covering arrays are discrete structures appearing in combinatorial design theory. Most frequently, they are introduced as arrays having specific coverage properties regarding the appearance of tuples in certain subarrays. The aim of this thesis is not only to give a thorough introduction to covering arrays and some of their generalizations, but also to describe combinatorial and algorithmic constructions of these structures. In doing so, links to various fields of discrete mathematics such as group theory and the theory of finite fields are established. Throughout the whole thesis, the reader will be guided by an objective for *optimality*, as one notorious problem that arises is to find covering arrays that have the smallest number of rows. Often the concept of optimality has to be replaced by the aim for covering arrays that have a *small* number of rows, as the current state of the art is that constructions of optimal covering arrays are only known for some special classes of covering arrays. The generation of optimal covering arrays is not only a theoretically interesting problem, but is also of interest for practical purposes, as covering arrays find applications in testing, especially in automated software testing.

Zusammenfassung

Covering Arrays sind kombinatorische Designs. Als solche werden diese üblicherweise als Matrizen mit speziellen Eigenschaften betreffend des Vorkommens von Tupeln in gewissen Teilmatrizen definiert. Ziel dieser Arbeit ist es, eine Einführung in Covering Arrays und deren Generalisierungen zu geben, um im Anschluss sowohl kombinatorische als auch algorithmische Konstruktionsmethoden dieser Strukturen zu diskutieren. Im Verlauf dieser Diskussion werden verschiedenste Verbindungen zu anderen Teilbereichen der diskreten Mathematik, wie Gruppentheorie und endliche Körper, hergestellt und angewandt. Bei dem Studium von Covering Arrays ergibt sich das zentrale Problem, *optimale* Covering Arrays, das sind solche mit der geringsten Anzahl an Zeilen, zu erzeugen. Oft muss das Ziel, Covering Arrays mit der geringsten Anzahl an Zeilen zu finden, aufgegeben und durch ein Streben nach solchen mit einer *geringen* Anzahl an Zeilen ersetzt werden. Dies zeigt der aktuelle Stand der Forschung, nach welchem Konstruktionen für optimale Covering Arrays nur für spezielle Klassen bekannt sind. Das Generieren optimaler Covering Arrays ist nicht nur aus theoretischer Sicht ein interessantes Problem, sondern auch von praktischem Interesse, da Covering Arrays in Testverfahren, vor allem im Bereich automa-tischer Softwaretests, Anwendung finden.

Publications arisen from this Thesis

During the work conducted as part of this Master thesis the following scientific publications have arisen, which are related to the field of Combinatorial Design Theory and their Algorithms as well as their application to Combinatorial Testing.

1. Ludwig Kampel and Dimitris E. Simos, "*Set-based Algorithms for Combinatorial Test Set Generation*", in **ICTSS 2016: Proceedings of the 28th International Conference on Testing Software and Systems, Lecture Notes in Computer Science**, vol. 9976, pp. 231-240, 2016.

2. Ludwig Kampel, Bernhard Garn and Dimitris E. Simos, "*Combinatorial methods for modelling composed software systems*", in **IWCT 2017: Proceedings of the 6th International Workshop on Combinatorial Testing, collocated with ICST 2017: 10th IEEE International Conference on Software Testing, Verification and Validation**, pp. 229-238, 2017.

3. Ludwig Kampel, Bernhard Garn and Dimitris E. Simos, "*Covering arrays via set covers*", to appear in **Electronic Notes in Discrete Mathematics, vol. 65, (2018)**.

4. Ludwig Kampel, Manuel Leithner, Bernhard Garn and Dimitris E. Simos, "*Problems and Algorithms for Covering Arrays via Set Covers*", submitted for publication.

Acknowledgments

My gratitude goes to my advisor Dimitrios Simos, who awakened my interest for combinatorial design theory. I want to thank you for your guidance and the professional, technical, emotional and financial support in the last years.
Further, my thanks go to my family, friends and colleges for supporting me during the writing process of this thesis. My special thanks go to Cathi, for helping me piecing together my broken English.
For a comprehensive overview of my thanks pleas see the table below.

Thanks to\for	Support	Guidance	Friendship	Love	Money
Dimitrios	×	×			×
Family	×	×		×	×
Hubert	×	×	×	×	You still owe me 5 €
Bernhard	×	×	×	×	
Dimitris	×	×	×		
Peter, Cathi, Manuel, Kris	×		×		

Introduction

Covering Arrays (CAs) are structures appearing in combinatorial design theory and can be considered a generalization of *Orthogonal Arrays.* As such, they are introduced as matrices having specific properties regarding the appearance of tuples in subarrays. More precisely, any subarray comprised of a fixed number t of columns of a CA has the property that any t-tuple over a given alphabet appears at least once as a row of this subarray. The fixed number t is called the *strength* of the CA and is of increased interest when it comes to application domains of CAs in practice. As mentioned in [70] CAs find practical use, amongst other fields, in testing networks [95] and hardware circuits [84], further they find applications in domains as material science [100] and genomics [85]. In recent years though such matrices have attained a lot of attention due to their use in automated software testing [13, 25, 24, 34].
The author's motivation to deal with the topic of CAs is rooted in a striving to use theoretical results and abstract structures to tackle real world challenges, and thus to generate materialistic value. To the author's opinion, combinatorial designs and particularly covering arrays, play a pivotal role in that process, comparable to the application of algebra to address issues that arise with transmission channels, which lead to the development of the field of *Error-Correcting Codes* in the mid of the 20th century. To explain this viewpoint further, one can think of the increasing number of electronic devices in our daily life, an increasing number of which depends on some software product, in one way or another. As hardware and software needs to be tested to secure the quality of the products, testing techniques based on covering arrays can provide means for an efficient and cost-effective way for testing these products. At the same time the effort that needs to be spent in order to apply covering arrays for testing a certain piece of software or hardware, can be very minimal ([53]), establishing a direct link from theory to practice.
According to a report of the National Institute of Standards and Technology (NIST) [91], faults in software cost the U.S. economy up to \$59.5 billion per year, where

these costs could be reduced by \$22.2 billion, if better software testing infrastructure was available. An empirical study of the NIST from 2010 [59] shows that, in all tested software applications, faults were triggered by interactions of up to 6 input parameters. This reveals the importance of generating software tests from CAs, as each column of a CA can be identified with an *input parameter* of the software, so that each row of the array gives rise to a test. The defining property of a CA of strength t ensures that each interaction of up to t parameter values is tested, once all tests generated from the rows of one CA have been executed. This shows that the generation of CAs with a small number of rows is of major interest for applications. In fact, the problem of generating *optimal* CAs, i.e. ones with the smallest possible number of rows, is not solved for the general case and remains a challenging problem for researchers.

This thesis provides an introductory overview of the topic of CAs, highlighting combinatorial and algorithmic aspects of these structures. To this end it is structured in two parts.

The thesis is structured as follows. The first part of this master thesis starts with an introduction to CAs in Chapter 1, stating basic definitions and properties, such as the *Logarithmic Guarantee* which states that the number of rows of optimal CAs grows with the logarithm of the number of columns. In Chapter 2 some state of the art construction methods for CAs are considered. First two constructions based on algebraic structures, in this case groups and finite fields, are considered. The first construction provides means to construct orthogonal arrays over alphabets, which cardinality is a prime power, and have a limited number of columns. The second construction, based on a group acting on the entries of a matrix, is used to determine the smallest number of rows for which a certain CA exists. Further we discuss plug-in constructions, which use properties of structures appearing in combinatorial design theory together with a replacement scheme. In this context we show that the plug-in of CAs into a CA yields again a CA, giving rise to a *nested CA construction* and further describe a plug-in construction for CAs involving *perfect hash families*. Finally connections between CAs and families of sets with specific intersection properties are discussed and the equivalence of binary CAs to *independent families of sets* is established, to be reused in the second part. Many of the discussed construction methods can only be applied, if certain constraints, e.g. regarding appearing alphabet sizes, are fulfilled. In terms of application domains

these constraints can be very limiting, as many applications demand for CAs with arbitrary or mixed alphabet sizes (e.g. [54]).
In the second part of this thesis the focus is shift to algorithmic approaches for CA construction, being capable of generating CAs over arbitrary alphabets. In Chapter 3 we give a summary of algorithms for CA computation, which make use of different paradigms, including greedy, metaheuristic and hybrid methods. Thereafter, in Chapter 4, we will highlight connections between CAs and *set covers*, which enables us to interpret problems pertaining CAs as problems regarding set covers. In this context we consider some algorithms for CA generation from the view point of set covers ([47]) and provide an experimental comparison of algorithms specialized for problems pertaining CAs and algorithms for set covers. Finally in Chapter 5 we will present a family of algorithms that construct CAs via families of sets fulfilling certain intersection properties ([48, 30]).

Part I.

Theoretical Constructions

1. Preliminaries and Definitions

In this first chapter we will introduce the basic definitions and notations used throughout this thesis. This includes the definition of Covering Arrays (CAs) as combinatorial objects, as well as definitions of generalizations of CAs. Having established the necessary definitions, next we will show some basic properties of these structures.

Notations and abbreviations used in this thesis

Throughout this thesis we will use the abbreviation $[n]$ for the integer interval $\{0, 1, \ldots, n-1\}$ for $n \in \mathbb{N}$. For a set $A \subseteq U$ we denote the complement of A in U also as $A^C := A \setminus U$, omitting the specification of the underlying set U whenever there is no danger for ambiguity.

1.1. Covering Arrays

In this section we introduce Covering Arrays as arrays having specific *coverage properties*. Further we define a first generalization of these objects, loosening limitations when it comes to the underlying alphabet from which the entries of these arrays arise.

There are various ways in existing literature on how to define Covering Arrays leading to inconsistent nomenclature. Some alternative names for covering arrays are for example *surjective matrices* [40] and *surjective arrays* [14]. Yet in more recent literature the term *covering array* seems to be established for these objects, which are usually defined along the lines of [21], deviating only slightly in the used terminology. We will also follow to the definition given in [21], augmenting it afterward with some additional terminologies that will be helpful throughout this thesis.

Definition 1.1. For an integer $v \in \mathbb{N}$ we say that an array A is a *v-ary array*, if its entries arise from a set Σ, also called the *alphabet*, of cardinality $|\Sigma| = v$.

Since the properties of the arrays considered in this work do not depend on the actual alphabet, i.e. the actual elements of the set from which the entries of these arrays arise, rather on the cardinality of the alphabet, in most cases we restrict ourselves, without loss of generality, to arrays over integer intervals $[v] = \{0, 1, \dots, v-1\}$ for $v \in \mathbb{N}$.

Definition 1.2. A *covering array*, denoted as $\mathsf{CA}(N; t, k, v)$ is an $N \times k$ array with the following properties:

(i) The entries of A arise from the set $[v]$,

(ii) for each selection of t different columns, the subarray comprised by these columns has the property that every t-tuple in $[v]^t$ appears at least once as a row of the subarray.[1]

For short we denote such covering arrays by $\mathsf{CA}(N; t, k, v)$, and refer to (t, k, v) as the *parameters* of the CA.

Example 1.3. The array below is an example of a $\mathsf{CA}(6; 2, 10, 2)$.

$$\begin{array}{cccccccccc}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Remark. In general, to avoid a notation heavy thesis, when there is no immediate need to know the exact parameters of the covering arrays considered, we may use the term *covering array* or just the abbreviation *CA*. Further in case we consider CAs over the binary alphabet $\{0, 1\}$ we also refer to them as *binary CAs* for short.

Note that in some works the transposed notation of CAs is being used, where the roles of rows and columns are interchanged (e.g. in [71]).
As mentioned in the introduction, CAs can be considered a generalization of *orthogonal arrays* (OAs) of index unity, which are defined as in [10] as follows.

[1]For a set B and an integer t we denote with $B^t := \underbrace{B \times B \times \ldots \times B}_{t \text{ times}}$ the t-th cartesian power of B.

Definition 1.4. An $N \times k$ array is called an *orthogonal array* of *strength* t and *index* λ, denoted as $\mathsf{OA}_\lambda(t,k,v)$ if it has the following properties:

1. The entries arise from the set $[v]$,
2. for each selection of t different columns of the array, the subarray comprised by these columns has the property that every t-tuple in $[v]^t$ appears exactly λ times as a row of the subarray.

Remark. From the definition of orthogonal arrays it follows immediately that an $\mathsf{OA}_\lambda(t,k,v)$ needs to have exactly $N = \lambda v^t$ rows. This is also the reason why the notation of OAs is not uniform in literature, since some authors (see for example [37] denote them as $\mathsf{OA}(N;t,k,v)$ where the index λ is implicitly determined by the number of rows N.

Example 1.5. The array below is an example of an $\mathsf{OA}_2(2,5,2)$, taken from [87].

$$\begin{array}{ccccc}
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0
\end{array}$$

Remark 1.6. From the definitions of CAs and OAs it follows immediately that every $\mathsf{OA}_1(t,k,v)$ is also a $\mathsf{CA}(v^t;t,k,v)$.

From a different perspective, informally speaking, one could argue that orthogonal arrays of index unity are *perfect* covering arrays, since they attain the required property (ii) in Definition 1.2 while having the smallest number of rows possible (see Theorem 1.22). Then the following question arises naturally in the theory of CAs. Which is the smallest N for which a $\mathsf{CA}(N;t,k,v)$ exists?

Definition 1.7. For given $t,k,v \in \mathbb{N}$, with $1 \le t \le k$ and $2 \le v$, the smallest N for which a $\mathsf{CA}(N;t,k,v)$ exists is called the *covering array number* for (t,k,v) and is denoted as $CAN(t,k,v)$,

$$\mathsf{CAN}(t,k,v) := \min_{N \in \mathbb{N}} \{N \mid \exists\, CA(N;t,k,v)\}.$$

A $\mathsf{CA}(N;t,k,v)$ achieving this bound, i.e. with $N = \mathsf{CAN}(t,k,v)$ is called *optimal.*

Example 1.3 (continuing from p. 6)**.** As will be shown later in this thesis (Section 2.3) any $\mathsf{CA}(6; 2, 5, 2)$ is optimal.

The latter definition is also reflected by the following problem.

Problem 1.8 (Optimal CA (OCA))**.** Given parameters t, k and v, find a $\mathsf{CA}(N; t, k, v)$ with $N = \mathsf{CAN}(t, k, v)$.

Determining the covering array number and optimal CAs for given (t, k, v) is subject to current research. State of the art tables of upper bounds for $\mathsf{CAN}(t, k, v)$ for various (t, k, v) can be found at [18]. We will revisit to this problem in later sections, as it will always be present throughout the chapters of this thesis in one form or another. From a theoretical point of view it is a challenging problem to find CAs with a small number of rows, which can be seen at hand of the number of different approaches that are involved to create the results documented in [18]. For practical applications it is of high interest to find optimal CAs, or at least CAs with a number of rows approximating the covering array number of the respective parameters in some sense, as test suites constructed from CAs with less rows consume less resources during test execution than those constructed from CAs with more rows, see [59].

1.2. Mixed Level Covering Arrays

In this section we introduce a generalization of CAs, allowing different alphabet sizes for the different columns of an array, which leads to the notion of *mixed level covering arrays* (MCAs). Notions and concepts similar to the ones given in this section can be found in [21, 72]. For the sake of more compact writing, we introduce the following terminology.

Definition 1.9. For a family of k integers $(v_1, \ldots, v_k) \in \mathbb{N}^k$ and an $n \times k$ array A, we say that A is an array *over* $(v_1, \ldots, v_k)$, if $A = (\mathbf{c}_1, \ldots, \mathbf{c}_k) \in \mathbb{N}^{n \times k}$ and the entries in column $\mathbf{c}_j$ arise from the set $[v_j]$ for all $j \in \{1, \ldots, k\}$. In case $n = 1$, we also speak of a vector over $(v_1, \ldots, v_k)$.

The following definition deviates slightly in terminology from the one given in [21] (see also [72]).

Definition 1.10. A *mixed level covering array*[2], denoted as $\mathsf{MCA}(N; t, k, (v_1, v_2, \ldots, v_k))$ is an $N \times k$ array with the following properties:

[2]In some literature, as in [35], these arrays are called covering arrays over *heterogeneous alphabets*

(i) It is an array over $(v_1, \ldots, v_k)$.

(ii) For each selection of t different columns, the subarray comprised by these columns has the property that every t-tuple [3] in $\prod_{r=1}^{t}[v_{j_r}]$ appears at least once as a row.

We refer to $(N; t, k, (v_1, v_2, \ldots, v_k))$ as the *parameters* of the MCA, and call the parameter t the *strength* of the MCA.

In case of $v = v_1 = \ldots = v_k$ an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ is exactly a $\mathsf{CA}(N; t, k, v)$.

Remark. Despite MCAs being a generalization of CAs, we informally use the abbreviation *CA* to refer to CAs as well as MCAs, when the meaning is clear from the context.

Remark 1.11. Throughout this work we only consider CAs and MCAs for cases where $2 \leq t \leq k$ and $2 \leq v_i \; \forall i \in \{1, \ldots, k\}$, using the same notation as in Definition 1.10, to avoid dealing with trivial cases and exceptions.

Remark 1.12. In the course of this work we may use the term *coverage property* to refer to the defining property in Definition 1.2 (ii) or Definition 1.10 (ii) of CAs respectively MCAs.

The following two notations are mainly to allow a more compact writing of arguments.

Definition 1.13. For an $N \times t$ array $C = (\mathbf{c}_1, \ldots, \mathbf{c}_t)$ over $(v_1, \ldots, v_t)$ we say that C *covers* a certain t-tuple $(x_1, \ldots, x_t) \in \prod_{j=1}^{t}[v_j]$, if $(x_1, \ldots, x_t)$ appears at least once as a row of C. Further, if C covers all t-tuples of $\prod_{j=1}^{t}[v_j]$, we say that C is *covering*.

Additionally, we introduce the following exponent notation for MCA parameters: In case the alphabet sizes $v_1, \ldots, v_k$ of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ form sequences of e_i equal numbers u_i for some $i \in \{1, \ldots, r\}$, we denote the MCA also as $\mathsf{MCA}(N; t, k, (u_1^{e_1}, u_2^{e_2}, \ldots, u_r^{e_r}))$.

Example 1.14. We give below an example of an $\mathsf{MCA}(16; 3, 12, (4^2, 3^3, 2^7))$.

Analogue to the covering array number for CAs we define a similar notation for MCAs.

[3]Note that, with $\prod_{r=1}^{t}[v_{j_r}]$ we denote the Cartesian product of the sets $[v_{j_1}], \ldots, [v_{j_t}]$, i.e. all t-tuples where the r-th entry can take values from $\{0, \ldots, v_{j_r} - 1\}$.

0	0	0	1	2	1	0	0	1	1	0	1
0	1	1	2	0	0	0	1	1	0	1	0
0	2	0	1	2	1	1	1	0	0	1	0
0	3	2	0	1	0	1	0	1	1	1	1
1	0	2	2	2	0	0	1	1	1	1	0
1	1	0	1	1	0	1	0	0	1	0	0
1	2	2	0	0	0	0	0	1	1	1	1
1	3	1	2	2	1	0	1	0	0	0	1
2	0	2	0	1	1	1	1	0	0	1	0
2	1	1	0	2	1	0	0	0	1	0	0
2	2	0	2	1	0	0	0	0	0	0	1
2	3	2	1	0	1	0	0	1	1	0	1
3	0	1	2	0	1	1	0	0	1	1	0
3	1	2	2	2	0	1	0	1	1	0	1
3	2	1	1	1	1	1	1	1	1	1	1
3	3	0	0	0	0	0	0	0	0	0	0

Figure 1.1.: An $\mathsf{MCA}(16; 3, 12, (4^2, 3^3, 2^7))$.

Definition 1.15. The smallest N for which an $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$ exists is called *mixed covering array number* for $(t, k, (v_1, \dots, v_k))$ and is denoted as $\mathsf{MCAN}(t, k, (v_1, \dots, v_k))$,

$$\mathsf{MCAN}(t, k, (v_1, \dots, v_k)) := \min_{N \in \mathbb{N}} \{N \mid \exists\ \mathsf{MCA}(N; t, k, (v_1, \dots, v_k))\}.$$

An $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$ achieving this bound, i.e. with $N = \mathsf{MCAN}(t, k, (v_1, \dots, v_k))$ is called *optimal*.

The notorious problem coming along with CAs and MCAs more generally, is that of finding optimal instances, i.e. arrays having the desired coverage properties, while having the smallest number of rows possible.

Problem 1.16 (Optimal MCA (OMCA))**.** Given parameters t, k and $(v_1, \dots, v_k)$, find a $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$ with $N = \mathsf{MCAN}(t, k, (v_1, \dots, v_k))$.

Example 1.14 (continuing from p. 9)**.** As we will see from Theorem 1.22 the MCA in Figure 1.1 is an optimal MCA.

In the last decades researchers have put a vast amount of work into solving this and related problems ([20, 35, 88, 71]). This problem will accompany us throughout this whole writing in one form or another.

The striving to solve Problem 1.16, or at least to find bounds on the number of rows, for specific MCA parameters, will be present throughout the whole thesis.
In the sequel of this work we will often use the following notation, which explains the concept of *tuples* that need to be covered.

Definition 1.17. For positive integers t, k and $v_1, \dots, v_k$ with $t \leq k$, we define a *$(v_1, \dots, v_k)$-ary t-tuple* as a pair $((x_1 \dots, x_t), (p_1, \dots, p_t))$ where $x_i \in \{0, \dots, v_{p_i} - 1\}$, $\forall i \in \{1, \dots, t\}$ and $1 \leq p_1 < \dots < p_t \leq k$. For the sake of compact writing we also use the notation $\mathbf{v}$-ary t-tuple for a vector $\mathbf{v} = (v_1, \dots, v_k)$, and $(v)_{i=1}^{k}$-ary t-tuple in the case of $v_1 = v_2 = \dots, = v_k = v$.

We can visualize $(v_1, \dots, v_k)$-ary t-tuple as a vector of length k over $(v_1, \dots, v_k)$ having specified only t entries at positions p_i for $i = 1, \dots, t$ where the entries arise from the specific alphabets. We use $(v_1, \dots, v_k)$-ary t-tuples, to encode a column selection, $(p_1, \dots, p_t)$, together with a t-tuple, $(x_1, \dots, x_t)$. Hence for illustration purposes we use an informal vector notation for $(v_1, \dots, v_k)$-ary t-tuples, e.g. we denote the $(3, 2, 2)$-ary 2-tuple $((2, 1), (1, 3))$ as $(2, -, 1)$, where "$-$" represents an undefined entry. This also motivates the following definition.

Definition 1.18. For positive integers t, k and $\mathbf{v} = (v_1, \dots, v_k)$ with $t \leq k$, we say that a vector w over $(v_1, \dots, v_k)$ *covers* a $(v_1, \dots, v_k)$-ary t-tuple $((x_1 \dots, x_t), (p_1, \dots, p_t))$, if the entries of w in positions p_i equal x_i for all $i = 1, \dots, t$. We also say that an array over $(v_1, \dots, v_k)$ covers a $(v_1, \dots, v_k)$-ary t-tuple, if it is covered by one of the rows of the array.

Having established the necessary terminology we are now able to show some properties of CAs and MCAs, including some bounds for $\mathsf{CAN}(t, k, v)$ and $\mathsf{MCAN}(t, k, (v_1, \dots, v_k))$ in the following section.

1.3. Some Basic Properties of CAs and MCAs

In this section we first state some basic properties of CAs and MCAs. In doing so we follow the work of [65] and [20].
In terms of applications, binary CAs, i.e. CAs over the binary alphabet, play a special role, due to their applicability to software and hardware testing, see for example [60] and [53]. Also theoretically speaking, CAs over binary alphabets are

already interesting combinatorial objects giving rise to various research questions that are subject to current research, such as the determination of the covering array number $\mathsf{CAN}(t,k,2)$ for arbitrary t and k ([18]).

1.3.1. Basic Properties of Binary CAs

To deal with the topic of determining covering array numbers and the generation of CAs with a small number of rows, we begin with the statement of some simple or trivial properties of binary CAs, respectively covering array numbers for binary arrays along the work of [65].

Theorem 1.19. *Some simple, but also useful inequalities are:*

(i) $\mathsf{CAN}(t,k,2) \geq 2^t$,

(ii) $\mathsf{CAN}(k,k,2) = 2^k$,

(iii) $\mathsf{CAN}(t,k+1,2) \geq \mathsf{CAN}(t,k,2)$,

(iv) $\mathsf{CAN}(t+1,k+1,2) \geq 2\mathsf{CAN}(k,t,2)$.

Proof:
(i) Any $\mathsf{CA}(N;t,k,2)$ needs to have at least 2^t rows so that all binary t-tuples can appear as rows in any subarray consisting of t columns of the $\mathsf{CA}(N;t,k,2)$. Hence also $CAN(t,k,2) \geq 2^t$.
(ii) $\mathsf{CAN}(k,k,2) = 2^k$ is clear, since the array having all 2^k binary vectors of length k is a $\mathsf{CA}(k,k,2)$, together with (i), the claim is established.
(iii) Suppose we are given a $\mathsf{CA}(N;t,k+1,2)$ in column notation as $(\mathbf{c}_1,\dots,\mathbf{c}_k,\mathbf{c}_{k+1})$. Deleting one column from it, say the last, yields an $\mathsf{CA}(N;t,k,2) = (\mathbf{c}_1,\dots,\mathbf{c}_k)$, since the restriction to the first k columns does not influence the coverage property of these. Therefor every $\mathsf{CA}(N;t,k+1,2)$ yields a $\mathsf{CA}(N;t,k,2)$ and hence $CAN(t,k+1,2) \geq CAN(t,k,2)$.
(iv) Suppose we are given a $\mathsf{CA}(N;t+1,k+1,2)$ given as $A = (\mathbf{c}_1,\dots,\mathbf{c}_k,\mathbf{c}_{k+1}) = (a_{i,j})$ for $i \in \{1,\dots,N\}$ and $j \in \{1,\dots,k+1\}$. We consider the arrays A_0 and A_1 defined as follows, $A_0 := (a_{i,j})$ for $j \in \{1,\dots,k\}$ and $i \in \{h | h \in \{1,\dots,N\} \wedge a_{h,k+1} = 0\}$. In words, A_0 consists of those row vectors appearing in A that have a 0 entry in the last component, which is ignored in the definition of A_0. Analogously we define $A_1 := (a_{i,j})$ for $j \in \{1,\dots,k\}$ and $i \in \{h | h \in \{1,\dots,N\} \wedge a_{h,k+1} = 1\}$, as the matrix

comprised by the rows appearing in A that have a 1 entry in the last component, ignoring the last component at the same time for the definition of A_1. Since A is a $\mathsf{CA}(N; t+1, k+1)$ we can select any subset $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ of cardinality t, being ensured that every binary $(t+1)$-tuple appears at least once as a row when considering the subarray $(\mathbf{c}_{i_1}, \ldots, \mathbf{c}_{i_t}, \mathbf{c}_{k+1})$ of A. All $(t+1)$-tuples having a 0 entry in the last component are covered by the rows corresponding to the rows of A_0, and those with a 1 entry in the last component are covered by the rows corresponding to the rows of A_1. Hence all binary t-tuples are covered by both, the rows of A_0 as well as of the rows of A_1. At least one of the two arrays A_0 or A_1 has at most $N/2$ rows. Hence we showed that each $\mathsf{CA}(N; t+1, k+1, 2)$ yields a $\mathsf{CA}(N'; t, k, 2)$ with $N' \leq N/2$ and therefore $\mathsf{CAN}(t+1, k+1, 2) \geq 2\mathsf{CAN}(k, t, 2)$. □

We will come back to binary CAs in a later section, where we show that the problem of determining covering array numbers for binary CAs of strength two is completely solved.

1.3.2. Basic Properties of MCAs

Many results from the previous section can be generalized not only for $\mathsf{CA}(N; t, k, v)$, but also for MCAs. Before we show these generalizations, we first want to discuss some even more basic properties of MCAs, which we summarize in the following theorem.

Theorem 1.20. *Some basic properties of MCAs are:*

(i) Permuting the rows of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ *yields again an* $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$.

(ii) Permuting the columns of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ *under a permutation* $\pi \in S_k$[4]*, yields an* $\mathsf{MCA}(N; t, k, (v_{\pi(1)}, \ldots, v_{\pi(k)}))$.

(iii) Permuting the values of any column of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ *yields again an* $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$.

Proof:
(i) Is clear, since the order in which t-tuples of $\prod_{j=1}^{t} [v_{i_j}]$ appear in any subarray comprised by columns $i_1, \ldots, i_t$ of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ has no significance for the defining property (see Definition 1.10) of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$.

[4]Let S_k denote the symmetric group on a set of k elements.

(ii) Let A be an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$. Permuting the columns of an array $A=(\mathbf{c}_1,\dots,\mathbf{c}_k)$ over $(v_1,\dots,v_k)$ via some $\pi\in S_k$ yields obviously an array $A^\pi=(\mathbf{c}_{\pi(1)},\dots,\mathbf{c}_{\pi(k)})$ over $(v_{\pi(1)},\dots,v_{\pi(k)})$. Lets consider any selection $(\mathbf{c}_{\pi(i_1)},\dots,\mathbf{c}_{\pi(i_t)})$ of t different columns of A^π.
Then a t-tuple $(x_{i_1},\dots,x_{i_t})$ is covered by a row r of $(\mathbf{c}_{\pi(i_1)},\dots,\mathbf{c}_{\pi(i_t)})$, if and only if the t-tuple $(x_{\pi^{-1}(i_1)},\dots,x_{\pi^{-1}(i_t)})$ is covered by row r of the subarry $(\mathbf{c}_{i_1},\dots,\mathbf{c}_{i_t})$ of A. Since A is an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$, there exists at least one such row r.
(iii) Let again $A=(\mathbf{c}_1,\dots,\mathbf{c}_k)=(a_{i,j})$ be an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$ and $\pi_i\in S_{v_i}$ a permutation of the values $\{0,\dots,v_i-1\}$ of the i-th column of A. Let $A^{\pi_i}:=(\mathbf{c}_1,\dots,\mathbf{c}_i^{\pi_i},\dots,\mathbf{c}_k)$, where $c_i^{\pi_i}:=(\pi_i(a_{i,j}))_{j=1}^N$. Any t selection of columns of A^{π_i} not including the i-th column is covering, since A is an MCA. Consider a subarray of A^{π_i} comprised by t different columns $(\mathbf{c}_{m_1},\dots,\mathbf{c}_{m_t})$, including the i-th column, lets say $\mathbf{c}_{m_1}=\mathbf{c}_i$ without loss of generality (w.l.o.g). Then the tuple $(x_1,\dots,x_t)\in[v_i]\times\prod_{j=2}^t[v_{m_j}]$ is covered in every row r of $(\mathbf{c}_i,\dots,\mathbf{c}_{m_t})A^{\pi_i}$ where the tuple $(\pi_i^{-1}(x_1),x_2,\dots,x_t)$ is covered in A. Since A is an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$ there is at least one such r. □

Remark 1.21. Due to Theorem 1.20 (iii), we may assume at certain points, without loss of generality regarding the number of rows of the considered arrays, that we deal with MCAs where the alphabet sizes are given in a descending order, i.e. when we consider an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$ we mostly assume that $v_1\ge v_2\ge\ldots\ge v_k$.

What follows is the generalization of Theorem 1.19 for MCAs. These statements can be partly found in [35] .

Theorem 1.22. *Some basic, but also useful inequalities for MCAs are:*

(i) $\mathsf{MCAN}(t,k,(v_1,\dots,v_k))\ge\prod_{i=1}^t v_i$.

(ii) $\mathsf{MCAN}(k,k,(v_1,\dots,v_k))=\prod_{i=1}^k v_i$. *Such (M)CAs are also called* trivial (M)CAs.

(iii) Given $v_i\ge u_i\ \forall i\in\{1,\dots,k\}$*, it holds that*

$$\mathsf{MCAN}(t,k,(v_1,\dots,v_k))\ge\mathsf{MCAN}(t,k,(u_1,\dots,u_k)).$$

(iv) $\mathsf{MCAN}(t+1,k,(v_1,\dots,v_k))\ge v_i\mathsf{MCAN}(t,k,(v_1,\dots,v_{i-1},v_{i+1},\dots,v_k))$.

Proof:
(i) Since every t-tuple of $\prod_{i=1}^t[v_i]$ must appear at least once as a row in the submatrix

comprised of the first t columns of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ we have $N \geq \prod_{i=1}^{t} v_i$ and hence $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k)) \geq \prod_{i=1}^{t} v_i$.
(ii) The array comprised of all vectors over $(v_1, \ldots, v_k)$ is an $\mathsf{MCAN}(k, k, (v_1, \ldots, v_k))$. With *(i)* follows the claim.
(iii) Given an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$, say $A = (a_{i,j})$, we can define an MCA B with the same number of rows over $(u_1, \ldots, u_k)$ as follows. For each alphabet of the k columns of A let $f_i : [v_i] \to [u_i]$ be an arbitrary function with $f_i \upharpoonright_{[u_i]} = id_{[u_i]}$, and define

$$B := (b_{i,j})_{(i,j) \in \{1,\ldots,N\} \times \{1,\ldots,k\}} := (f_i(a_{i,j}))_{(i,j) \in \{1,\ldots,N\} \times \{1,\ldots,k\}},$$

by applying the function f_i to the entries in the i-th column of A for all $i \in \{1, \ldots, k\}$. Doing so collapses the alphabet $[v_i]$ of the i-th column to $[u_i]$. Consider any selection $(\mathbf{b}_{i_1}, \ldots, \mathbf{b}_{i_t})$ of columns of B and a t-tuple $(x_1, \ldots, x_t) \in \prod_{r=1}^{t} [v_{i_r}]$. Since A is an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ the t-tuple $(x_1, \ldots, x_t)$ appears at least once as a row r in $(\mathbf{a}_1, \ldots, \mathbf{a}_t)$, and since $f_i \upharpoonright_{[u_i]} = id_{[u_i]}$ also appears in row r of $(\mathbf{b}_{i_1}, \ldots, \mathbf{b}_{i_t})$.
(iv) Given an $\mathsf{MCA}(N; t+1, k, (v_1, \ldots, v_k))$, say $A = (\mathbf{a}_1, \ldots, \mathbf{a}_k)$, analogue to the construction in the proof of (iv) in Theorem 1.19 we define arrays $A_0, A_1, \ldots, A_{v_i - 1}$, where for every $s \in \{0, \ldots, v_i - 1\}$ the array A_s consist of exactly those rows of the array $(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}, \mathbf{a}_{i+1} \ldots, \mathbf{a}_k)$ where the respective row of the array $(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}, \mathbf{a}_i, \mathbf{a}_{i+1}, \ldots, \mathbf{a}_k)$ has the entry s in column i. Then there is at least one array A_s having at most N/v_i rows. Lets assume again, without loss of generality, due to Theorem 1.20 (iii), that this holds for A_0. Any selection of t indices $(m_1, \ldots, m_t)$ of columns of A_0 can be completed to a selection of $t+1$ columns of A, adding the index i to the selection $(m_1, \ldots, i, \ldots, m_t)$. Since A is an $\mathsf{MCA}(N; t+1, k, (v_1, \ldots, v_k))$ a $(t+1)$-tuple $(x_1, \ldots, 0, \ldots, x_t)$, with a 0 entry in the position corresponding to the i-th column of A, is covered at least once by these columns of A. By our construction it follows, that the t-tuple $(x_1, \ldots, x_t)$ is covered at least once by the columns of A_0. Hence A_0 is an $\mathsf{MCA}(N'; t, k, (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k))$ with $N' \leq N/v_i$, which shows

$$\mathsf{MCAN}(t, k, (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k)) \leq \frac{1}{v_i} \mathsf{MCAN}(t+1, k, (v_1, \ldots, v_k)). \quad \square$$

Lemma 1.23. *If there exists an* $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k, v_{k+1})$*, there also exists an* $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$.

Proof:
We can simply delete the last column of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k, v_{k+1})$ to obtain

an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$, as this does not influence the coverage property of the first t columns.
The statement (iii) of Theorem 1.22 shows connections between the number of rows of MCAs and the reduction of alphabet sizes. Next we show a result from [72] for the converse, increasing the size of the alphabet of the column with the largest alphabet.

Theorem 1.24. *Let $e \geq 0$ and $v_1 \geq v_2 \geq \ldots \geq v_k$, then*

$$\mathsf{MCAN}(2, k, (v_1 + e, \ldots, v_k)) \leq \mathsf{MCAN}(2, k, (v_1, \ldots, v_k)) + ev_2$$

Proof:
Given $A = (\mathbf{c}_1, \ldots, \mathbf{c}_k)$ an $\mathsf{MCA}(N; 2, k, (v_1, \ldots, v_k))$, we can consider the array consisting of the first two columns $(\mathbf{c}_1, \mathbf{c}_2)$. One way to cover all 2-tuples of $[v_1 + e] \times [v_2]$ is to append an array $(\mathbf{d}_1, \mathbf{d}_2)$ having as rows all pairs of $\{v_1, \ldots, v_1 + e - 1\} \times \{0, \ldots, v_2 - 1\}$. Vertically juxtaposing the array $(\mathbf{d}_1, \mathbf{d}_2)$ to $(\mathbf{c}_1, \mathbf{c}_2)$ we get an array

$$(\mathbf{c}_1', \mathbf{c}_2') = \begin{pmatrix} \mathbf{c}_1, \mathbf{c}_2 \\ \mathbf{d}_1, \mathbf{d}_2 \end{pmatrix},$$

consisting of two columns that are covering all 2-tuples of $[v_1 + e] \times [v_2]$. We can append the column vector $\mathbf{d}_2$ to all columns of A, yielding an array

$$(\mathbf{c}_1', \ldots, c_k') := \begin{pmatrix} \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k \\ \mathbf{d}_1, \mathbf{d}_2 \ldots, \mathbf{d}_2. \end{pmatrix},$$

where every subarray $(\mathbf{c}_1', \mathbf{c}_j')$ covers all pairs of $[v_1 + e] \times [v_j]$. After collapsing the alphabets of the columns $\mathbf{c}_j'$ for all $j \in \{2, \ldots, k\}$, via functions $f_j : [v_2] \to [v_j]$ with $f_j \restriction_{[v_j]} = id_{[v_j]}$, we attain an $\mathsf{MCA}(N + ev_2; t, k, (v_1 + e, \ldots, v_k))$. $\square$

1.4. Asymptotics for Covering Array Numbers

In this section we establish an important result, showing that the smallest number of rows for which (mixed-level) covering arrays exist is lower and upper bound by multiples of the logarithm of the number of columns of the considered arrays. The following result is also mentioned in [7].

Theorem 1.25. *For fixed v and t, we have $\mathsf{CAN}(t, k, v) \geq \log_v k$.*

Proof:
A necessary condition for an $N \times k$ array $A = (\mathbf{c}_{i_1}, \mathbf{c}_{i_2}, \ldots, \mathbf{c}_{i_t})$ to be an $\mathsf{CA}(N; t, k, v)$ is that all columns are different. If there were two identical columns, say $\mathbf{c}_{i_1} = \mathbf{c}_{i_2}$ any selection of t (recall that we assume $t \geq 2$) columns including these columns would only cover those t-tuples with identical entries in positions i_1 and i_2, which cannot be all t-tuples, as we only consider alphabets with $v \geq 2$. Since there exist only v^N different v-ary column vectors of length N, $v^N \geq k$, respectively $N \geq \log_v(k)$ is a necessary condition for the existence of a $\mathsf{CA}(N; t, k, v)$. □

Corollary 1.26. *Expressing* $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k))$ *in terms of* k*, it belongs to* $\Omega(\log k)$*, in particular:*

$$\mathsf{MCAN}(t, k, (v_1, \ldots, v_k)) \in \Omega(\log k).$$

Proof:
Assume that $(v_1, \ldots, v_k)$ is ordered descending. From Theorem 1.22 (iii) we know that $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k)) \leq \mathsf{CAN}(t, k, v_1)$ and from Theorem 1.25 we know that

$$\mathsf{CAN}(t, k, v_1) \leq \log_{v_1}(k).$$

Since $\log_{v_1} k = \log_b k / \log_b v_1$, and $\log_b(v_1)$ is a constant not depending on k, we get $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k)) \in \Omega(\log_b(k))$ for arbitrary base b of the logarithm. □
The following theorem was proven for the special case of CAs of strength $t = 2$ over a v-ary alphabet in [13], which we generalize for arbitrary strength and mixed alphabets as follows.

Theorem 1.27. *Let* $1 \leq t \leq k$, $(v_1, \ldots, v_k) =: \mathbf{v}$ *be a* k*-tuple with* $v_1 \geq v_2 \geq \ldots \geq v_k \geq 2$*,* A *be an* $s \times k$ *array over* $(v_1, \ldots, v_k)$ *and* n *be the number of* $\mathbf{v}$*-ary* t*-tuples not covered by any row of* A*. Then there exists a row* $r \in \prod_{i=1}^{k}\{0, \ldots, v_i - 1\}$ *that covers at least* n/h *of the* $\mathbf{v}$*-ary* t*-tuples not covered by the rows of* A*, where* $h := \prod_{i=1}^{t} v_i$.

Proof:
Let $R := \prod_{i=1}^{k}\{0, \ldots, v_{i-1}\}$ denote the set of all rows over $(v_1, \ldots, v_k)$. We define W to be the set of all pairs of rows over $(v_1, \ldots, v_k)$ and $\mathbf{v}$-ary t-tuples covered by them, i.e.

$$W := \{(d, p) | d \in R \text{ and } d \text{ covers the } \mathbf{v}\text{-ary } t\text{-tuple } p\}.$$

Each $d \in R$ appears exactly in $\binom{k}{t}$ elements of W as a first component. Furthermore, a $\mathbf{v}$-ary t-tuple $p = ((x_1, \dots, x_t), (p_1, \dots, p_t))$ is covered by exactly $\prod_{i \in \{1,\dots,k\} \setminus \{p_1,\dots,p_t\}} v_i$ rows of R, and hence appears as second component of exactly that many elements of W. Therefore, $\mathbf{v}$-ary t-tuples $((x_1, \dots, x_t), (p_1, \dots, p_t))$ with $(p_1, \dots, p_t) = (v_1, \dots, v_t)$ appear least often as second components of elements of W, namely exactly $\ell := \prod_{i=t+1}^{k} v_i$ times. We consider the subset $V \subseteq W$, defined as

$$V := \{(d, p) |\ p \text{ is not covered by any row of } A \text{ and } d \text{ covers } p\},$$

and prove the theorem by counting the cardinality of V in two different ways.
For any pair $(d, p) \in V$, since p is a $\mathbf{v}$-ary t-tuple not covered by the rows of A, all pairs (d', p) where d' covers p appear in V, and hence p appears at least ℓ times as second component of a pair in V. Since this holds for all of the n $\mathbf{v}$-ary t-tuples that are not covered by the rows of A, we get

$$\ell \cdot n \quad \leq \quad |V|. \tag{1.1}$$

Conversely, let m_d denote the number of $\mathbf{v}$-ary t-tuples the row d covers that are not covered by the rows of A, i.e. $0 \leq m_d \leq \binom{k}{t}$, and let $m := \max_{d \in R} m_d$ denote the maximum of the m_d's. Then we also have

$$|V| = \sum_{d \in R} m_d \leq m \cdot |R| = m \cdot \prod_{i=1}^{k} v_i. \tag{1.2}$$

From (1.1) together with (1.2) we get

$$m \geq \frac{|V|}{\prod_{i=1}^{k} v_i} \geq \frac{\ell \cdot n}{\prod_{i=1}^{k} v_i} = \frac{n}{h},$$

and hence there exists a $d \in R$ that covers at least n/h $\mathbf{v}$-ary t-tuples that are not covered by the rows of A. □

Using the same notation as in the last theorem, we obtain the following.

Corollary 1.28 (Logarithmic Guarantee)**.** *Let t, k and $(v_1, \dots, v_k)$ be given and let denote $h := \prod_{i=1}^{t} v_i$, then*

$$\mathsf{MCAN}(t, k, (v_1, \dots, v_k)) \leq h \cdot \log(h) + h \cdot \log\left(\binom{k}{t}\right) + 1. \tag{1.3}$$

Hence we have in terms of k:

$$\mathsf{MCAN}(t, k, (v_1, \dots, v_k)) \in O(\log k). \tag{1.4}$$

Proof:
We give a constructive proof, constructing an $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$ with $N \leq h \cdot \log(h) + h \cdot \log\left(\binom{k}{t}\right) + 1$. We start with an empty array, adding one row after another proceeding in steps. Form Theorem 1.27 we know that in each such step there exists a row that covers at least a fraction of $\frac{1}{h}$ of the yet uncovered $\mathbf{v}$-ary t-tuples. The initial number of uncovered $\mathbf{v}$-ary t-tuples is

$$\sum_{\{i_1,\dots,i_t\}\subseteq\{1,\dots,k\}} \prod_{j=1}^{t} v_{i_j} \leq \binom{k}{t} \prod_{i=1}^{t} v_i = \binom{k}{t} h =: n_0$$

We are therefore ensured to have found an MCA, after adding at most s rows to the initially empty array, where

$$n_0 \cdot \left(1 - \frac{1}{h}\right)^s < 1 \tag{1.5}$$

$$\Leftrightarrow \quad s \cdot \ln\left(1 - \frac{1}{h}\right) < \ln\left(\frac{1}{n_0}\right) \tag{1.6}$$

$$\Leftrightarrow \quad s > -\frac{\ln n_0}{\ln(1 - 1/h)}. \tag{1.7}$$

Using $-\ln(1 - x) > x$, for $0 < x < 1$, we get

$$-\frac{\ln n_0}{\ln(1 - 1/h)} < h \cdot \ln(n_0),$$

and hence $s > h \cdot \ln(n_0)$ also ensures

$$n_0 \cdot \left(1 - \frac{1}{h}\right)^s < 1.$$

We therefore need to add at most $h \cdot \ln(n_0) + 1 = h \cdot \ln(h) + h \cdot \ln\binom{k}{t} + 1$ rows, which is in $O(h(\log h + t \log k))$ or in $O(\log\ k)$ in terms of k. □
The results of Corollary 1.26 and Corollary 1.28 can be summarized in the following theorem.

Theorem 1.29. *In terms of the number of columns k we have for MCAs:*

$$\mathsf{MCAN}(t, k, (v_1, \dots, v_k)) \in \Theta(\log k).$$

Proof:
From Corollary 1.26 we get $\mathsf{MCAN}(t, k, (v_1, \dots, v_k)) \in \Omega(\log k)$ and from Corollary 1.28 we get $\mathsf{MCAN}(t, k, (v_1, \dots, v_k)) \in O(\log k)$, which proves the claim.

2. Combinatorial Constructions

In this chapter we detail several combinatorial constructions for CAs. The first constructions considered make use of algebraic structures, utilizing properties of finite fields and groups. After that, we focus on purely combinatorial constructions, describing two methods that bear a commonality, both using a replacement scheme. Finally we show how CAs can be represented as families of sets having certain intersection properties.

2.1. Constructions Based on Algebraic Structures

The first construction discussed in this section makes use of some properties of finite fields and provides means to construct optimal CAs with a restricted number of columns, over alphabets that have a number of elements that is a prime power ([88]). The second construction, makes use of a group acting on the entries of an array and can also be helpful to determine covering array numbers, as has been shown in [11].

2.1.1. Orthogonal Arrays over Finite Fields

In the following we will prove a theorem that deals with orthogonal arrays (recall Definition 1.4) over finite fields. We will follow the proof given in [10].
Giving a thorough introduction to finite fields goes well beyond the scope of this work, instead we would like to refer the reader to [42], for the basic notions we will use of this topic. We denote with $GF(q^n)$ the finite field with q^n elements for a prime power q.
For the proof of the main result of this subsection we need the following well known lemma.

Lemma 2.1. *Let F be an arbitrary field, and M be a* Vandermonde matrix *over K, i.e.*

$$M = \begin{pmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{pmatrix}.$$

Then the determinant of M is given by

$$det(M) = \prod_{1 \leq i < j \leq n} (x_j - x_i). \tag{2.1}$$

Proof:
We show the assertion by induction.
Induction base: $n = 1$: As $det(1) = 1$ equals the empty product, the induction base holds.
Induction hypothesis: Equation (2.1) holds for n.
Induction step: $n \to n+1$:
We can substract the $i+1$st column multiplied by x_1 from the i-th for all $i = 1, \dots, n-1$ without changing the determinants value:

$$\begin{aligned} det(M) &= det \begin{pmatrix} x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n+1}^n & x_{n+1}^{n-1} & \cdots & x_{n+1} & 1 \end{pmatrix} \\ &= det \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ x_2^n - x_1 x_2^{n-1} & x_2^{n-1} - x_1 x_2^{n-2} & \cdots & x_2 - x_1 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n+1}^n - x_1 x_{n+1}^{n-1} & x_{n+1}^{n-1} - x_1 x_{n+1}^{n-2} & \cdots & x_{n+1} - x_1 & 1 \end{pmatrix} \\ &= det \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ (x_2 - x_1) x_2^{n-1} & (x_2 - x_1) x_2^{n-2} & \cdots & (x_2 - x_1) & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ (x_n - x_1) x_{n+1}^{n-1} & (x_n - x_1) x_{n+1}^{n-2} & \cdots & (x_{n+1} - x_1) & 0 \end{pmatrix}. \end{aligned}$$

Where we get the second equality by subtracting the first row once from all others. Further we can extract the factors $(x_i - x_1)$ from the i-th row for $i = 2, \dots, n+1$

to get

$$det(M) \quad = \quad \prod_{i=2}^{n+1}(x_i - x_1) \cdot det \begin{pmatrix} x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & & \vdots & \\ x_{n+1}^{n-1} & x_{n+1}^{n-2} & \cdots & x_{n+1} & 1 \end{pmatrix}.$$

The assertion follows immediately by applying the induction hypothesis. □

Theorem 2.2. *Let $q = p^n$ be a prime power ($q \in \mathbb{P}$, a prime number) and $t < q$. Then an $\mathsf{OA}_1(q^t; t, q+1, q)$ can be constructed.*

Proof:
Let us denote the elements of $GF(q)$ as $GF(q) = \{e_0, e_1, \ldots, e_{q-1}\}$. We consider the set of all polynomials of degree smaller or equal to $t-1$ over $GF(q)$:

$$H \quad := \quad \{g(x) \in GF(q)[x] \big| g(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \ldots + a_1 x + a_0\}. \quad (2.2)$$

Since the coefficients of the polynomials in H range over $GF(q)$, there are q^t polynomials in H, for which we fix an enumeration $g_0(x), g_1(x), \ldots, g_{q^t-1}$. We now define an $q^t \times q$ array $A = (a_{i,j})$ as follows

$$(a_{i,j}) := u, \text{ where } e_u = g_i(e_j), \quad (2.3)$$

and show that it is in fact an $\mathsf{OA}_1(q^t; t, q+1, q)$. Suppose on the contrary that A is not an $\mathsf{OA}_1(q^t; t, q, q)$. Hence there exist t columns $c_1, \ldots, c_t$ in which not all q-ary t-tuples appear exactly once, in other words, there exists a q-ary t-tuple that appears at least twice as a row of A, lets say in rows i and i', and let $g_i(x) = a_{t-1}x^{t-1} + \ldots, a_1 x + a_0$ and $g_{i'} = a'_{t-1}x^{t-1} + \ldots, a'_1 x + a'_0$ be the polynomials associated to these rows. Then we have $g_i(e_{c_j}) = g_{i'}(e_{c_j})$ for all $j \in \{c_1, \ldots, c_t\}$ due to the definition of A. Defining $b_i := a_i - a'_i$ for all $i \in \{0, \ldots, t-1\}$ this yields the following system of t linear equations

$$b_{t-1}e_{c_j}^{t-1} + b_{t-2}e_{c_j}^{t-2} + \ldots + b_1 e_{c_j} + b_0 = 0, \quad \forall j \in \{1, \ldots t\}. \quad (2.4)$$

Since the two polynomials $g_i(x)$ and $g_{i'}(x)$ are different, not all of the b_i's can be zero. Hence $y_0 = b_0, y_1 = b_1, \ldots, y_{t-1} = b_{t-1}$ is a *non-trivial* solution of the following system of linear equations in the unknowns $y_0, y_1, \ldots, y_{t-1}$:

$$\begin{aligned} y_{t-1}e_{c_1}^{t-1} + y_{t-2}e_{c_1}^{t-2} + \ldots + y_1 e_{c_1} + y_0 &= 0 \\ &\vdots \\ y_{t-1}e_{c_t}^{t-1} + y_{t-2}e_{c_t}^{t-2} + \ldots + y_1 e_{c_t} + y_0 &= 0 \end{aligned}$$

This means that the determinant of the matrix

$$V = \begin{pmatrix} e_{c_1}^{t-1} & e_{c_1}^{t-2} & \cdots & e_{c_1} & 1 \\ e_{c_2}^{t-1} & e_{c_2}^{t-2} & \cdots & e_{c_2} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ e_{c_t}^{t-1} & e_{c_t}^{t-2} & \cdots & e_{c_t} & 1 \end{pmatrix}$$

of coefficients of this system of linear equations must vanish. However, from Lemma 2.1 we know that the matrix V of Vandermonde type has the property that

$$det(V) = \prod_{1 \le u < v \le t} (e_{c_v} - e_{c_u}). \tag{2.5}$$

Since $e_{c_u} \neq e_{c_v}$, $\forall u \neq v \in \{1, \ldots, t\}$, as we chose t different columns of A, corresponding to different elements of $GF(q)$, none of the factors in (2.5) is zero, and hence $det(V) \neq 0$. A contradiction to (2.4), which proves A to be an $\mathsf{OA}_1(q^t; t, q, q)$. We are able to add yet another column to A, constructing an $\mathsf{OA}_1(q^t; t, q+1, q)$, by adding the column $\mathbf{a}_{q+1} = (a_{i,q+1})$ where $a_{i,q+1} := u$, where e_u is the leading coefficient of $g_i(x)$. We show that $(A|\mathbf{a}_{q+1})$ (denoting the array A extended by the column $\mathbf{a}_{q+1}$) is the desired $\mathsf{OA}_1(q^t; t, q+1, q)$. We already showed that all subarrays comprised by any t of the first q columns of $(A|\mathbf{a}_{q+1})$ are covering. Which leaves us to show, that all subarrays comprised by the last column $\mathbf{a}_{q+1}$ together with $t-1$ other columns of $(A|\mathbf{a}_{q+1})$ are covering. Lets again assume there exist t columns $c_1, \ldots, c_{t-1}, q+1$ which are not covering. Hence there are not all q-ary t-tuples covered exactly once by the q^t rows, i.e. there exist two rows i and i' that cover the same q-ary t-tuple. Let $g_i(x) = a_{t-1}x^{t-1} + \ldots, a_1 x + a_0$ and $g_{i'} = a'_{t-1}x^{t-1} + \ldots, a'_1 x + a'_0$ be the polynomials corresponding to these rows. Hence we have again $g_i(e_{c_j}) = g_{i'}(e_{c_j})$ for all $j \in \{0, \ldots, t-1\}$ as well as $a_{t-1} = a'_{t-1}$.
We define again $b_i := a_i - a'_i$ for all $i \in \{0, \ldots, t-1\}$ this yields the following system of $t-1$ linear equations ($b_{t-1} = 0$)

$$b_{t-2}e_{c_j}^{t-2} + \ldots + b_1 e_{c_j} + b_0 = 0, \quad \forall j \in \{1, \ldots t-1\}. \tag{2.6}$$

Since the two polynomials $g_i(x)$ and $g_{i'}(x)$ are different, again not all of the b_i's can be zero. Hence $y_0 = b_0, y_1 = b_1, \ldots, y_{t-2} = b_{t-2}$ is a *non-trivial* solution of the following system of linear equations in the unknowns $y_0, y_1, \ldots, y_{t-2}$:

$$\begin{aligned} y_{t-2}e_{c_1}^{t-2} + y_{t-3}e_{c_1}^{t-3} + \ldots + y_1 e_{c_1} + y_0 &= 0 \\ &\vdots \\ y_{t-2}e_{c_{t-1}}^{t-2} + y_{t-3}e_{c_{t-1}}^{t-3} + \ldots + y_1 e_{c_{t-1}} + y_0 &= 0 \end{aligned}$$

Again considering the determinant of the matrix V of the coefficients of this system

$$V = \begin{pmatrix} e_{c_1}^{t-2} & e_{c_1}^{t-3} & \cdots & e_{c_1} & 1 \\ e_{c_2}^{t-2} & e_{c_2}^{t-3} & \cdots & e_{c_2} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ e_{c_{t-1}}^{t-2} & e_{c_{t-1}}^{t-3} & \cdots & e_{c_{t-1}} & 1 \end{pmatrix}$$

it must vanish. Yet again the matrix is of Vandermonde type and hence its determinant does not vanish, since all factors in the product below are non-zero:

$$det(V) = \prod_{1 \le u < v \le t-1} (e_{c_v} - e_{c_u}) \neq 0. \tag{2.7}$$

A contradiction to (2.6). □

Applying Lemma 1.23, we immediately get the following.

Corollary 2.3. *Using the same notation as in Theorem 2.2, for every $t \le k \le q+1$ an $\mathsf{OA}_1(q^t; t, k, q)$ can be constructed.*

The following corollary was proven in [69].

Corollary 2.4. *Let n be the smallest integer such that $v \le 2^n$ and $k \le 2^n$, then we have*

$$\mathsf{CAN}(t, k, v) \le 2^t v^t - 1.$$

Proof:
If $v = 2^n$ is a power of two then, due to Theorem 2.2, there exists an $\mathsf{OA}_1(2^{n \cdot t}; t, 2^n + 1, 2^n)$ and hence also an $\mathsf{OA}_1(2^{n \cdot t}; t, 2^n, 2^n)$, and $2^{n \cdot t} \le 2^t 2^{nt} - 1$.
For the case that v is not a power of 2, i.e. $2^{n-1} < v < 2^n$, we can take the $\mathsf{OA}_1(2^{n \cdot t}; t, 2^n, 2^n)$ and reduce the underlying alphabet from $[2^n]$ to $[v]$. As in the proof of Theorem 1.22 (iii) we can construct a $\mathsf{CA}(2^{nt}; t, 2^n, v)$. Hence $\mathsf{CAN}(t, k, v) \le 2^{nt} = 2^t 2^{(n-1)t} < 2^t v^t$. □

2.1.2. CAs via Group Actions

In this subsection we review a construction for CAs of strength three described in [11], which was used to prove $\mathsf{CAN}(3,6,3) = 33$.
For the proof of the main result of this section, we need some additional notions and lemmas. As graph theory is not part of the main interest of this thesis, we refer the reader to an introductory work (e.g. [94]) for the terminology used, and directly state the following definition, which can also be found in [94].

Definition 2.5. Let $G = (V(G), E(G))$ be a graph. A *factor* or *spanning subgraph* of G is a subgraph with the same vertex set $V(G)$. A *factorization* of G is a set of factors of G whose union is whole G and that are pairwise edge-disjoint, i.e. no two factors have an edge in common. A 1-factor is a factor that is a regular graph of degree 1, i.e. each vertex is incident to exactly one edge. A 1-factorization of G is a partition of $E(G)$ into edge-disjoint 1-factors.

It is also well known that the complete graph on an even number of vertices has an 1-factorization, see [94] Theorem 6.2. for a proof.

Theorem 2.6. *For all $n \in \mathbb{N}$ the complete graph K_{2n} has an 1-factorization.*

The construction of CAs described in the following heavily relies on a group acting on the entries of an array. To describe these constructions we further need the following notion, see also [42].

Definition 2.7. An *action* of a group G on a set S is a function

$$\alpha : \begin{cases} G \times S & \to S \\ (g, s) & \mapsto s^g, \end{cases}$$

such that $\forall g_1, g_2 \in G$ and $\forall s \in S$ it holds that

$$s^e = s \quad \text{and} \quad (s^{g_1})^{g_2} = s^{g_1 g_2}.$$

In this case we also say that *G acts on S* (via the *group action* α).

Remark 2.8. As follows directly from the definition of a group action, we get for a group G acting on a set S (using the same notation as in Definition 2.7):

(i) holding $g \in G$ fixed, the induced function $\alpha_g : S \to S : \ s \mapsto \alpha(g, s) = s^g$ is a permutation of S, as its inverse is given by α_g^{-1}.

(ii) G also acts on the set $S^{n\times k}$ of $n \times k$ arrays over S, by defining the latter action component wise:

$$\begin{cases} G \times S^{n\times k} & \to S^{n\times k} \\ (g, (s_{i,j})) & \mapsto (s^g_{i,j}) \end{cases}$$

In the light of the previous remark we define the following.

Definition 2.9. Let G be a group acting on a set S, and $M = (m_{i,j}) \in S^{n\times k}$ be an array over S. Then for all $g \in G$, we define the image of M under g as

$$M^g := (m^g_{i,j}) \in S^{n\times k}.$$

The $n \cdot |G| \times k$ matrix M^G is defined by *developing* M by G, i.e. by vertically juxtaposing the images of M under the elements $g \in G$:

$$M^G := [M^g]_{g\in G} \in S^{n\cdot|G|\times k}.$$

We continue by importing the following theorem from [42]. As the proof is straightforward and not of major importance for the content of this thesis, it is omitted.

Theorem 2.10. *Let G be a group acting on S.*

(i) The relation $\sim$ defined by

$$x \sim y \Leftrightarrow \exists g \in G : x^g = y$$

is an equivalence relation. For $x \in S$ the equivalence class $[x]_\sim$ is called orbit of x (in S under G)*.*

(ii) For any $x \in S$

$$G_x := \{g \in G | x^g = x\},$$

is a subgroup of G, called the stabilizer of x*.*

The following theorem is also proven in [42]. We use the same notation as in Definition 2.7 and Theorem 2.10.

Lemma 2.11. *Let G be a group acting on a set S, then the cardinality of the orbit $[x]_\sim$ is given by the index (the number of cosets) of the stabilizer of x in G:*

$$\big|[x]_\sim\big| = [G : G_x]$$

Proof:
Let $g, h \in G$, then

$$x^g = x^h \Leftrightarrow x^{gh^{-1}} = x \Leftrightarrow gh^{-1} \in G_x \Leftrightarrow gG_x = hG_x.$$

It follows that the map defined by $gG_x \mapsto x^g$ is a well-defined bijection of the set of cosets of G_x in G onto the orbit $[x]_\sim = \{x^g | g \in G\}$. □
Using the same notation as in the last lemma, we state the following well-known theorem (see also [49]).

Theorem 2.12 (Orbit-counting Lemma). *Let G be a group acting on a set S, S/G denote the set of orbits in S under G and for $g \in G$ let $fix_S(g) := |\{x \in S | x^g = x\}|$ denote the number of* fix-points *of g in S, then*

$$|G| \cdot |S/G| = \sum_{g \in G} fix_S(g).$$

Proof:
With Lagrange's theorem ($|G| = [G : U] \cdot |U|$ for a subgroup $U \leq G$ of G, also see [49]), we get

$$\begin{aligned}
\sum_{g \in G} fix_S(g) &= \sum_{g \in G} \sum_{s \in S, s^g = s} 1 = \sum_{s \in S} \sum_{g \in G, s^g = s} 1 \\
&= \sum_{s \in S} |[s]_\sim| \overset{2.11}{=} \sum_{s \in S} [G : G_s] \\
&= \sum_{s \in S} |G| \frac{1}{|G_s|} = |G| \sum_{[s]_\sim \in S/G} \sum_{x \in [s]_\sim} \frac{1}{|G_s|} \\
&= |G| \sum_{[s]_\sim \in S/G} 1 = |G| \cdot |S/G|,
\end{aligned}$$

where for the fifth equation we use Lagrange's theorem, which states $|G| = [G : U] \cdot |U|$ for a subgroup $U \leq G$ of G (see e.g. [49]). □
We further need the following notions that can also be found in [81].

Definition 2.13. A group G is *acting r-transitive* on a set S (via the group action $(g,s) \mapsto s^g$), if and only if for all r-tuples $(s_1,\dots,s_r),(s'_1,\dots,s'_r) \in S^r$ with pairwise distinct elements $s_1,\dots,s_r$, and pairwise distinct elements $s'_1,\dots,s'_r$, there is an element $g \in G$, such that

$$(s_1^g,\dots,s_r^g) = (s'_1,\dots,s'_r).$$

If G is acting 1-transitive on S, then we simply say G is acting *transitive* on S for short.

Definition 2.14. For a finite field $GF(q)$, we define $GF_\infty(q) := GF(q) \cup \{\infty\}$, adjoining the symbol ∞ to $GF(q)$, and

$$\mathcal{L}(q) := \big\{f : GF_\infty(q) \to GF_\infty(q) \big| f(x) = \frac{ax+b}{cx+d};\ a,b,c,d \in GF(q) \wedge ad-bc \neq 0\big\},$$

called the set of *linear fractional transformations* [1], where we define $f(x) = \frac{ax+b}{cx+d}$ as follows:

(i) for $x \in GF(q)$ and $cx+d \neq 0$ via evaluating $(ax+b)(cx+d)^{-1}$ in $GF(q)$,

(ii) for $x \in GF(q)$ and $cx+d = 0$ ($\Rightarrow ax+b \neq 0$), $f(x) := \infty$,

(iii) $f(\infty) = \frac{a\infty+b}{c\infty+d} := \frac{a}{c} := \begin{cases} ac^{-1} \in GF(q),\ c \neq 0 \\ \infty,\ c = 0 \end{cases}$.

The proof of the following lemma is technical and is hence omitted.

Lemma 2.15. *For a finite field $GF(q)$ and $\mathcal{L}(q)$ as defined above, we have*

(i) $(\mathcal{L}(q),\circ)$ is a group, where $\circ$ denotes the composition of functions.

(ii) The stabilizer $\mathcal{L}(q)_\infty$ of $\infty \in GF_\infty(q)$ in $\mathcal{L}(q)$ is the set of all linear functions $Lin(q) := \{f \in \mathcal{L}(q) | f(x) = \frac{ax+b}{cx+d} \wedge c = 0\}$, which hence forms a subgroup of $\mathcal{L}(q)$.

Notice that as $c = 0$ implies $d \neq 0$ for all $f(x) = \frac{ax+b}{cx+d} \in \mathcal{L}(x)$, we can interpret $\frac{ax+b}{cx+d} = \frac{ax+b}{d} = \frac{a}{d}x + \frac{b}{d}$, with $\frac{a}{d}, \frac{b}{d} \in GF(q)$, as a linear function $f(x) = \tilde{a}x + \tilde{b}$ over $GF(q)$, with $\tilde{a}, \tilde{b} \in GF(q)$, additionally defining $\tilde{a}\infty + \tilde{b} = \infty$ for all $a,b \in GF(q)$.

[1] In projective geometry $GF_\infty(q)$ corresponds to the projective line over $GF(q)$ and $\mathcal{L}(q)$ to the projective general linear group $PGL(2,q)$.

Theorem 2.16. *Let G be a group acting transitive on a set S via*

$$\begin{cases} G \times S & \to S \\ (g,s) & \mapsto s^g, \end{cases}$$

and $x \in S$ be a fixed element. Then G is acting r-transitive on S if and only if the stabilizer G_x of x in G is acting $(r-1)$-transitive on $S \setminus \{x\}$.

Proof:
Let G act r-transitive on S, and let $(s_1, \dots, s_{r-1}), (s'_1, \dots, s'_{r-1}) \in (S \setminus \{x\})^{r-1}$, with pairwise different elements $s_1, \dots, s_{r-1}$ respectively $s'_1, \dots, s'_{r-1}$. Then $(s_1, \dots, s_{r-1}, x)$, $(s'_1, \dots, s'_{r-1}, x) \in S^r$, with pairwise different elements $s_1, \dots, s_{r-1}, x$ respectively $s'_1, \dots, s'_{r-1}, x$. Hence there exists a $g \in G$ with

$$(s_1, \dots, s_{r-1}, x)^g = (s'_1, \dots, s'_{r-1}, x),$$

and as $x^g = x$, we even have $g \in G_x$, with $(s_1, \dots, s_{r-1})^g = (s'_1, \dots, s'_{r-1})$.
Conversely let G_x act $(r-1)$-transitive on $S \setminus \{x\}$, and let $(s_1, \dots, s_r), (s'_1, \dots, s'_r) \in S^r$ with pairwise different elements $s_1, \dots, s_r$ respectively $s'_1, \dots, s'_r$. As G acts transitive on S, there are $h_1, h_2 \in G$ with $s_r^{h_1} = x$ and $x^{h_2} = s'_r$. As G_x acts $(r-1)$-transitive on $S \setminus \{x\}$, there is a $g \in G_x$ with

$$(s_1^{h_1}, \dots, s_{r-1}^{h_1})^g = (s_1'^{h_2^{-1}}, \dots, s_{r-1}'^{h_2^{-1}}),$$

as $s_1^{h_1}, \dots, s_{r-1}^{h_1}$ and $s_1'^{h_2^{-1}}, \dots, s_{r-1}'^{h_2^{-1}}$ are pairwise different (see Remark 2.8). Then

$$\begin{aligned} (s_1, \dots, s_{r-1}, s_r)^{h_1 g h_2} &= (s_1^{h_1}, \dots, s_{r-1}^{h_1}, x)^{g h_2} \\ &= (s_1'^{h_2^{-1}}, \dots, s_{r-1}'^{h_2^{-1}}, x)^{h_2} \\ &= (s'_1, \dots, s'_{r-1}, s'_r). \end{aligned}$$

As $h_1 g h_2 \in G$ the assertion follows. □

Theorem 2.17. *For a finite field $GF(q)$ and $\mathcal{L}(q)$ respectively $Lin(q)$ defined as in Definition 2.14 and Lemma 2.15, we have*

(i) $Lin(q)$ acts 2-transitive on $GF(q)$ via α, where

$$\begin{aligned} \alpha : Lin(q) \times GF(q) &\to GF(q) \\ (f, x) &\mapsto f(x). \end{aligned}$$

(ii) $\mathcal{L}(q)$ *acts* 3*-transitive on* $GF_\infty(q)$*, via* β*, where*

$$\begin{array}{rcl}\beta : \mathcal{L}(q) \times GF_\infty(q) & \to & GF_\infty(q) \\ (f, x) & \mapsto & f(x).\end{array}$$

Proof:
(i) For given $x_1 \neq x_2$ and $y_1 \neq y_2 \in GF(q)$, the linear function $f(x) = \frac{ax+b}{cx+d}$ with $c = 0$, $d = 1$, $a = (y_1 - y_2)(x_1 - x_2)^{-1}$ and $b = y_1 - x_1(y_1 - y_2)(x_1 - x_2)^{-1}$ satisfies $(x_1, x_2)^f = (y_1, y_2)$.
(ii) Follows directly from $\mathcal{L}(q)_\infty = Lin(q)$ (Lemma 2.15) and Theorem 2.16 together with *(i)*. □

Notation. In the following proof we use the notation $C_{S,k}$ to denote the $v \times k$ array, having a constant row for each element of a given set $S = \{s_0, \ldots, s_{v-1}\}$, i.e.

$$C_{S,k} = \begin{pmatrix} s_0 & s_0 & \ldots & s_0 \\ s_1 & s_1 & \ldots & s_1 \\ \vdots & \vdots & \vdots & \vdots \\ s_{v-1} & s_{v-1} & \ldots & s_{v-1} \end{pmatrix}.$$

The main idea behind the construction used in the following theorem, is to reduce the construction of a CA, to the construction of an $n \times k$ array M over S, that covers a representative of most orbits in the respective subarrays of M, such that for an appropriate group G acting on S, the array M^G or $\begin{pmatrix} M^G \\ \hline C_{S,k} \end{pmatrix}$, denoting the vertical concatenation of M^G and $C_{S,k}$, is a CA.
The following theorem is proven in [11].

Theorem 2.18. *Let* $v \geq 3$ *be an integer and* $q \geq v - 1$ *be a prime power. Then there is a* $\mathsf{CA}((2v-1)(q^3 - q) + v; 3, 2v, v)$.

Proof:
We first show that there exists a $\mathsf{CA}(N; 3, 2v, q + 1)$, with

$$N = (2v - 1)(q^3 - q) + q + 1,$$

from which, as we will see, the desired $\mathsf{CA}((2v - 1)(q^3 - q) + v; 3, 2v, v)$ can be constructed in a simple manner.

Since q is a prime power, the group $G := \mathcal{L}(q)$ acts 3-transitive on $GF(q)_\infty$ (Theorem 2.17 (ii)). Hence there are exactly the following orbits of 3-tuples in $GF_\infty(q)^3$ under $\mathcal{L}(q)$:

1. $O_1 = \{(a, a, a) | a \in GF(q)_\infty\}$,
2. $O_2 = \{(a, a, b) | a, b \in GF(q)_\infty \wedge a \neq b\}$,
3. $O_3 = \{(a, b, a) | a, b \in GF(q)_\infty \wedge a \neq b\}$,
4. $O_4 = \{(a, a, b) | a, b \in GF(q)_\infty \wedge a \neq b\}$,
5. $O_5 = \{(a, b, c) | a, b, c \in GF(q)_\infty \wedge a \neq b \wedge b \neq c \wedge a \neq c\}$.

We show the existence of an array $M \in GF_\infty(q)^{(2v-1)\times 2v}$, such that $A := \left(\frac{M^G}{C_{GF_\infty(q),k}}\right)$ is a $\mathsf{CA}(N; 3, 2v, q+1)$. Therefor we have to ensure that for any selection of three columns of M at least one representative of each orbit O_2, O_3, O_4, O_5 is covered at least once, as all 3-tuples belonging to the orbit O_1 are covered anyway by the rows of $C_{GF_\infty(q),k}$. To this end we consider the (undirected) complete graph $K_{2v} = (E, V)$ on the vertex set $V = [2v]$. Let $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_{2v-1}$ be a 1-factorization of K_{2v}, which exists due to Theorem 2.6. In other words, $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_{2v-1}$ are $2v-1$ edge-disjoint perfect matchings of K_{2v}, each containing exactly v edges.. Let $f : E \to GF_\infty(q)$ be an arbitrary function with the property that $\forall i \in \{1, \dots, 2v-1\}\ \forall e, e' \in \mathcal{F}_i : f(e) \neq f(e')$, i.e. f is injective on the edges belonging to the same 1-factor. Such a map exists, as $|GF_\infty(q)| > v$. An example for such a function would be a function f that maps an edge $\{x_1, x_2\}$ to one of its incident vertices x_1 or x_2, as 1-factors are regular graphs of degree 1, there are no edges of a 1-factor incident to the same vertex. We define the $(2v-1) \times 2v$ matrix $M = (m_{i,j})$ where $i \in \{1, \dots, 2v-1\}$, $j \in \{0, \dots, 2v-1\}$ with entries in $GF_\infty(q)$ by

$$m_{i,j} := f(e),$$

where e is the, well defined, edge of $\mathcal{F}_i$ that is incident to j. Now let j_1, j_2, j_3 be any of the $2v$ columns of M. The edge $e = \{j_1, j_2\}$ is an edge of some factor $\mathcal{F}_i$, in which x_3 is incident to some other edge e' of $\mathcal{F}_i$. Thus $f(e) = a \neq b = f(e')$ for some $a, b \in GF_\infty(q)$ and $(m_{i,j_1}, m_{i,j_2}, m_{i,j_3}) = (a, a, b)$. Thus a representative of the orbit O_2 is covered in columns j_1, j_2, j_3. The analogue argument can be carried out for the orbits O_3 and O_4 considering the edges $\{j_1, j_3\}$ and $\{j_2, j_3\}$ respectively. To show that also a representative of orbit O_5 is covered, consider that there are $(2v-4)$ 1-factors that do not contain any of the edges $\{j_1, j_2\}, \{j_1, j_3\}$ or $\{j_2, j_3\}$. As $2v \geq 5$ there is at least one 1-factor $\mathcal{F}_i'$, in which the vertices v_1, v_2 and v_3 are incident to individual edges e, e' and e'' with different images under f and hence $(m_{i',j_1}, m_{i',j_2}, m_{i',j_3}) = (a, b, c)$ for pairwise different $a, b, c \in GF_\infty(q)$.

Summarizing we have that $A := \left(\frac{M^G}{C_{GF_\infty(q),k}}\right)$ is a $\mathsf{CA}(N; 3, 2v, q{+}1)$ over the alphabet $GF_\infty(q)$ with $N = (2v{-}1)(q^3{-}q){+}q{+}1$. To obtain a $\mathsf{CA}((2v{-}1)(q^3{-}q){+}v; 3, 2v, v)$, we reduce the alphabet size as in the proof of Theorem 1.22 (iii) by replacing the v symbols of $GF_\infty(q)$ with the v symbols of $[v]$ according to an arbitrary bijection, and replace the remaining $q+1-v$ symbols of $GF_\infty(q)$ all with the same value of $[v]$, say 0. The thus additionally generated constant zero rows in $C_{GF_\infty(q),k}$ can be omitted, as they are duplicates of the constant zero row already appearing in $C_{GF_\infty(q),k}$. □

Remark 2.19. As mentioned in [71], the construction used in the previous proof can be generalized for an arbitrary group G acting on a set S^k. To construct *small* CAs the aim is to choose a group G being of *small* cardinality $|G|$ and that has view orbits in S^k, such that, provided a specific matrix $M \in S^{n \times k}$, the development of M by G yields an array M^G that has view rows. By the Orbit-counting Lemma (Theorem 2.12) we know that this is the case exactly when the all elements of G have a small number of fix-points in S^k, as it states

$$|S^k/G| \cdot |G| = \sum_{g \in G} fix(g),$$

where $fix(g)$ denotes the number of fix points of g in S^k, and $|S^k/G|$ denotes the number of orbits of G over S^k.

To conclude this subsection, we will show how Theorem 2.18 can be used for the determination of covering array numbers. To this end we import the result $\mathsf{CAN}(2, 5, 3) = 11$ stated in [88].

Theorem 2.20. *We have* $\mathsf{CAN}(3, 6, 3) = 33$.

As $\mathsf{CAN}(2, 5, 3) = 11$ we get from Theorem 1.22 (iv) that $\mathsf{CAN}(3, 6, 3) \geq 33$. Applying Theorem 2.18 with $v = 2$ and $q = 3$ we get the existence of a $\mathsf{CA}(33; 3, 6, 3)$ and hence $\mathsf{CAN}(3, 6, 3) \leq 33$, which shows $\mathsf{CAN}(3, 6, 3) = 33$. □

2.2. Plug-In Constructions

In this section we will detail two methods for CA generation, called *plug-in* constructions, which have in common a replacement scheme well known in combinatorial design theory (see [83]). Informally speaking this term is used to refer to constructions where elements of an object get replaced by some entities, yielding a new object. By ensuring specific properties of the involved structures, the resulting object has the properties of interest. It comes as no surprise that there exist plug-in constructions for CAs, due to similar constructions used for related structures in design theory [88].

2.2.1. Nested CAs

In this subsection we show that the plug-in of CAs into another CA yields again a CA, under the right conditions. This construction hence yields a *nested CA*. Note that this construction has appeared so far in the literature under different disguises, see e.g. [23, 58]. We amplify these works by providing a detailed proof of the *coverage inheritance*, that guarantees, that the result of the discussed plug-in construction is again a CA. The structure of this subsection follows the work introduced in [46].
We start with formalizing the necessary notations, before we prove the main results that provide the coverage inheritance. Further we will give examples that visualize the discussed constructions and present the used constructions as algorithmic procedures. Finally, we will briefly discuss the relevance of the presented plug-in construction in terms of applications.

Definitions

A formal description of *the* [2] plug-in constructions that we will consider in this subsection can be given as follows.

[2]Notice that there does not necessarily exist a prototype of a plug-in construction, as *plug-in* rather describes a scheme. There are plug-in constructions where arrays are plugged in ([83]), some where rows are plugged in, as is the case in this subsection, and there are plug-in constructions where columns are plugged in, as will be the case in the next subsection. As we only consider a single plug-in construction in this subsection there is no danger of ambiguity and we will refer to it as *the* plug-in construction.

Definition 2.21. Given an array $\mathcal{M}$, and arrays S_i for $i \in \{1, \dots, k\}$, with the properties:

(i) $\mathcal{M}$ has exactly k columns, $\mathcal{M} = (M_1, \dots, M_k)$.

(ii) For every $i \in \{1, \dots, k\}$ there exists a surjective mapping ϕ_i from the set of values that can appear in column M_i of $\mathcal{M}$ onto the set of rows of S_i.

Then the result of the *plug-in construction* applied to the family of arrays $(S_i)_{i=1}^k$ and the array $\mathcal{M}$, denoted as $(S_i)_{i=1}^k \times \mathcal{M} := (S_1 \times M_1, S_2 \times M_2, \dots, S_k \times M_k)$, is defined as the array that results, when, for every $i \in \{1, \dots, k\}$, each entry of column M_i of $\mathcal{M} = (m_{r,i})$ is replaced with its image under ϕ_i, i.e. a row of S_i:

$$(S_i)_{i=1}^k \times \mathcal{M} := (\phi_i(m_{r,i})).$$

Remark 2.22. In the context of the latter definition we also refer to the family $(S_i)_{i=1}^k$ as the **seed arrays** and to $\mathcal{M}$ as the **meta array**. We may also say that we *plug-in* $(S_i)_{i=1}^k$ into $\mathcal{M}$, when we apply the plug-in construction to the family of seed arrays $(S_i)_{i=1}^k$ and the meta array $\mathcal{M}$.

To get an impression of the structure of the outcome of the plug-in construction, the reader may have a look at Figure 2.1.

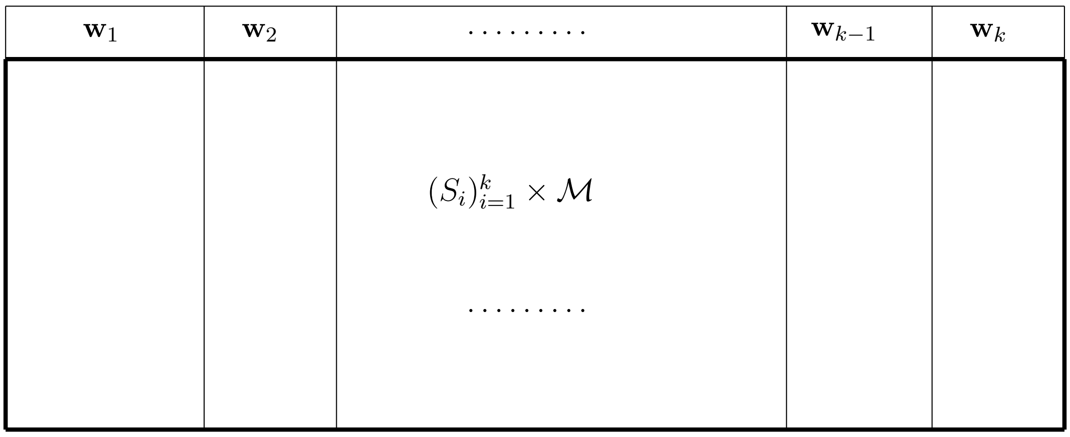


Figure 2.1.: The structure of the result $\mathcal{R} = (S_i)_{i=1}^k \times \mathcal{M}$ of the plug-in construction, applied to the seed arrays $(S_i)_{i=1}^k$ and meta array $\mathcal{M}$.

Example 2.23. Assume we are given the seed arrays A, B, C and D, as well as the meta array $\mathcal{M}$, as depicted in Figures 2.2 and 2.3.

A	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

Figure 2.2.: The arrays A, B, C and D used as seed arrays in Example 2.23.

We can map the values appearing in the columns of $\mathcal{M}$ to the rows of the seed arrays, according to the mappings given on the right hand side of Figure 2.3. Applying the plug-in construction 2.21 to the seed arrays (A, B, C, D) and the meta array $\mathcal{M}$, we attain the resulting array $\mathcal{R}$, see Figure 2.4, having the same number of rows as $\mathcal{M}$.

	$\mathcal{R}$										
#	a_1	a_2	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3
1	0	0	0	0	1	1	1	1	0	1	0
2	0	0	0	1	0	1	0	0	1	0	0
3	0	0	1	0	0	0	1	0	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
24	2	1	1	1	1	0	1	0	1	1	1

Figure 2.4.: The result $\mathcal{R}$ of the plug-in of (A, B, C, D) into $\mathcal{M}$, referring to Example 2.23.

Coverage Inheritance

Having introduced this plug-in construction for arrays, we can now prove some properties regarding the inheritance of t-way coverage when MCAs are involved in the aforementioned construction. In doing so, we follow the work in [46].

Theorem 2.24 (Coverage Inheritance: $\mathsf{CA} \times \mathsf{CA} \to \mathsf{CA}$). *Given a mixed-level covering array* $\mathcal{M} = \mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ *and a family* $S_i = \mathsf{MCA}(v_i; t_i, g_i, \mathbf{w}_i =$

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	2	2
3	0	2	3	3
4	0	3	0	0
5	1	0	2	3
6	1	1	3	0
7	1	2	0	1
8	1	3	1	2
9	2	0	3	2
10	2	1	0	3
11	2	2	1	0
12	2	3	2	1
13	3	0	0	2
14	3	1	1	3
15	3	2	2	0
16	3	3	3	1
17	4	0	0	0
18	4	1	1	1
19	4	2	2	2
20	4	3	3	3
21	5	0	0	0
22	5	1	1	1
23	5	2	2	2
24	5	3	3	3

$$\phi_1 \quad : \overset{M_1 \leftrightarrow (a_1, a_2)}{\begin{cases} 0 \mapsto & (0,0) \\ 1 \mapsto & (0,1) \\ 2 \mapsto & (1,0) \\ 3 \mapsto & (1,1) \\ 4 \mapsto & (2,0) \\ 5 \mapsto & (2,1) \end{cases}}$$

$$\phi_2 \quad : \overset{M_2 \leftrightarrow (b_1, b_2, b_3)}{\begin{cases} 0 \mapsto & (0,0,1) \\ 1 \mapsto & (0,1,0) \\ 2 \mapsto & (1,0,0) \\ 3 \mapsto & (1,1,1) \end{cases}}$$

$$\phi_3 \quad : \overset{M_3 \leftrightarrow (c_1, c_2, c_3)}{\begin{cases} 0 \mapsto & (0,0,1) \\ 1 \mapsto & (1,1,1) \\ 2 \mapsto & (1,0,0) \\ 3 \mapsto & (0,1,0) \end{cases}}$$

$$\phi_4 \quad : \overset{M_4 \leftrightarrow (d_1, d_2, d_3)}{\begin{cases} 0 \mapsto & (0,0,1) \\ 1 \mapsto & (0,1,0) \\ 2 \mapsto & (1,0,0) \\ 3 \mapsto & (1,1,1) \end{cases}}$$

Figure 2.3.: The meta array $\mathcal{M} = (M_1, M_2, M_3, M_4)$ over $(6, 4, 4, 4)$ used in Example 2.23, and functions ϕ_i that map the values occurring in M_i to rows of the arrays A, B, C and D from Figure 2.2 accordingly.

$(w_{i,1},\dots,w_{i,g_i})$) of MCAs, for all $i \in \{1,\dots,k\}$. Then an $\mathsf{MCA}(N;\tau,\sum_i g_i,(\mathbf{w}_1,\dots,\mathbf{w}_k))$ *with $\tau = \min\{t,t_1,t_2\dots,t_k\}$ and $(\mathbf{w}_1,\dots,\mathbf{w}_k)$ denoting the horizontal concatenation of the vectors $\mathbf{w}_i$, can be constructed, by applying the plug-in construction to the arrays $(S_i)_{i=1}^k$ and $\mathcal{M}$.*

Proof:
Assume $\mathcal{M},(S_i)_{i=1}^k$ and τ are given as specified above. By assumption, for each $i \in \{1,\dots,k\}$, the number of values that can appear in the i-th column of $\mathcal{M}$ is equal to the number of rows in the array S_i. Therefore, for every $i \in \{1,\dots,k\}$, there exists a bijective function ϕ_i from the set $\{1,\dots,v_i\}$ onto the set of rows of S_i. We are now in a position to apply the plug-in construction to the family of arrays $(S_i)_{i=1}^k$ and the array $\mathcal{M}$, using the bijections $\phi_i, i = 1,\dots,k$. Let $\mathcal{R}$ denote the result of this plug-in construction $(S_i)_{i=1}^k \times \mathcal{M}$ (Figure 2.1 gives a schematic of the structure of $\mathcal{R}$). We claim that $\mathcal{R}$ is an $\mathsf{MCA}(N;\tau,\sum_i g_i,(\mathbf{w}_1,\dots,\mathbf{w}_k))$. First, we show that $\mathcal{R}$ is indeed an array with the correct number of rows over the correct alphabets. Since, for all $i \in \{1,\dots,k\}$, the i-th column of $\mathcal{M}$ is expanded to exactly g_i columns via the bijection ϕ_i, it follows that $\mathcal{R}$ has $\sum_i g_i$ columns. We enumerate the columns of $\mathcal{R}$ with tuples (i,j), where $i \in \{1,\dots,k\}$ and $j \in \{1,\dots,g_i\}$. By construction it also holds that the entries in column (i,j) of $\mathcal{R}$ are elements of the set $\{1,\dots,w_{i,j}\}$, since by assumption S_i is a covering array with the parameter configuration $\mathbf{w}_i$. In the plug-in construction, entries of the meta array $\mathcal{M}$ are replaced with row vectors, therefore the number of rows does not change and $\mathcal{R}$ has N rows.
What remains to be shown, is that $\mathcal{R}$ is an MCA of strength τ. To this end, we next remark that any selection of at most τ columns that arise from one column of $\mathcal{M}$ is always covering.

Remark 2.25. Let $P = \{\mathbf{u}_1,\dots,\mathbf{u}_p\}$ be a selection of $p \leq \tau$ column indices of $\mathcal{R}$, which all arise from the expansion of a single column M_i of $\mathcal{M}$, i.e. $\mathbf{u}_s = (i,j_s)$ for a fixed i, and p different values $j_s \in \{1,\dots,g_i\}$, $\forall s = 1,\dots,p$. Then all the p-tuples in $\prod_{s=1}^p\{1,\dots,w_{i,j_s}\}$ appear within the subarray comprised of the columns (i,j_s), for $s = 1,\dots,p$, of $\mathcal{R}$. This is due to the reason, that S_i is an MCA of strength t_i, therefore also an MCA of strength p, since $p \leq \tau \leq t_i$, and due to ϕ_i being a surjective function, which guarantees that each row of S_i appears at least once in the respective sub-array of $\mathcal{R}$. This concludes our remark.

For simplicity we distinguish two cases, despite Case 1 can be viewed as a special instance of Case 2. Let $\mathbf{u}_1,\dots,\mathbf{u}_\tau$ be τ different column indices from $\mathcal{R}$, and $y \in$

$\prod_{s=1}^{\tau}\{1,\ldots,w_{\mathbf{u}_s}\}$ (note that we identify $w_{\mathbf{u}} = w_{(i,j)} = w_{i,j}$). We show that y appears as a row in the sub-array of $\mathcal{R}$ comprised of the columns $\mathbf{u}_1,\ldots,\mathbf{u}_\tau$.

Case 1: The selected columns of $\mathcal{R}$ with indices $\mathbf{u}_1,\ldots,\mathbf{u}_\tau$ arise from the expansion of exactly one column M_i of $\mathcal{M}$. Then, by Remark 2.25, all necessary τ-tuples appear in the selected sub-array of $\mathcal{R}$.

Case 2: The columns with indices $\mathbf{u}_1,\ldots,\mathbf{u}_\tau$ of $\mathcal{R}$ arise from exactly $l \geq 2$ ($l = 1$ yields Case 1) different columns $M_{i_1},\ldots,M_{i_l}$ of $\mathcal{M}$.
We partition the set of column indices $\{\mathbf{u}_1,\ldots,\mathbf{u}_\tau\}$ as follows. Let P_{i_e} be the set containing exactly those indices from $\{\mathbf{u}_1,\ldots,\mathbf{u}_\tau\}$, that arise from the expansion of column M_{i_e} of $\mathcal{M}$ via ϕ_{i_e}, i.e. $\mathbf{u}_s \in P_{i_e} \Leftrightarrow (\mathbf{u}_s = (i_e, j_s) \wedge j_s \in \{1,\ldots,g_{i_e}\})$. Then, the set $\mathcal{P} = \{P_{i_1},\ldots,P_{i_l}\}$ is a partition of the indices set $\{\mathbf{u}_1,\ldots,\mathbf{u}_\tau\}$ with l nonempty, pairwise-disjoint classes. For the given τ-tuple y, we consider its components y_e with respect to the partition $\mathcal{P}$, i.e.

$$y_e \in \prod_{j\in P_{i_e}} \{1,\ldots,w_{i_e,j}\},\ \forall e \in \{1,\ldots,l\}.$$

As $|P_{i_e}| \leq \tau - 1$, by Remark 2.25, for each class $P_{i_e} \in \mathcal{P}$, it holds that all the $|P_{i_e}|$-tuples in $\prod_{j\in P_{i_e}}\{1,\ldots,w_{i_e,j}\}$ appear within the columns of $\mathcal{R}$, specified by P_{i_e}. Furthermore, since S_{i_e} is an MCA of strength $t_{i_e} \geq \tau - 1$, for each $e \in \{1,\ldots,l\}$ we can find a row r_e of S_{i_e}, which has the $|P_{i_e}|$-tuple y_e in the positions specified by P_{i_e}. (Note that in general this row r_e is not unique.) By the bijections ϕ_{i_e}, we get elements $x_e \in \{1,\ldots,v_{i_e}\}$, with $\phi_{i_e}(x_e) = r_e, \forall e = 1,\ldots,l$. Now consider the l-tuple $(x_1,\ldots,x_l) \in \prod_{e=1}^{l}\{1,\ldots,v_{i_e}\}$. Since by assumption $\mathcal{M}$ is an MCA of strength $\tau \geq l$, there exists a row r in $\mathcal{M}$ that covers the l-tuple $(x_1,\ldots,x_l)$, and since $\phi_{i_e}(x_e)$ covers y_e, $\forall e \in \{1,\ldots,l\}$, row r in $\mathcal{R}$ covers therefore the τ-tuple y. It follows that $\mathcal{R}$ is an MCA of strength τ, which completes the proof. □

Remark 2.26. Notice that for the proof of Theorem 2.24 it would suffice that for all $i \in \{1,\ldots,k\}$, the number of values that can appear in the i-th column of $\mathcal{M}$ is *at least* the number of rows v_i of the seed array S_i and use surjections ϕ_i instead of bijections. Nevertheless, using a meta array $\mathcal{M}$ having more than v_i different values in the i-th column might yield a result $(S_i)_{i=1}^{k} \times \mathcal{M}$ having more rows than using a meta array $\mathcal{M}$ having exactly v_i different values in the i-th column, as Theorem 1.22

(iii) shows. Hence we stick to plug-in constructions $(S_i)_{i=1}^k \times \mathcal{M}$ involving arrays as described in the previous theorem.

When the CAs $\mathcal{M}$ and $(S_i)_{i=1}^k$ have the same strength, we immediately get the following result.

Corollary 2.27 (t-way Coverage Inheritance)**.** *Given a mixed-level covering array $\mathcal{M} = \mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ and a family $S_i = \mathsf{MCA}(v_i; t, g_i, \mathbf{w}_i = (w_{i,1}, \ldots w_{i,g_i}))$ of MCAs, for $i = 1, \ldots, k$. Then an $\mathsf{MCA}(N; t, \sum_i g_i, (\mathbf{w}_1, \ldots, \mathbf{w}_k))$ can be constructed, by applying the plug-in construction to the arrays $(S_i)_{i=1}^k$ and $\mathcal{M}$.*

In the light of the previous corollary we consider once more Example 2.23.

Example 2.23 (continuing from p. 34)**.** The arrays A, B, C, D and $\mathcal{M}$ are MCAs of strength $t = 2$ for their respective parameter configurations. By Theorem 2.24, the result $\mathcal{R}$ of the plug-in construction is again an MCA of strength two, namely an $\mathsf{MCA}(24, 2, 11, (3, 2^{10}))$.

Construction of Nested CAs

We can use the plug-in construction analyzed in the previous subsection for the construction of a target $\mathsf{MCA}(N; t, g, (v_1, \ldots, v_g))$, which is, due to the nature of the plug-in construction, a composition of seed arrays according to a certain pattern given by the meta array. For that reason we call such constructed arrays *nested CAs.* The construction of nested CAs can be formalized algorithmically as follows.

Algorithmic Procedure 1 (Nested CA)**.** To construct an $\mathsf{MCA}(N; t, \ell, (v_1, \ldots, v_\ell))$ proceed in the following steps:

Step 1: Partition the given parameter configuration $(v_1, \ldots, v_\ell)$ into arbitrary classes $V_1, \ldots, V_k$. [3]

Step 2: For each $i = 1, \ldots, k$, construct seed arrays $S_i = \mathsf{MCA}(N_i; t, |V_i|, V_i)$.

Step 3: Construct a meta array $\mathcal{M} = \mathsf{MCA}(N_m; t, k, (N_1, \ldots, N_k))$.

Step 4: Apply the plug-in construction to the seed arrays $(S_i)_{i=1}^k$ and the meta array $\mathcal{M}$.

[3]Recall that due to Theorem 1.20 (ii) we can permute the columns of the array, such that all parameters belonging to the same partition V_i appear sequentially in $(v_1, \ldots, v_\ell)$.

The validity of this construction, meaning that $(S_i)_{i=1}^{k} \times \mathcal{M}$ is in fact an $\mathsf{MCA}(N; t, g, (v_1, \ldots, v_g))$[4], is ensured by Theorem 2.24 (and Corollary 2.27). We exemplify this construction with the following example.

Example 2.28. Suppose we want to compute a $\mathsf{CA}(N; 3, 1000, 2)$ and that available tools are not capable of generating CAs for that many parameters, due to resource consumption or architecture. However, one way to still construct the desired CA is to follow these steps:

Step 1: Partition parameters into classes of 10 parameters each.

Step 2: Compute a seed array $S = \mathsf{CA}(12; 3, 10, 2)$.

Step 3: Compute a meta array $\mathcal{M} = \mathsf{CA}(N_m; 3, 100, 12)$.

Step 4: Apply the plug-in construction to the seed arrays $(S)_{i=1}^{100}$ and the meta array $\mathcal{M}$.

Notice that each class of the partition of the parameters consist of exactly 10 binary parameters. Hence it is sufficient to compute a seed array $S = \mathsf{CA}(12; 3, 10, 2)$ only once and use it for all classes of the partition of the parameter configuration. This way we reduce the problem of computing the whole $\mathsf{CA}(N; 3, 1000, 2)$ at once, to the computation of a seed array $S = \mathsf{CA}(12; 3, 10, 2)$ and a meta array $\mathcal{M} = \mathsf{CA}(N_m; 3, 100, 12)$, both having a fraction of the columns, the desired array has.

We will discuss potential advantages and disadvantages of the nested CA construction later in this subsection.

2.2.2. A Refinement of the Nested CA Construction

We already mentioned that the number of rows v_i of the seed arrays S_i (using the notation from Theorem 2.24) is influencing the size of the meta array, for given t and k, and therefore the size of the resulting array. The next theorem can be regarded a refined version of Theorem 2.24. It enables us to formalize a refinement of the nested CA construction, using seed arrays of strength $t - 1$ for the plug-in construction, while compensating the loss of coverage by vertically juxtaposing arrays of strength t. We will first use this construction in the proof of the following

[4]When desired the columns of the result can be permuted again.

theorem, exemplify it afterwards and finally formally present it as an algorithmic procedure. The following theorem and its proof are also from [46].

Theorem 2.29. *Given a mixed-level covering array* $\mathcal{M} = \mathsf{MCA}(N; t, k, (u_1, \dots, u_k))$ *and two families* $T_i = \mathsf{MCA}(v_i; t_i, g_i, \mathbf{w}_i = (w_{i,1}, \dots w_{i,g_i}))$ *and* $S_i = \mathsf{MCA}(u_i; t_i - 1, g_i, \mathbf{w}_i = (w_{i,1}, \dots w_{i,g_i}))$ *of MCAs, for all* $i \in \{1, \dots, k\}$*. Then an* $\mathsf{MCA}(M; \tau, \sum_i g_i, (\mathbf{w}_1, \dots, \mathbf{w}_k))$ *can be constructed, where* $M = N + \max\{v_i | i \in \{1, \dots, k\}\}$ *and* $\tau = \min\{t, t_1, t_2 \dots, t_k\}$*.*

Proof:
First we apply the plug-in construction to the arrays $(S_i)_{i=1}^k$ and $\mathcal{M}$, using arbitrary bijective functions ϕ_i from $\{1, \dots, u_i\}$ onto the set of rows of S_i for all $i \in \{1, \dots, k\}$. Let $\mathcal{H} = (S_i)_{i=1}^k \times \mathcal{M}$ denote the resulting array from the plug-in construction, and $\mathcal{R}$ denote the array when additionally vertically juxtaposing the arrays $(T_i)_{i=1}^k$ (see Figure 2.5 for the structure of $\mathcal{R}$). Since the number of rows of the $(T_i)_{i=1}^k$ can differ, we have to add some rows with *don't-care values* [5], so that all T_i have the same number of rows, to obtain a proper array. See Figure 2.5 for the structure of $\mathcal{R}$, where *don't-care values* are depicted by the symbol $*$. The justification for the claimed number of rows and for the parameter configuration of $\mathcal{R}$ is analogue to the one given in the proof of Theorem 2.24. We distinguish two cases to show that $\mathcal{R}$ is an MCA of strength τ.

Case 1: The columns with indices $\mathbf{u}_1, \dots, \mathbf{u}_\tau$ of $\mathcal{R}$ arise from a single column M_i of $\mathcal{M}$:
In this case, all required τ-tuples are covered by the rows of T_i, since T_i is an MCA of strength $t_i \geq \tau$, and it is vertically juxtaposed to the columns of $\mathcal{H}$, that arise from the expansion of column M_i of $\mathcal{M}$.

Case 2: The columns of $\mathcal{R}$ with indices $\mathbf{u}_1, \dots, \mathbf{u}_\tau$ arise from exactly $l \geq 2$ different columns $M_{i_1}, \dots, M_{i_l}$ of $\mathcal{M}$:
In this case all the required τ-tuples are covered by the rows of $\mathcal{H}$, which can be shown in analogous manner to the 2nd case of the proof of Theorem 2.24, while

[5]In literature these *don't care values* are mostly considered as *new* symbols, not belonging to the alphabet of any of the parameters, represented by $*$. We never introduced this notion formally, but we can consider these don't-care-values as arbitrary symbols of the respective alphabet represented by $*$.

noting that for the argumentation we only require that S_{i_e} is an MCA of strength $(\tau - 1)$, which holds by assumption, since $t_{i_e} - 1 \geq \tau - 1$. $\square$

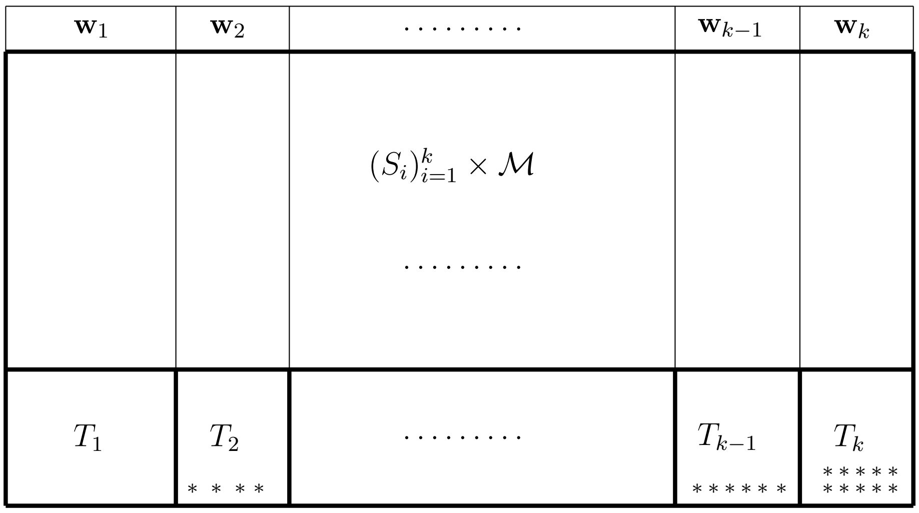


Figure 2.5.: The structure of the outcome of the refined nested CA construction, with the $(T_i)_{i=1}^k$ vertically juxtaposed to the result of the plug-in of the $(S_i)_{i=1}^k$ to $\mathcal{M}$. The array T_1 in the figure is the one having the most rows amongst the arrays $(T_i)_{i=1}^k$, this is why it needs not to be filled up with rows consisting of only *don't-care values* (denoted as $*$), like the other ones pictured.

Example 2.30. We now give an example for the construction used in the proof of Theorem 2.29. Assume we are given the seed arrays A_i, B_i, C_i and D_i for $i \in \{1, 2\}$, as well as the meta array $\mathcal{M}$, as follows:

A_1	
a_1	a_2
0	0
1	0
2	1

B_1		
b_1	b_2	b_3
0	0	0
1	1	1

C_1		
c_1	c_2	c_3
0	0	0
1	1	1

D_1		
b_1	b_2	b_3
0	0	0
1	1	1

A_2	
a_1	a_2
0	0
0	1
1	0
1	1
2	0
2	1

B_2		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

C_2		
c_1	c_2	c_3
0	0	1
1	1	1
1	0	0
0	1	0

D_2		
b_1	b_2	b_3
0	0	1
0	1	0
1	0	0
1	1	1

Figure 2.6.: A_1, B_1, C_1 and D_1 are CAs of strength 1, used as seeds for a plug-in construction with the meta array $\mathcal{M}$. A_2, B_2, C_2 and D_2 are CAs of strength 2 used for vertical juxtaposition under $\mathcal{M}$.

	$\mathcal{M}$			
#	M_1	M_2	M_3	M_4
1	0	0	1	1
2	0	1	0	0
3	1	0	0	1
4	1	1	1	0
5	2	0	0	0
6	2	1	1	1

$$M_1 \leftrightarrow A_1 \qquad \phi_1 : \begin{cases} 0 \mapsto & (0,0) \\ 1 \mapsto & (1,0) \\ 2 \mapsto & (2,1) \end{cases}$$

$$M_2 \leftrightarrow B_1 \qquad \phi_2 : \begin{cases} 0 \mapsto & (0,0,0) \\ 1 \mapsto & (1,1,1) \end{cases}$$

$$M_3 \leftrightarrow C_1 \qquad \phi_3 : \begin{cases} 0 \mapsto & (0,0,0) \\ 1 \mapsto & (1,1,1) \end{cases}$$

$$M_4 \leftrightarrow D_1 \qquad \phi_4 : \begin{cases} 0 \mapsto & (0,0,0) \\ 1 \mapsto & (1,1,1) \end{cases}$$

Figure 2.7.: Meta array $\mathcal{M}$ and maps ϕ_i, $i \in \{1,2,3,4\}$, which identify values of $\mathcal{M}$ with rows of the respective seed array, referring to Example 2.30

	$\mathcal{R}$										
#	a_1	a_2	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	1	1	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	1	1	1
4	1	0	1	1	1	1	1	1	0	0	0
5	2	1	0	0	0	0	0	0	0	0	0
6	2	1	1	1	1	1	1	1	1	1	1
7	0	0	0	0	1	0	0	1	0	0	1
8	0	1	0	1	0	1	1	1	0	1	0
9	1	0	1	0	0	1	0	0	1	0	0
10	1	1	1	1	1	0	1	0	1	1	1
11	2	0	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
12	2	1	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$

Figure 2.8.: The result $\mathcal{R}$ of the refined plug-in construction applied to A_1, B_1, C_1, D_1; A_2, B_2, C_2, D_2 and $\mathcal{M}$ as given in Figures 2.6 and 2.7.

We apply the plug-in construction (Definition 2.21) to the seed arrays (A_1, B_1, C_1, D_1) and the meta array $\mathcal{M}$, which are all MCAs of strength $t = 1$ for their respective parameter configurations. The result of this plug-in can be found in the first six rows of $\mathcal{R}$ given in Figure 2.8. $\mathcal{R}$ itself is attained by vertically juxtaposing the arrays A_2, B_2, C_2 and D_2 to the array $(A_1, B_1, C_1, D_1) \times \mathcal{M}$, and filling up empty entries with don't-care-values (denoted as $*$) if necessary. Since the arrays A_2, B_2, C_2 and D_2 are MCAs of strength $t = 2$ for their respective parameter configurations, we are ensured, by Theorem 2.29, that the resulting array $\mathcal{R}$ is an $\mathsf{MCA}(12; 2, 11, (3, 2^{10}))$, having exactly half the number of rows as the MCA in Example 2.23 attained with the plug-in construction (Algorithmic Procedure 1).

Construction of Refined Nested CAs

The construction used in the proof of Theorem 2.29 as well as in the previous example, can be formalized algorithmically as follows, yielding a refinement of the nested CA construction, referred to as *refined nested CA construction*:

Algorithmic Procedure 2 (Refined Nested CA)**.** To construct an $\mathsf{MCA}(N; t, \ell, (v_1, \ldots, v_\ell))$ proceed in the following steps:

Step 1: Partition the given parameter configuration $(v_1, \ldots, v_\ell)$ into arbitrary classes $V_1, \ldots, V_k$.

Step 2: For each $i = 1, \ldots, k$ construct seed arrays $S_i = \mathsf{MCA}(N_i; t-1, |V_i|, V_i)$.

Step 3: For each $i = 1, \ldots, k$ construct seed arrays $T_i = \mathsf{MCA}(M_i; t, |V_i|, V_i)$.

Step 4: Construct a meta array $\mathcal{M} = \mathsf{MCA}(N_m; t, k, (N_1, \ldots, N_k))$.

Step 5: Apply the plug-in construction to the seed arrays $(S_i)_{i=1}^k$ and the meta array $\mathcal{M}$.

Step 6: Vertically juxtapose the seed arrays $(T_i)_{i=1}^k$ accordingly, and fill up possible empty positions with *don't-care-values*, or arbitrary elements of the respective alphabet, to obtain a proper array.

Example 2.31. Suppose again we want to compute a $\mathsf{CA}(N; 3, 1000, 2)$ and that available tools are not capable of generating CAs for that many parameters, due to resource consumption or architecture. However, one way to still construct the desired CA may be by following these steps:

Step 1: Partition parameters into classes of 5 binary parameters each.

Step 2: Compute a seed array $S = \mathsf{CA}(6; 2, 5, 2)$.

Step 3: Compute a seed array $T = \mathsf{CA}(20; 3, 5, 2)$.

Step 4: Construct a meta array $\mathcal{M} = \mathsf{CA}(1635; 3, 200, 6))$.

Step 5: Apply the plug-in construction to the seed array $(S)_{i=1}^{200}$ and the meta array $\mathcal{M}$.

Step 6: Vertically juxtapose the seed arrays $(T)_{i=1}^{200}$ accordingly.

Again all classes of the partition are of the same size and consist of 5 binary parameters each. Therefore it is sufficient to compute the seed arrays $S = \mathsf{CA}(6; 2, 5, 2)$ and $T = \mathsf{CA}(20; 3, 5, 2)$ only once and use them for all classes of the partition of the parameter configuration. This way we reduce the problem of computing the whole $\mathsf{CA}(N; 3, 1000, 2)$ at once, to the computation of the seed arrays S, T and a meta array $\mathcal{M} = \mathsf{CA}(N_m; 3, 200, 12)$, all three having a fraction of the columns, the desired array has.

2.2.3. Relevance of Nested CA Constructions for Applications in Combinatorial Testing

As we will explain in this subsection, CAs can be interpreted as abstract test suites in terms of software testing, which can be transformed to software artifact, provided the CA matches with an *input parameter model* of the software of interest. There are cases where input parameter models demand for CAs with a specific parameter configuration and strength, that existing tools fail to compute, due to resource consumption as we will see later in this subsection. In the following we will picture how the nested CA constructions can provide means to overcome these boundaries. In this regard, our proposed algorithmic procedures provide a way to address computational challenges in the test generation process of combinatorial testing. In a second step, we will explain how far nested CA constructions fit to the modelling of composed systems.

The Tuple Counting Problem

We can benefit from the nested CA construction and its refinement, when using tools that can handle less parameters with a higher number of parameter values better than a large number of parameters, with less parameter values. To justify this claim, we present in Table 2.1 the computation time and memory usage for some instances of CAs that fail to compute using the ACTS tool (Version 2.93) [98], [76] but using the refined nested CA construction (Algorithmic Procedure 2) we are able to compute their individual building blocks (i.e. seed and meta arrays) and as a result, to construct the arrays in question.
Since the number of respective values of the parameters influences the size N, of an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ significantly, as the lower bound $\prod_{i=1}^{t} v_i$ (Theorem 1.22 (i)) for the number of rows of an MCA shows, the plug-in construction benefits from smaller seed arrays, because the parameters of the meta array have less values.
The *tuple counting problem* for CA generation depends heavily on a data structure, which keeps track of which $\mathbf{v}$-ary t-tuples are covered and which are not (see e.g. [55],

where such a data structure was subject to research). In particular, we consider as a setting for this problem the computation of a $\mathsf{CA}(N; t, k, v)$ using a CA generation tool (that is based on tuple counting).
We want to compare the direct computation of a $\mathsf{CA}(N; t, k, v)$ versus the computation of the seed arrays $S = \mathsf{CA}(N_S; t-1, k_s, v)$, $T = \mathsf{CA}(N_T; t, k_s, v)$ and a meta array $\mathcal{M} = \mathsf{CA}(N_m; t, k_m, N_{s_1})$, where $k_s \cdot k_m = k$, as used in the refined nested CA construction. Our focus is to determine how many tuples need to be counted during each computation and also to give bounds on the size of resulting arrays.
The main advantage of this algorithmic construction lies in the fact that they can be used for CA computation with considerable less memory consumption. In particular, comparing the number of tuples that need to be covered by the involved arrays, we can get smaller numbers when using the refined nested CA construction with a suitable partitioning of the parameters, compared to the direct computation of a CA. This can be crucial for CA generation tools that rely on tuple counting. We justify our claim with the following example, and generalize it afterwards:
Direct computation using CA generation tools. Suppose we want to construct a $\mathsf{CA}(N; 3, 1000, 2)$, using a CA generation tool, which is based on tuple counting, such as the framework described in [8] or the ACTS tool [98]. For the computation of a $\mathsf{CA}(N; 3, 1000, 2)$ the respective tool has to check the appearance of all possible $\binom{1000}{3} \cdot 2^3 = 1.329336 \cdot 10^9$ 3-tuples.
Refined nested CA construction using CA generation tools. If we use the refined nested CA construction, partitioning the parameters in groups of five, we first have to compute the seed arrays $S = \mathsf{CA}(N_S; 2, 5, 2)$ and $T = \mathsf{CA}(N_T; 3, 5, 2)$. CA generation tools, relying on tuple counting, have to check for the appearance of only $\binom{5}{2} \cdot 2^2 = 40$ tuples for computing S respectively $\binom{5}{3} \cdot 2^3 = 80$ tuples for computing T, which is negligible. Suppose further we get a seed array S with $N_s = 6$ rows (as is the case using IPOG as implemented in ACTS version 2.93). For the computation of the meta array $\mathsf{CA}(N_m; 3, 200, 6)$ the previous tool has to check for $\binom{200}{6} \cdot 6^3 = 2.836944 \cdot 10^8$ 3-tuples (notice that we plug-in S into $\mathcal{M}$), which are $1.0456416 \cdot 10^9$ less tuples, or roughly 1/5-th of the tuples, compared to the case of direct computation of the CA in question.

Advantages. In the general case, we know that the number of tuples to cover in the seed arrays will always be less than when constructing a CA directly, since the parameter configurations of the seed arrays are always part of the parameter configuration of the target CA. To enable extending the boundaries of the usage of a CA generation tool based on tuple counting, we only have to ensure that the number of tuples that need to be covered when constructing the meta array, is less than when directly constructing the CA:

$$\binom{k_m}{t} N_s^t < \binom{k}{t} v^t.$$

In Table 2.1 two cases are documented, where CA generation is feasible due to the refined nested CA construction. All CAs for these experiments were computed with the CA generation tool ACTS [99], [76] (in this case ACTS, version 2.93), which failed to compute instances of a $\mathsf{CA}(N; 3, 1000, 2)$ and $\mathsf{CA}(N; 3, 2000, 3)$ directly. In the first column, the entries $\mathsf{CA}(N; t, k, v)$ point to which array we are interested to construct. S, T, and $\mathcal{M}$ are the seed respectively meta arrays used for the refined Nested CA construction, and $\mathcal{R}$ the resulting array. For the cases where the "Computation Time" reads "o.o.M.", the computation aborted with an out-of-memory error. In the fourth column we give the available memory for the computations. The table entries with the resulting arrays have no computation time, as the time needed to apply the refined nested CA construction, as a process of substituting the rows of the seed arrays into the meta array, is negligible and was not measured.

Disadvantages. However, the nested CA constructions, besides the previous advantages, have also certain limitations. For example, one disadvantage of using the refined nested CA construction is that it generally constructs larger CAs compared to a direct computation. A natural lower bound on the number of rows in a $\mathsf{CA}(N; t, k, v)$ is given by v^t. In the refined nested CA construction howerver, the seed array S of strength $t-1$ has $v^{t-1} \leq N_{s_1}$ number of rows. Since the number of rows in the seed array equals the cardinality of parameter values of the meta array, we obtain a lower bound of v^{t^2-t} for the number of rows appearing in the meta array. Hence, the number of rows of the resulting array of the refined nested CA construction is bounded at least by the same value v^{t^2-t}, due to the additional vertical juxtaposition of arrays. If we choose to directly construct a $\mathsf{CA}(N; t, k_s \cdot k_m, v)$ array, we have the lower bound v^t (which is sharp for orthogonal arrays), and more generally $N \in O(v^t t(\log v + \log(k_s k_m)))$ (provided of course an algorithm capable of returning CAs of that size, see e.g. [13] or Algorithm 6 and Corollary 1.28), yielding much smaller arrays, especially for large t.

Array	Computation Time	Size	Memory
$\mathsf{CA}(N;3,1000,2)$	o.o.M.	-	5 GB
$S=\mathsf{CA}(6;2,5,2)$	0 sec	6	5 GB
$T=\mathsf{CA}(20;3,5,2)$	0 sec	20	5 GB
$\mathcal{M}=\mathsf{CA}(1635;3,200,6)$	∼10 min	1635	5 GB
$\mathcal{R}=\mathsf{CA}(1655;3,1000,6)$	-	1655	5 GB
$\mathsf{CA}(N;3,2000,3)$	o.o.M. (after >7 hrs)	-	10 GB
$S=\mathsf{CA}(6;2,5,2)$	0 sec	6	10 GB
$T=\mathsf{CA}(20;3,5,2)$	0 sec	20	10 GB
$\mathcal{M}=\mathsf{CA}(1930;3,400,6)$	∼2.5 hrs	1930	10 GB
$\mathcal{R}=\mathsf{CA}(1950;3,2000,2)$	-	1950	10 GB

Table 2.1.: Two CA generations enabled by using the ACTS tool in combination with the refined plug-in construction (Algorithmic Procedure 2).

Application of Nested CAs in Testing

Before we highlight the applicability of the previously discussed plug-in construction, in the field of combinatorial testing, we give a short introduction to combinatorial testing in the first place, and introduce some notations.

Combinatorial Testing. To apply combinatorial testing to a system under test (SUT), i.e. testing the system with a test suite based on combinatorial designs, e.g. based on CAs ([59, 60, 86]), *orthogonal latin squares* ([68]) or others, it is required to model the input space of the SUT by determining parameters (also called *factors* or *stages*), and their respective values such that, an input to this model of the SUT can be represented by a specific assignment of values to these parameters. This modelling technique is referred to as *input parameter modelling*, and the resulting model is the *input parameter model* (IPM) of the SUT [33]. Focusing on combinatorial testing based on CAs, in general the parameters of the IPM are identified with the columns of a CA. The symbols that appear in a column of the CA are mapped to the values, the corresponding parameter can take. Each row of the CA gives rise to a test, when assigning values to the input parameters of the SUT according to the entries in the row. The strength t of the CA underlying the test suite, then translates to the *t-way interaction coverage* of the parameters modelling the SUT, guaranteeing, that any t-way combination of parameter-value combinations, are executed at least once, when the whole test suite is executed. This property is of major importance, considering that a study of the National Institute of Standards and Technology (NIST) from

2010 [59], revealed that a significant amount of software faults are induced by the interaction of two or more parameters. At the same time the empirical data in [59] shows that in fielded software products these faults rely on the interactions of at most 6 parameters. For this reason CAs have attracted a lot of research attention, as their properties predestine them for applications in automated software testing [59, 86, 60].
In this context we motivate the applicability of the nested CA constructions for composed software systems, despite they can also be applicable in domains.

Nested CAs in Software Testing. Modern software design relies heavily on modular software architecture, as well structured software is easier understandable for software users and developers. Additionally, it makes further development and maintenance of the software more manageable. It is often easier to understand certain components of a software and understanding the interplay of these in a second step, in comparison to understanding the whole system at once.
Consider an SUT, being composed of several components $C_1, \ldots, C_k$, which are interacting with each other and where each component can have its own IPM. In this sense, an IPM of the composed SUT is comprised of the many (possibly different) IPMs of its components. This hierarchy of IPMs, gives a way to combine tests for the components to obtain a test suite for the SUT, that has a different IPM. Figure 2.9 shows the schematics of an SUT that is composed of four components, each having its own internal structure, i.e. each component has its own IPM. In software testing one distinguishes between testing the components of a composed SUT independently, which is referred to as unit testing ([43]), and integration testing ([77]), where the whole SUT, i.e. the interplay between the components, is tested. In the following we will briefly highlight how the nested CA constructions can be applied for testing such software systems, and hence adopt to this hierarchical structure of modern software design and interconnected systems.
Various approaches are devoted to model the operational environment of an SUT and its input space, focusing on appearing internal structures or dependencies of the acting entities, see e.g. [23, 58]. The nested CA construction is a purely combinatorial way to bridge the gap between unit testing and integration testing when applying combinatorial testing using CAs. We further explain this as follows: Assume the SUT depicted in Figure 2.9 comprised of the four components C_1, C_2, C_3, C_4, where C_1 can be modelled with one ternary and one binary input parameter and C_2, C_3

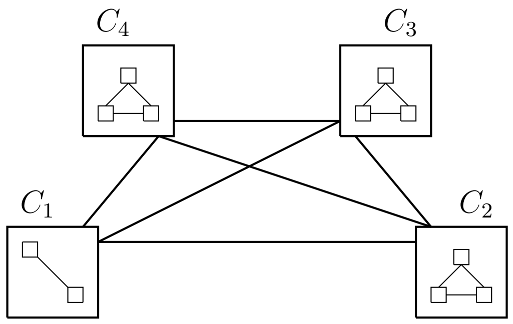


Figure 2.9.: Composed SUT, with components C_1, C_2, C_3, C_4, each having its own internal structure.

and C_4 can be modelled having three binary input parameters each. An example for such IPMs would be as follows:

IPM(C_1) :	IPM(C_2) :	IPM(C_3) :	IPM(C_4) :
p_1: on, standby, off	p_3: on, off	p_6: on, off	p_9: on, off
p_2: on, off	p_4: on, off	p_7: on, off	p_{10}: on, off
	p_5: on, off	p_8: on, off	p_{11}: on, off

Given the IPM of an SUT, we can generate a CA over the appropriate alphabet, which then in terms of testing can be considered as an abstract test suite. For example regarding C_2 we can construct a $\mathsf{CA}(N; 2, 3, 2)$ and consider it as an abstract test suite (see Figure 2.10). An IPM for the SUT comprised of the components $C_1, C_2.C_3$ and C_4 would consist of a single ternary parameter and ten binary parameters.

$\mathsf{CA}(4; 2, 3, 2)$			Test suite		
0	0	0	off	off	off
0	1	1	off	on	on
1	0	1	on	off	on
1	1	0	on	on	off

Figure 2.10.: On the left hand side a $\mathsf{CA}(4; 2, 3, 2)$ that can be interpreted as an abstract test suite for C_2 provided the IPM(C_2). On the left hand side a test suite for C_2, generated from the CA on the right, by is instantiating the values according to IPM(C_2).

Consider the case that there already exist CAs, i.e. abstract test suites, for the components, e.g. the covering arrays A, B, C respectively D in Example 2.23. Given these CAs of strength 2, we can construct an abstract test suite for the whole SUT, by nesting the CAs that give rise to the test suites of the components, merging them using the nested CA construction (Algorithmic Procedure 1), using an appropriate meta array $\mathcal{M} = (M_1, M_2, M_3, M_4)$, also of strength 2 (see again Example 2.23). The resulting CA is compatible with the IPM for the whole SUT, consisting of one ternary and ten binary parameters. Then a test suite for the whole SUT, generated from this array, enjoys full 2-way interaction coverage, due to Theorem 2.27. Put differently, we just described how to generate an abstract combinatorial integration test suite for an SUT, starting from abstract combinatorial unit test suites for the SUTs components.
An application of this methodology to modelling the system call interface of the Linux kernel can be found in [46].

2.2.4. CAs from Perfect Hashfamilies

In this subsection we describe another plug-in construction for CAs which was introduced in [69]. Different to the *row-wise* nature of the plug-in construction discussed in the previous subsection, in the sequel of this subsection we will consider a *column-wise* plug-in of arrays. Also the target structure in which the columns get plugged-in are no longer CAs, but perfect hash families (PHFs).

Perfect Hash Families

Following the work of [69] we first give a definition, before we discuss connections of PHFs to other classes of mathematical objects, including CAs.

Definition 2.32. A *t-perfect hash family* is an $N \times k$ array over the alphabet $[q]$, denoted as $\mathsf{PHF}(N; k, q, t)$, with the property that for a fixed integer t all subarrays comprised of any t columns, have the property that there is at least one row that has pairwise different entries in these columns.

The typical problem for PHFs is to construct a $\mathsf{PHF}(N; k, q, t)$, for given k, q and t, with a small, or the smallest possible, number of rows N. Therefore the interesting cases are those where $k \geq q \geq t$, as for $k < q$ there always exists a PHF consisting of only one row having pairwise different entries, and as for $q < t$ there does not exist a q-ary vector of length t having pairwise different entries and hence no $\mathsf{PHF}(N; k, q, t)$.

Remark. We also immediately see that any $\mathsf{CA}(N; t, k, v)$ with $t \leq v$ is also a $\mathsf{PHF}(N; k, v, t)$, as $t \leq v$ guarantees the appearance of a t-tuple having pairwise different entries in every selection of t columns.

Example 2.33. The following array gives an example of a $\mathsf{PHF}(6; 12, 3, 3)$:

$$\begin{bmatrix} 0 & 1 & 2 & 2 & 1 & 2 & 2 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 2 & 2 & 2 & 1 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 0 & 2 \\ 2 & 0 & 1 & 1 & 2 & 0 & 2 & 0 & 1 & 1 & 2 & 1 \\ 2 & 0 & 2 & 1 & 2 & 1 & 0 & 2 & 2 & 1 & 1 & 0 \\ 2 & 0 & 1 & 2 & 1 & 1 & 2 & 2 & 0 & 1 & 2 & 1 \end{bmatrix}.$$

Interpreting the rows of a $\mathsf{PHF}(N; k, q, t)$ as functions $f_i : [k] \to [q]$ for all $i \in \{1, \ldots, N\}$, the defining property of PHFs means that for each subset of $[k]$ having cardinality t, there is at least one function f_i that is injective on this subset.
There is a close connection between PHFs and error-correcting codes, as the following theorem, which was proven in [1], shows. Before we show the theorem, we introduce the following definition for the sake of completeness. See also [41].

Definition 2.34. An (N, k, d, q) *code* is a subset $C \subseteq A^N$ of vectors, ehich are also called *codewords*, with $|A| = q$ and $|C| = k$, with the property that the *Hamming distance* $d_H(u, v) := |\{i | i \in \{1, \ldots, N\} \wedge u_i \neq v_i\}|$ for any two distinct codewords $u = (u_1, \ldots, u_N)$ and $v = (v_1, \ldots, v_N)$ in C is at least d.

Theorem 2.35. *If there exists an (N, k, d, q) code C with $N > (N - d)\binom{t}{2}$ for an integer t, then there exists a $\mathsf{PHF}(N; k, q, t)$.*

Proof:
We show that the $N \times k$ array $(\mathbf{u}_1, \ldots, \mathbf{u}_k)$ comprised by the column-wise vectors of an (N, k, d, q) code C, with $N > (N - d)\binom{t}{2}$, is the desired $\mathsf{PHF}(N; k, q, t)$. Let $\{c_1, \ldots, c_t\} \subseteq C$ be a set of t different arbitrary vectors of C. We show that there is a coordinate where these codewords are pairwise different, which concludes the proof.
Suppose there exists no such coordinate. Then for each coordinate $i \in \{1, \ldots, N\}$ there exist at least two vectors, having the same entry in the coordinate i. If we

consider the sum of all Hamming distances of pairs of vectors from $\{c_1, \dots, c_t\}$ we hence get

$$\begin{aligned}\binom{t}{2} d \leq \sum_{1 \leq i < j \leq N} d_H(c_i, c_j) &\leq N\binom{t}{2} - N \\ \Leftrightarrow N &\leq (N-d)\binom{t}{2},\end{aligned}$$

a contradiction. □

In the next theorem we will make use of the following notion.

Definition 2.36. Let $A = (a_{i,j})$ be a $\mathsf{CA}(N; t, k, v)$. For any two rows $j_1, j_2 \in \{1, \dots, N\}$ of A we define

$$I(j_1, j_2) := |\{i \in \{1, \dots, k\} | a_{j_1,i} = a_{j_1,i}\}|,$$

the number of positions these two rows are equal, and

$$I(A) := \max\{I(j_1, j_2) | j_1 \neq j_2 \in \{1, \dots, N\},$$

as the maximum of these values over all selections of two different rows of A.

Theorem 2.37. *Let A be a $\mathsf{CA}(N; t, k, v)$ with $k/I(A) > \binom{t'}{2}$, then there exists a*

$$\mathsf{PHF}(k; N, v, t').$$

Proof:
Consider the code C that has as codewords the rows of A. Then C is a $(k, N, k - I(A), v)$ code. Since $k/I(A) > \binom{t'}{2} \Leftrightarrow k > (k-(k-I(A)))\binom{t'}{2}$ we can apply Theorem 2.35, to show the existence of a $\mathsf{PHF}(k; N, v, t')$. □

Corollary 2.38. *If there exists an $\mathsf{OA}_1(t, k, v)$ and an integer t' with $k/(t-1) > \binom{t'}{2}$, then there also exists a $\mathsf{PHF}(k; v^t, v, t')$.*

Proof:
Let A be an $\mathsf{OA}_1(t, k, v)$, to apply Theorem 2.37 we only have to show that $I(A) = t-1$. $I(A) \leq t-1$ is clear, since if $I(A) \geq t$ there would be a subarray comprised of t columns of A, where a certain $(v)_{i=1}^k$-ary t-tuple is covered more than once. A contradiction to A being an $\mathsf{OA}_1(t, k, v)$. $I(A) \geq t-1$ holds, as the t-tuples $(0, 0, \dots, 0)$ and $(1, 0, \dots, 0)$ need to be covered by the rows of the subarray comprised of the first t columns of A. Let j_1, j_2 be the rows that cover these two t-tuples, then we have $I(j_1, j_2) \geq t-1$ and hence $I(A) \geq t-1$. □

In combination with Theorem 2.2 we get the following.

Corollary 2.39. *For any prime power q and any integer r with $2 \leq r \leq q$, there exists a $\mathsf{PHF}(q; q^r, q, t')$ as long as $q/(r-1) > \binom{t'}{2}$.*

A Plug-In Constructions of CAs using PHFs

Note once more, that the plug-in construction in the proof of the following Theorem is column-wise.

Theorem 2.40. *Suppose there exists a $\mathsf{PHF}(s; k, m, t)$ and a $\mathsf{CA}(N; t, m, v)$, then there also exists a $\mathsf{CA}(sN; t, k, v)$, which can be constructed by replacing the i-th symbol of the PHF with the i-th column of the CA.*

Proof:
Let $B = (b_{i,j}) \in [m]^{s \times k}$ denote the $\mathsf{PHF}(s; k, m, t)$, and $A = (a_{i,j}) = (\mathbf{a}_1, \dots, \mathbf{a}_m) \in [v]^{N \times m}$ denote the $\mathsf{CA}(N; t, m, v)$. Consider the map $f : [m] \to \{\mathbf{a}_1, \dots, \mathbf{a}_m\} : \ell \mapsto \mathbf{a}_\ell$, and define C as the $sN \times k$ array over $[v]$ that results when replacing all entries of B with their image under f,

$$C := (\mathbf{c}_1, \dots, \mathbf{c}_k) := (f(b_{i,j}))_{(i,j) \in \{1,\dots,s\} \times \{1,\dots,k\}} \in [v]^{sN \times k}.$$

We show that C is a CA of strength t. Let $(\mathbf{c}_{i_1}, \dots, \mathbf{c}_{i_t})$ be a subarray comprised of t arbitrary columns of C. Since B is a t-perfect hash family, there exists a row j, so that the entries in $(b_{i_1,j}, \dots, b_{i_t,j})$ are pairwise different. Hence their images $f(b_{i_r,j}) = a_{i_r}$ for $r \in \{1, \dots, t\}$ are pairwise different columns of A. Since A is a CA of strength t, $(\mathbf{a}_{i_1}, \dots, \mathbf{a}_{i_t})$ is covering (recall Definition 1.13). As $(\mathbf{a}_{i_1}, \dots, \mathbf{a}_{i_t})$ appears as a subarray of $(\mathbf{c}_{i_1}, \dots, \mathbf{c}_{i_t})$, as expansion of the j-th row of B, also $(\mathbf{c}_{i_1}, \dots, \mathbf{c}_{i_t})$ is covering. $\square$

Example 2.41. As an example for the construction described in the last theorem consider the following $\mathsf{PHF}(2; 6, 4, 2)$ and the $\mathsf{CA}(5; 2, 4, 2)$ given in Table 2.11. For demonstration purposes we consider the PHF to be over the alphabet $\{a, b, c, d\}$. We also label the columns of the CA with the elements of $\{a, b, c, d\}$, so that each element labels the column that will replace the element.

Using Corollary 2.38 together with Theorem 2.40 it becomes possible to recursively construct CAs.

$\mathsf{CA}(5;2,4,2)$

a	b	c	d
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
0	0	1	1

$\times$

$\mathsf{PHF}(2;6,4,2)$

a	b	c	d	a	b
b	a	c	d	a	b

$=$

$\mathsf{CA} \times \mathsf{PHF}$

0	0	0	0	0	0
0	1	1	0	0	1
1	0	1	0	1	0
1	1	0	1	1	1
0	0	1	1	0	0
0	0	0	0	0	0
1	0	1	0	0	1
0	1	1	0	1	0
1	1	0	1	1	1
0	0	1	1	0	0

Figure 2.11.: The plug-in construction described in Example 2.41. The boxes emphasize the structure of the resulting $\mathsf{CA}(10;2,6,2)$.

Theorem 2.42. *Let q be a prime power. Suppose there exists a $\mathsf{CA}(N_0;t,q^{s_0},v)$ and $q^{s_0} > \binom{t}{2}$. Then there exists a $\mathsf{CA}(N_0R_i;t,q^{s_i},v)$, for all $i \geq 0$, where $R_0 = 1$ and*

$$\begin{aligned} R_i &= q^{s_{i-1}}R_{i-1}, \\ s_i &= s_{i-1}\left\lceil \frac{q^{s_{i-1}}}{\binom{t}{2}} \right\rceil, \end{aligned}$$

for all $i \geq 1$.

Proof:
We proceed by induction on i.
Induction base: For $i = 0$ we are given a $\mathsf{CA}(N_0;t,q^{s_0},v)$, since $R_0 = 1$ the assertion is holds.
Induction hypothesis: There exists a $\mathsf{CA}(N_0R_i;t,q^{s_i},v)$ with $R_i = q^{s_{i-1}}R_{i-1}$ and $s_i = s_{i-1}\left\lceil \frac{q^{s_{i-1}}}{\binom{t}{2}} \right\rceil$.
Induction step: $i \to t+1$:
Applying Corollary 2.39 with the prime power $q^{s_{i-1}}$ and $r = \left\lceil \frac{q^{s_{i-1}}}{\binom{t}{2}} \right\rceil$, we get the existence of a $\mathsf{PHF}(q^{s_{i-1}};q^{s_i},q^{s_{i-1}},t)$, as

$$\frac{q^{s_{i-1}}}{\left(\left\lceil \frac{q^{s_{i-1}}}{\binom{t}{2}} \right\rceil - 1\right)} > \frac{q^{s_{i-1}}}{\frac{q^{s_{i-1}}}{\binom{t}{2}}} = \binom{t}{2}$$

holds. By the induction hypothesis there exists a $\mathsf{CA}(N_0R_{i-1}; t, q^{s_{i-1}}, v)$. The column-wise plug-in of this $\mathsf{CA}(N_0R_{i-1}; t, q^{s_{i-1}}, v)$ into the $\mathsf{PHF}(q^{s_{i-1}}; q^{s_i}, q^{s_{i-1}}, t)$, according to Theorem 2.40 yields the desired $\mathsf{CA}(N_0R_i; t, q^{s_i}, v)$. $\square$

2.3. CAs as Families of Sets

There is a vast amount of structures in combinatorial design theory. Many of these are closely related, and some can be viewed from various aspects providing connections to different fields of scientific applications. In this section we consider certain families of sets, which, as will be shown, turn out to be equivalent to CAs. We will show that the size of a maximal 2-independent family of sets is completely determined, and such a family can explicitly be constructed, as proven in [50, 56]. In a second step, we will use the equivalence between these structures and CAs to completely address the problem of optimal $\mathsf{CA}(N; 2, k, 2)$ generation for arbitrary k.

2.3.1. Independent Families of Sets (IFSs)

The definitions given below are slightly different from the ones given in [21], and can also be found in [65].

Definition 2.43. A *t-independent family of sets* [6] $\mathsf{IFS}(N; t, k)$ is a family $(A_1, \dots, A_k)$ of k subsets of $[N]$, with the property that for each choice $\{i_1, \dots, i_t\} \subseteq \{1, \dots, k\}$ of t different indices and for all $\bar{A}_{i_j} \in \{A_{i_j}, A_{i_j}^C\}$ it holds that $\bigcap_{j=1}^{t} \bar{A}_{i_j} \neq \emptyset$. The parameters t and k are respectively called the *strength* and the *size* of the IFS.

We also say that a family of sets is *t-independent*, if it is an $\mathsf{IFS}(N; t, k)$ for some value of N and k without mentioning them explicitly, when it is clear from the context and need no further specification. Note that it is possible to define IFSs over arbitrary finite sets. Yet, since for our purposes, only the cardinality of the underlying set is of importance, we only consider IFSs over the underlying set $[N]$ for some $N \in \mathbb{N}$. In the following we use the notation $\bar{A}$ for a variable that can take the values A or $A^C = [N] \setminus A$, for a given set $[N]$.

[6]Note that in literature (e.g. in [50]) also the term *t-qualitatively independent family of sets* is used. Nevertheless we use the term *t-independent family of sets* also used by the authors of [56].

Example 2.44. An example of a 2-independent family of sets over [5], $\mathsf{IFS}(5; 2, 4)$, is given by $\mathcal{F} = (A_1, A_2, A_3, A_4)$, where

$$\begin{aligned} A_1 &= \{0, 1\}, \\ A_2 &= \{0, 2\}, \\ A_3 &= \{0, 3\}, \\ A_4 &= \{0, 4\} \end{aligned} \tag{2.8}$$

The check that all intersections $\bar{A}_i \cap \bar{A}_j$ for $A_i \neq A_j \in \mathcal{F}$, where $\bar{A} \in \{A, A^C\}$, are nonempty, is left to the reader.

Further we would like to motivate the nomenclature "independent family of sets" in the same manner, as Katona does in [50]. Considering two sets $A, B \subseteq [N]$ and the intersections

$$A \cap B,\ A^C \cap B,\ A \cap B^C,\ A^C \cap B^C. \tag{2.9}$$

If one of these intersections is empty, then the information $x \in A$ (or $x \notin A$), may already contain information whether x resides in B or not. Say for example $A \cap B^C = \emptyset$, then the information $x \in A$ contains the information $x \in B$. In the contrary case that none of the intersections in (2.9) is empty, the question after $x \in B$ can be answered independently from the answer to the question whether $x \in A$. In that regard, judging from (2.9), A and B are *independent.*
The typical question that arises with independent families of sets, is how large such families can get when the underlying set $[N]$ is fixed.

Definition 2.45. The largest number k such that an $\mathsf{IFS}(N; t, k)$ exists, is defined as

$$\mathsf{CAK}(N; t) := max\{k : \exists\ \mathsf{IFS}(N; t, k)\}.$$

As the similarity of the definitions of CAs and IFSs implies, there is a close relation between these two combinatorial structures. In fact, it is known that every $\mathsf{CA}(N; t, k, 2)$ is equivalent to an $\mathsf{IFS}(N; t, k)$ (see for example [65] and [21], Remark 10.5). We will make this connection explicit in the proof of the following theorem.

Theorem 2.46. *Every t-independent family of sets $\mathsf{IFS}(N; t, k)$ induces a binary covering array $\mathsf{CA}(N; t, k, 2)$ and vice versa.*

Proof:
Let $\mathcal{A} = (A_1, \dots, A_k)$ be an $\mathsf{IFS}(N; t, k)$. It is well-known that there exists an one-to-one correspondence between subsets of $\{0, \dots, N-1\}$ and binary vectors of length N (see also Chapter 4). Let $\mathfrak{P}(\{0, \dots, N-1\})$ denote the powerset of $\{0, \dots, N-1\}$. In particular, we have that:

$$\begin{aligned} \phi : \mathfrak{P}(\{1, \dots, N\}) &\to \{0,1\}^{(N\times 1)} \\ A &\mapsto (v_0, \dots, v_{N-1})^T, \text{ where } v_i = \begin{cases} 1, & i \in A \\ 0, & i \notin A \end{cases}, \end{aligned}$$

is a bijection (and in fact even an isomorphism of the Boolean algebras $(\mathfrak{P}([N]), \cap, \cup, .^C, \emptyset, [N])$ and $(\{0,1\}, \wedge, \vee, \neg, 0, 1)^N$, [90]).
By virtue of ϕ we can identify the sets A_i of $\mathcal{A}$ with their corresponding binary N tuples $\mathbf{a}_i$ for all $i \in \{1, \dots, k\}$. Let us now consider the matrix $M = (\mathbf{a}_1, \mathbf{a_2}, \dots, \mathbf{a}_k) = (a_{i,j})$ composed of the column-wise indicator vectors of the $A_i \in \mathcal{A}$. In other words, the j-th row of M has a 1-entry in column i if and only if $j \in A_i$ and a 0-entry otherwise. Let $\{i_1, \dots, i_t\} \subseteq \{1, \dots, k\}$ be a set of t different arbitrary indices, and $(u_1, \dots, u_t) \in \{0,1\}^t$ be an arbitrary binary t-tuple. We show that the $N \times t$ subarray $(\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \dots, \mathbf{a}_{i_t})$ of M covers the tuple $(u_1, \dots, u_t)$. Let us define

$$\bar{A}_{i_s} := \begin{cases} A_{i_s}, & \text{if } u_s = 1 \\ A_{i_s}^C, & \text{if } u_s = 0 \end{cases}, \forall s \in \{1, \dots, t\}.$$

Since $\mathcal{A}$ is a t-independent family of sets, it holds that

$$\bigcap_{s=1}^{t} \bar{A}_{i_s} \neq \emptyset.$$

Let $r \in \bigcap_{s=1}^{t} \bar{A}_{i_s}$, this is exactly the case if $r \in A_{i_s}$ for all $s \in \{1, \dots, t\}$ with $u_s = 1$ and $r \notin A_{i_s}$ for all $s \in \{1, \dots, t\}$ with $u_s = 0$, or equivalently that row r of $(\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \dots, \mathbf{a}_{i_t})$ equals $(u_1, \dots, u_t)$. Summarizing we have

$$r \in \bigcap_{s=1}^{t} \bar{A}_{i_s} \Rightarrow (a_{r,i_s})_{s=1}^{t} = (u_1, \dots, u_t), \tag{2.10}$$

which shows that M is a binary CA of strength t, as $(u_1, \dots, u_t)$ and $\{i_1, \dots, i_t\}$ were arbitrary.

For the reverse direction, let $M = (\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k)$ be a binary $\mathsf{CA}(N; t, k, 2)$. Define $\mathcal{A} = (A_1, \ldots, A_k)$ by $A_i := \phi^{-1}(\mathbf{m}_i)\ \forall i \in \{1, \ldots, k\}$. We want to show that an arbitrary intersection $\bigcap_{s=1}^{t} \bar{A}_{i_s}$, for $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ and some fixed $\bar{A}_{i_s} \in \{A_{i_s}, A_{i_s}^C\}$ is nonempty. Consider the binary t-tuple $u = (u_1, \ldots, u_t)$ where

$$u_s = \begin{cases} 1, \text{ if } \bar{A}_{i_s} = A_{i_s} \\ 0, \text{ if } \bar{A}_{i_s} = A_{i_s}^C \end{cases}, \ \forall s \in \{1, \ldots, t\}.$$

Since M is a binary $\mathsf{CA}(N; t, k, 2)$, there is a row r of $(\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k)$ that covers $(u_1, \ldots, u_t)$. By the definition of $(u_1, \ldots, u_t)$, we have

$$(u_1, \ldots, u_t) = (m_{r, i_s})_{s=1}^{t} \Rightarrow r \in \bigcap_{s=1}^{t} \bar{A}_{i_s} \neq \emptyset.$$

□

Example 2.44 (continuing from p. 58)**.** By virtue of the last theorem, we can map the $\mathsf{IFS}(5; 2, 4)$ defined by the equations in (2.9) to the following $\mathsf{CA}(5; 2, 4, 2)$:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.11}$$

An immediate consequence of the Theorem 2.46 is the following.

Corollary 2.47. *For $N,\ t,\ k \in \mathbb{N}$ with $k \geq t \geq 1$ we have*

$$\begin{aligned} CAN(t, k, 2) &= \min\{N | \exists\ \mathsf{IFS}(N; t, k)\}, \\ CAK(N; t) &= \max\{k | \exists\ \mathsf{CA}(N; t, k, 2)\}. \end{aligned}$$

2.3.2. Maximal 2-Independent Families of Sets

We will show that the size of a maximal 2-independent family of sets is completely determined. The main result of this section shares a lot of similarities with a well known Theorem of Sperner ([89]), which is the reason, why it is also referred to as *a Sperner-type theorem* ([57]). Before we give the proof of it, we introduce the necessary notions and theorems used in the proof.
The following definition can be also found in [2].

Definition 2.48. An *antichain* of a partially ordered set $(P, \preccurlyeq)$ is a subset $A \subseteq P$ such that any two different elements $a_1 \neq a_2 \in A$ are incomparable:

$$a_1 \not\preccurlyeq a_2 \text{ and } a_2 \not\preccurlyeq a_1.$$

The following theorem describes maximal antichains in power sets of finite sets and was proven due to Sperner in [89]; for an English source see also [2].

Theorem 2.49 (Sperner's Theorem)**.** *Let M be a finite set with $|M| = N$ elements. Then an antichain in the partially ordered set $(\mathfrak{P}(M), \subseteq)$ has at most $\binom{N}{\lfloor N/2 \rfloor}$ elements. A maximal antichain A with $|A| = \binom{N}{\lfloor N/2 \rfloor}$ can be constructed, by defining A as the set of all subsets of M with cardinality $\lfloor N/2 \rfloor$:*

$$A = \{S \big| S \subseteq M \wedge |S| = \lfloor N/2 \rfloor\}.$$

The following theorem is proven in [27], where we refer the interested reader for the proof.

Theorem 2.50 (Erdös-Ko-Rado Theorem)**.** *Let $\ell, m \in \mathbb{N}$ with $l \leq m/2$, then for any family of sets $\mathcal{F}$ with the properties*

(i) $\forall B \in \mathcal{F} : \ B \subseteq \{0, \ldots, m-1\}$,

(ii) $\forall B \in \mathcal{F} : \ |B| \leq \ell$,

(iii) $\forall B, D \in \mathcal{F} : B \not\subseteq D$,

(iv) $\forall B, D \in \mathcal{F} : B \cap D \neq \emptyset$,

it holds that $|\mathcal{F}| \leq \binom{m-1}{\ell-1}$.

The following theorem has been proven in the beginning of the 1970s by several authors (e.g. in [56, 50]). The proof we give here is similar to the one given in [56].

Theorem 2.51. *For every $N \geq 2$ it holds that*

$$\mathsf{CAK}(N; 2) = \binom{N-1}{\lfloor N/2 \rfloor - 1}.$$

More particular a maximal 2-independent family of sets over $\{0, \ldots, N-1\}$ is given by the family $\mathcal{F}_2(N)$ of all subsets of $\{0, \ldots, N-1\}$ of cardinality $\lfloor N/2 \rfloor$, all containing a fixed element.

Proof:
We distinguish two cases for N even and N odd.
<u>1st Case:</u> $N \equiv 0 \mod 2$.
Let $\mathcal{F} = \{B_1, \dots, B_k\}$ be a 2-independent family of sets and $B^C := \{0, \dots, N-1\} \setminus B$ denote the complement of B in $\{0, \dots, N-1\}$. Then $\mathcal{F}^* := \{B_1, \dots, B_k\} \cup \{B_1^C, \dots, B_k^C\}$ is an anti-chain in the partially ordered set $([N], \subseteq)$, as for all $i \neq j$ we have:

$$\begin{aligned} B_i \cap B_j \neq \emptyset &\Rightarrow B_i \not\subseteq B_j^C, \\ B_i^C \cap B_j^C \neq \emptyset &\Rightarrow B_i^C \not\subseteq B_j, \\ B_i \cap B_j^C \neq \emptyset &\Rightarrow B_i \not\subseteq B_j, \\ B_i^C \cap B_j \neq \emptyset &\Rightarrow B_i^C \not\subseteq B_j^C. \end{aligned}$$

From Sperners Theorem [89] we hence know

$$\begin{aligned} |\mathcal{F}^*| &= 2k \leq \binom{N}{\lfloor N/2 \rfloor} \\ \Leftrightarrow k &\leq \frac{1}{2}\binom{N}{\lfloor N/2 \rfloor} = \binom{N-1}{\lfloor N/2 \rfloor - 1}. \end{aligned}$$

Conversely assume given the maximal anti-chain $\mathcal{F}^*$ of $(\{1, \dots, N-1\}, \subseteq)$ consisting of all subsets of $\{1, \dots, N-1\}$ of size $\lfloor \frac{N-1}{2} \rfloor$ (see Sperner's Theorem [89]). Then $|\mathcal{F}^*| = \binom{N-1}{\lfloor \frac{N-1}{2} \rfloor} = \binom{N-1}{\lfloor N/2 \rfloor - 1}$, as N is even. Then for any $B_1 \neq B_2 \in \mathcal{F}^*$ from $B_1 \not\subseteq B_2$ we get $B_1 \cap B_2^C \neq \emptyset$, and from $B_2 \not\subseteq B_1$ we get $B_1^C \cap B_2 \neq \emptyset$. As $|B_1| = |B_2| = \lfloor \frac{N-1}{2} \rfloor$ we have $|B_1^C| = |B_2^C| = N-1-\lfloor \frac{N-1}{2} \rfloor = N-1-(\frac{N}{2}-1) = N/2$. Hence $|B_1^C| + |B_2^C| > N-1$ and B_1^C, B_2^C must intersect in at least one element: $B_1^C \cap B_2^C \neq \emptyset$. Finally, by adjoining one element to all sets in $\mathcal{F}^*$, by defining $\mathcal{F} := \{B \cup \{0\} | B \in \mathcal{F}^*\}$, we get $0 \in D_1 \cap D_2 \neq \emptyset$ for all $D_1, D_2 \in \mathcal{F}$. The other relations $D_1^C \cap D_2 \neq \emptyset$, $D_1 \cap D_2^C \neq \emptyset$ and $D_1^C \cap D_2^C \neq \emptyset$, transfer from the $B_i \in \mathcal{F}^*$ to the $D_i = B_i \cup \{0\} \in \mathcal{F}$, for which complements D_i^C are considered respectively $\{0, 1, \dots, N-1\}$. Hence $\mathcal{F}$ is a 2-independent family of sets over $[N]$, where all sets contain 0.
<u>2nd Case:</u> $N \equiv 1 \mod 2$.
Let $N = 2u+1$ with $u \in \mathbb{N}$. We define $\mathcal{F} := \{B \big| B \subseteq [N] \wedge 0 \in B \wedge |B| = u\}$ and show that $\mathcal{F}$ is a 2-independent family of sets with, as can be seen easily, $\binom{2u}{u} = \binom{N-1}{\lfloor N/2 \rfloor - 1}$ elements. Let $B \neq D \in \mathcal{F}$, then we heave

<u>$B \cap D \neq \emptyset$:</u> as $B \cap D \supseteq \{0\} \neq \emptyset$

$\underline{B^C \cap D^C \neq \emptyset}$: as $B^C \subseteq \{1, \ldots, N-1\}$ and $|B^C| = u+1$. The same holds for D^C. Hence B^C and D^C must intersect on at least one of the $2u$ elements of $\{1, \ldots, N-1\}$.

$\underline{B^C \cap D \neq \emptyset}$: As $B \neq D$ we have $B^C \neq D^C$ and hence, as $|B^C| = |D^C|$, we have $B^C \not\subseteq D^C$. Therefore $B^C \cap D \neq \emptyset$.

$\underline{B \cap D^C \neq \emptyset}$: analogue to $B^C \cap D \neq \emptyset$.

Conversely assume given a 2-independent family of sets $\mathcal{F}$. As we are only interested in the cardinality of $\mathcal{F}$, we may assume without loss of generality that each $B \in \mathcal{F}$ fulfills $|B| \leq u$, since if $|B| > u$ we can always remove B from $\mathcal{F}$ and replace it with B^c having $|B^C| = 2u + 1 - |B| \leq u = \lfloor N/2 \rfloor$ elements. Applying the Erös-Ko-Rado Theorem 2.50 to $\mathcal{F}$ we immediately get $|\mathcal{F}| \leq \binom{2u}{u-1} = \binom{N-1}{\lfloor N/2 \rfloor - 1}$. □

Combining Theorem 2.46 and Corollary 2.47 we immediately get the following.

Corollary 2.52. *For every $k \geq 1$ it holds that*

$$\mathsf{CAN}(2, k, 2) = \min\left\{N \middle|\ k \leq \binom{N-1}{\lfloor N/2 \rfloor - 1}\right\}.$$

Further an optimal $\mathsf{CA}(N; 2, k, 2)$ can be constructed by horizontally juxtaposing k different binary column vectors of length N, each having a 1-entry in the first position, and $\lfloor N/2 \rfloor$ 1-entries in total.

Example 2.44 (continuing from p. 58)**.** As $\mathsf{CAK}(5; 2) = \binom{5-1}{\lfloor 5/2 \rfloor - 1} = 4$, the IFS defined by the equations of (2.9) and hence also the binary CA in (2.11), are optimal.

2.4. MCAs as Families of Partitions

In this section we generalize the representation of CAs as families of sets to MCAs. To this end, we have to consider a generalization of independent families of stets to independent families of partitions (IFPs).

Definition 2.53. We call a family $P_1, P_2, \ldots, P_k$ of partitions of the set $\{1, \ldots, N\}$, with $|P_r| = v_r$, a t-independent family of partitions, if for all subsets $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ all $\prod_{j=1}^{t} v_j$ intersections $\bigcap_{j=1}^{t} A_{i_j}$, with $A_{i_j} \in P_{i_j}$, are non-empty. Such a family of partitions is denoted as $\mathsf{IFP}(N; t, k, (v_1, \ldots, v_k))$, and the parameters t and k are called respectively *strength* and *size* of the IFP.

As was the case for IFSs and CAs, there is also a strong connection between IFPs and MCAs. This connection was also mentioned in [65] and [21], and can be stated as follows. Note that to the best of the author's knowledge a proof was never given explicitly in the literature.

Theorem 2.54. *Every t-independent family of partitions* $\mathsf{IFP}(N;t,k,(v_1,\dots,v_k))$ *yields a mixed-level t-covering array* $\mathsf{MCA}(N;t,k,((v_1,\dots,v_k))$ *and vice versa.*

Proof:
Let $(P_1,P_2,\dots,P_k)$ be an $\mathsf{IFP}(N;t,k,(v_1,\dots,v_k))$, where $P_j=\{A_{j,0},\dots,A_{j,v_j-1}\}$ is a partition of the set $\{1,\dots,N\}$ for all $j\in\{1,\dots,k\}$. We define the $N\times k$ matrix $M=(m_{i,j})$ as:

$$m_{i,j}=\ell \Leftrightarrow i\in A_{j,\ell}.$$

M can be considered the matrix of the column-wise generalized indicator vectors of the P_j's. It follows, that the entries of the j-th column of M are elements of $[v_j]$. Let us consider a set $\{i_1,\dots,i_t\}\subseteq\{1,\dots,k\}$ and a t-tuple $u=(u_1,\dots,u_t)\in\prod_{j=1}^{t}[v_{i_j}]$. Since $P_1,P_2,\dots,P_k$ is a t-independent family of partitions, there exists an $r\in\{1,\dots,N\}$:

$$r\in\bigcap_{j=1}^{t}A_{i_j,u_j}\neq\emptyset,$$

i.e. in row r of the columns $i_1,\dots,i_t$ of M the t-tuple $u=(u_1,\dots,u_t)$ is covered. This proves M being an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$.
If we are given an $MCA(N;t,k,(v_1,\dots,v_k))$, consider the following family $(P_1,\dots,P_k)$ of subsets of $\{1,\dots,N\}$, defined by $P_j=\{A_{j,0},\dots,A_{j,v_j-1}\}$, where

$$i\in A_{j,\ell}\Leftrightarrow M_{i,j}=\ell. \tag{2.12}$$

Hence M being again the matrix of column-wise generalized indicator vectors of the P_j's. For j fixed, the $A_{j,0},\dots,A_{j,v_j-1}$ are disjoint, as the entries of M are well defined, and they satisfy $\bigcup_{r=0}^{v_j-1}A_{j,r}=[v_j]$, as M is an MCA of strength $t\geq 1$. So $(P_1,\dots,P_k)$ is a family of partitions of $\{1,\dots,N\}$. Considering any intersection $\bigcap_{j=1}^{t}A_{i_j,u_j}$ with $\{i_1,\dots,i_t\}\subseteq\{1,\dots,k\}$, and $(u_1,\dots,u_t)\in\prod_{j=1}^{t}[v_{i_j}]$ arbitrary Since M is an $\mathsf{MCA}(N;t,k,(v_1,\dots,v_k))$, the t-tuple u is covered by some row r in the columns $i_1,\dots,i_t$ of M. By (2.12) we get $r\in\bigcap_{j=1}^{t}A_{i_j,u_j}\neq\emptyset$. This proves $P_1,\dots,P_k$ to be a t-independent family of Partitions.
Consider the following example that makes this connection more concrete.

Example 2.55. Given a family (P_1, P_2, P_3, P_4) of four Partitions of the set $\{1, \ldots, 15\}$, whereat

$$
\begin{aligned}
P_1 &= \{\underbrace{\{2,4,5.6,7\}}_{A_{1,0}}, \underbrace{\{1,9,10,12,14\}}_{A_{1,1}}, \underbrace{\{3,8,11,13,15\}}_{A_{1,2}}\},\\
P_2 &= \{\underbrace{\{1,2,3,4,5,6\}}_{A_{2,0}}, \underbrace{\{7,8,9,10,11,12,13,14,15\}}_{A_{2,1}}\},\\
P_3 &= \{\underbrace{\{4,5,6,8,9\}}_{A_{3,0}}, \underbrace{\{3,7,10,12,14\}}_{A_{3,1}}, \underbrace{\{1,2,11,13,15\}}_{A_{3,2}}\},\\
P_4 &= \{\underbrace{\{1,7,8\}}_{A_{4,0}}, \underbrace{\{2,3,9\}}_{A_{4,1}}, \underbrace{\{4,10,11\}}_{A_{4,2}}, \underbrace{\{5,12,13\}}_{A_{4,3}}, \underbrace{\{6,14,15\}}_{A_{4,4}}\}
\end{aligned}
$$

Table 2.12 shows the corresponding matrix $M = (\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4) = (m_{i,j})$ of the column-wise generalized indicator vectors. For instance $m_{5,4} = 3$ means that 5 is an element of $A_{4,3}$. Considering the partitions as families rather than sets, we could say that $m_{5,4} = 3$ means that 5 is an element of the 3rd set of the fourth partition P_4. In fact the given family of partitions is 2-independent, i.e. an $\mathsf{IFP}(15; 2, 4, (3, 2, 3, 5))$, or equivalently M an MCA over $(3, 2, 3, 5)$ of strength 2, i.e. an $\mathsf{MCA}(15; 2, 4, (3, 2, 3, 5))$. Let us now reconstruct the relation between the coverage property of M and the independence property of (P_1, P_2, P_3, P_4) with the help of an example. We choose the set $\{1, 4\} \subseteq \{1, \ldots, 4\}$ of cardinality 2. For demonstration purposes we choose the $(3, 2, 3, 5)$-ary 2-tuple $((1, 4), (1, 3))$. Now we have

$$(1, 3) \text{ is covered by row 12 of } M \Leftrightarrow 12 \in A_{1,1} \cap A_{4,3}.$$

We will use the connection between binary CAs and IFSs in the second part of this thesis, in Chapter 5. Where in Chapter 4, we will establish a different connection between CAs and families of sets.

	$P_1 = (A_{1,0}, A_{1,1}, A_{1,2})$	$P_2 = (A_{2,0}, A_{2,1})$	$P_3 = (A_{3,0}, A_{3,1}, A_{3,2})$	$P_4 = (A_{4,0}, A_{4,1}, A_{4,2}, A_{4,3}, A_{4,4})$
1	1	0	2	0
2	0	0	2	1
3	2	0	1	1
4	0	0	0	2
5	0	0	0	3
6	0	0	0	4
7	0	1	1	0
8	2	1	0	0
9	1	1	0	1
10	1	1	1	2
11	2	1	2	2
12	$\textcircled{1}$	1	1	$\textcircled{3}$
13	2	1	2	3
14	1	1	1	4
15	2	1	2	4

Figure 2.12.: An $\mathsf{MCA}(15, 4, 2, (3, 2, 3, 5))$, where the rows are labeled with the elements of $\{1, \ldots, N\}$, and each columns with its corresponding partition of $\{1, \ldots, N\}$, referring to Example 2.55. The $(3, 2, 3, 5)$-ary 2-tuple $((1, 4), (1, 3))$ covered in the 12-th row is highlighted.

Part II.

Algorithms for Covering Arrays

3. An Overview of Algorithms for CA generation

This chapter gives an overview of various algorithmic approaches for CA generation. The notorious difficulty of constructing optimal CAs has been the subject of many algorithmic approaches. Some closely related NP-complete problems (e.g. [6], [26],[73] and [84]) suggest that the problem of finding an optimal covering array is a hard combinatorial optimization problem. In general, there is still no known strategy to efficiently construct covering arrays with the smallest number of rows, nor to determine the covering array number for specific covering array parameters. Despite the effort expended by numerous researchers, finding optimal CAs remains a challenging problem, as much research has been devoted to finding approximations to CAs with the smallest number of rows via related algorithmic approaches. The works [61, 62, 92, 15] provide surveys of such CA generation methods. To give an exhaustive overview of all these approaches is beyond the scope of this writing.
Instead, in this chapter we will give an overview of some of the algorithmic methodologies used for CA generation. Amongst these methodologies are greedy heuristics ([22]), metaheuristic approaches, as *simulated annealing* ([52]) or *tabu search* ([31, 32]), as well as combinations of these algorithmic approaches with theoretical constructions.

3.1. Greedy Algorithms for CA Generation

Among the most popular strategies for CA generation, greedy methods are the most popular one. According to [22]:" A greedy algorithm always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution." There are several ways to generate CAs *one step at a time*. Greedy algorithms for CA generation usually start from an empty array, or from a trivial CA (see Theorem 1.22 (ii)), then growing the array vertically (row by row), horizontally (column by column) or both, one column or row at a time.

3.1.1. Vertical Greedy Algorithms

A general algorithmic framework for greedy methods for CA construction can be found in [7] and [9]. The most popular greedy algorithms that grow CAs row by row are the AETG algorithm [13] and a *deterministic density algorithm* (DDA) [6, 7]. Both algorithms are given as input the strength t as well as the alphabet sizes $(v_1, \dots, v_k)$ of the desired $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$ and start with an initially empty array, proceeding in steps, adding one row at a time. The AETG algorithm produces a beforehand fixed number of rows at random, selecting the row, which covers the most number of yet uncovered $\mathbf{v}$-ary t-tuples, from these randomly chosen ones. The algorithm stops, when there are no more $\mathbf{v}$-ary t-tuples left to be covered. For further details see [13], where also another heuristic is mentioned, where in each step, from all vectors over $(v_1, \dots, v_k)$, one is selected that covers the most $\mathbf{v}$-ary t-tuples that are not yet covered by the current array. The strategy of DDA on the other hand is to add a row to the produced array, that is of at least average quality, in terms of covering yet uncovered $\mathbf{v}$-ary t-tuples. Again, the algorithm stops, when there are no more $\mathbf{v}$-ary t-tuples left to be covered. We will take a closer look at the latter two strategies in Chapter 4.

3.1.2. Horizontal Greedy Algorithms

The greedy algorithms belonging to the IFS-family ([30, 48]) of algorithms construct *independent families of sets*. As we will see in Chapter 5, when this methodology is translated in terms of arrays, this means that these algorithms grow binary CAs column by column. In that sense, the input to these algorithms is the number of rows N and the strength t of the desired $\mathsf{CA}(N; t, k, 2)$. Additionally some restrictions on the number of appearing binary i-tuples for $i \leq t$ have to be imposed. The algorithm starts with a randomly selected binary column vector of length N, with $\lfloor N/2 \rfloor$ 1-entries. In each step of the algorithm these restrictions are used to determine which columns can be used to grow the array at hand, ensuring that the array stays a CA of strength t. The algorithm stops, when there is no more column that can be added to the generated array, so that it maintains a CA of strength t.

3.1.3. Two Dimensional Growth

Algorithms belonging to the IPO-family [66, 67] of algorithms, grow CAs in two dimensions. The input to these algorithms is the desired strength t as well as the alphabet sizes $(v_1, \dots, v_k)$ underlying the columns of the desired MCA. Some algorithms optionally can also handle constraints, such as forbidding specific $\mathbf{v}$-ary t-tuples to be covered by the array. The method common to these algorithms is to start with a $t \times (\prod_{i=1}^{t} v_i)$ array of all vectors over $(v_1, \dots, v_t)$. From there on the algorithms proceed in two phases, alternating each other, the horizontal extension (adding a column) and the vertical extension (adding rows), until the desired CA with k columns over $(v_1, \dots, v_k)$ has been constructed. Thereby it is ensured that the result of the vertical extension phase is always a CA. The IPOG algorithm can be described informally as follows. In the horizontal extension step an initially empty column is added to the current array A, which is a CA, say with i columns. For each row of A a value for the $(i+1)$-st column is selected, such that the number of newly covered $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples is maximal under all possible values that can be selected for this position.
In case all $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples can be covered in this way, there is no vertical extension phase, and the algorithm proceeds by adding the next column to the array. Otherwise the remaining uncovered $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples have to be covered, where for each such tuple τ the algorithm tries to find a row of A, where unspecified values can be set such that τ is covered by this row. If no such row exists, a new row that covers τ and which is unspecified in other positions, is added to A.
This procedure is repeated until all $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples are covered, hence the current array is a $\mathsf{CA}(N; t, i+1, (v_1, \dots, v_i, v_{i+1}))$ with some unspecified values, representing *don't care* values. The algorithm enters again the horizontal extension phase, unless k columns are already reached.
As the first i columns of the generated array always constitute a $\mathsf{CA}(N; t, i, (v_1, \dots, v_i))$ when adding the $i+1$st column, the only $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples of interest in these steps are those involving the position $i+1$. Such tuples correspond to selections of t columns, involving the newly added $i+1$st column.

Algorithmic Procedure 3 (IPOG)**.** To construct an $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$, on input $t, k, (v_1, \dots, v_k)$, proceed as follows:

Step 1: Initialize an array having as rows all vectors of $\prod_{i=1}^{t}[v_i]$.

Step 2: While the number i of columns of A is less than k, repeat steps 3 to 4.

Step 3: Extend A with a column having alphabet size v_{i+1}, with initially unspecified entries.

Step 4: As long as uncovered $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples remain, iterate over the rows of A, specifying the unspecified values of the $i+1$st column such that each row covers the maximal number of the yet uncovered $(v_1, \dots, v_i, v_{i+1})$-ary t-tuples.

Step 5: Every such uncovered $(v_1, \dots, v_i, v_{i+1})$-ary t-tuple τ is now covered by either

(a) specifing unspecified values of A such that τ gets covered by a row, or

(b) in case there does not exist such a row, a new row that covers τ and is unspecified in the other positions, is added to A.

Step 6: Once A has k columns, return A.

3.2. Metaheuristic Methods

Other than greedy heuristics, a number of works use metaheuristics to design algorithms for CA construction, as [15, 93], using simulated annealing, the work in [15], using hill climbing and [75], using tabu search. Before we briefly summarize these methodologies, we introduce the necessary terminologies to do so.
Simulated annealing, hill climbing and tabu search are closely related search techniques, as all of them rely on a *set of feasible solutions* Σ and certain *costs* $c(S)$ associated to each $S \in \Sigma$, to specify an optimization problem. Then, an optimal solution to the problem corresponds to a feasible solution S with minimal (or maximal) cost $c(S)$. With regard to CAs a feasible solution could be an array over the desired alphabet, and the cost of an array could be the number of uncovered tuples, as used for example in [15, 93]. Then an array of cost zero is a CA. For each solution $S \in \Sigma$ a set of transitions T_S is defined, where each such transition transforms the current solution to another feasible solution. The set of feasible solutions that can be reached from S via transitions of T_S is called the *neighborhood* $N(S)$ of S. Again, for CAs, such transitions could be modifying single entries of an array, see [15, 93].

In the following we describe how the mentioned metaheuristics can be applied for the search of a $\mathsf{CA}(N; t, k, v)$, although they can also be applied for the search for $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$, [15].

Hill Climbing

For the hill climbing approach presented in [15], initially a randomly generated feasible solution is chosen, in this case a random $N \times k$ array over $[v]$. From the current feasible solution S a random transition is performed to obtain a feasible solution S'. In case of $c(S') \leq c(S)$, S' is accepted as the new current feasible solution, and is rejected otherwise. This enables the algorithm to randomly traverse Σ, decreasing the cost of the solution at hand. To prevent the algorithm from getting stuck in local minima, an upper bound on the number of randomly selected transitions from the current solution S can be used, which when reached, the algorithm aborts and returns the current feasible solution S. Note that, with a cost function as described above, S is a CA, if and only if $c(S) = 0$. The whole process can be iterated, each time starting with a random initial feasible solution, to obtain several random walks through Σ, trying to increase the chances of finding a CA.

Simulated Annealing

The approaches based on simulated annealing proposed in [15, 93], both also start from randomly generated feasible solutions S, i.e. a randomly generated array over the appropriate alphabet. As for the previously described hill climbing approach, in each step a neighbor $S' \in N(S)$ is computed, via selecting a random transition from T_S. In case the cost $c(S')$ is lower or equal to the cost $c(S)$, S' is accepted as current feasible solution. To avoid that the algorithm from getting stuck in local minima, in the case of $c(S') > c(S)$, S' might still be accepted as current feasible solution, but only with a certain probability, which decreases over time. As described in [15], this probability can be computed via $\exp(-(c(S') - c(S))T_i/K$, where T_i is called the *temperature* of the system in iteration i and K is a constant. This temperature is decreased in each iteration $T_{i+1} := T_i\alpha$ for a given α being part of the input and satisfying $0 < \alpha < 1$. After an appropriate stopping criterion is met, e.g. $c(S) = 0$, the number of iterations bypassed a beforehand given bound or the temperature reaches a given value T_f, the current solution is returned as an approximation to the solution to the given problem. Again, the whole process can be iterated to increase the chance of finding a solution to the problem.

Tabu Search

The work in [75] uses an algorithm based on tabu search for computing covering arrays. The algorithm starts also by constructing a random $N \times k$ array S over $[v]$. In each step of the algorithm a $(v)_{i=1}^{k}$-ary t-tuple that is not covered by the rows of S is selected at random. Next it checks which rows of S require only the change of a single entry to cover the $(v)_{i=1}^{k}$-ary t-tuple at hand. Each of these changes correspond to a transition, and the resulting arrays of these changes constitute the neighborhood $N(S)$ of S. The costs of the neighbors $S' \in N(S)$, in this case the number of uncovered $(v)_{i=1}^{k}$-ary t-tuples, is computed, and a neighbor of minimal cost is selected, in case this move is not a *tabu*. To prevent the algorithm from getting stuck in local minima, a *tabu list* of an a priori specified length T (in [75] typically $1 \leq T \leq 10$) is maintained. The tabu list prevents the algorithm to make any transitions that would change any of the T latest modified entries of the array, which enables the algorithm to escape from local minima. In case that $N(S) = \emptyset$ for the current solution S, i.e. more than a single entry in the array needs to be changed to cover the $(v)_{i=1}^{k}$-ary t-tuple at hand, on each row of S is selected at random and modified so that the tuple is covered.
Finally it is worthwhile to mention that all of the previously described algorithms depend on the input of N, the number of rows of the desired $\mathsf{CA}(N; t, k, v)$. This means that for values $N < \mathsf{CAN}(t, k, v)$, these algorithms are doomed to fail, as for such N there exists no $\mathsf{CA}(N; t, k, v)$. On the other hand, if the input N is chosen to large, a potentially returned $\mathsf{CA}(N; t, k, v)$ might have much more rows than an optimal CA for (t, k, v). The problem is of course that $\mathsf{CAN}(t, k, v)$ is only known for some special cases as discussed in Part I of this thesis (Chapter 2). We hence, do not know a priori for which values of N there even exists a $\mathsf{CA}(N; t, k, v)$. In [15] upper and lower bounds for N are used to apply binary search in the created interval, to obtain values for the input of N to start an algorithm based on hill climbing. This method could also be applied to the described algorithms, based on simulated annealing and tabu search. As an upper bound the result of Corollary 1.28 could be used, and v^t as a lower bound, when searching for a $\mathsf{CA}(N; t, k, v)$.
Another limitation of the approaches described above are computational resources such as time and memory. To extend the boundaries of the usability of metaheuristic

approaches, one way is to combine them with theoretical constructions for CAs, yielding hybrid methods, as we briefly picture in the next section.

3.3. Hybrid Methods

Due to the computational difficulties arising with the construction of optimal CAs, the metaheuristic methods previously described degrade in speed as the problem size increases and hence only work in a reasonable amount of time up to certain sizes of CA parameters (i.e. strength, number of columns and alphabet sizes). For constructing larger arrays, there are some approaches that combine combinatorial constructions, in this case plug-in constructions comparable to the one used in Theorem 2.40, with computational search. For example the authors of [16] successfully use recursive combinatorial constructions, such as plug-in constructions for CAs, in combination with computational search based on simulated annealing. While benefiting from the generality of heuristic search, the combinatorial constructions enable the generation of CAs for higher values of t, k or v.

3.4. Exact Approaches

Finally it is worthwhile to mention that there exists some literature on applying *exact methods* for optimal CA generation, such as [38, 88, 96, 97], using constraint programming, integer programming, backtracking or SAT approaches. Although these methods are capable of producing optimal CAs, they suffer from the problem of combinatorial explosion for larger CA parameters, and hence find these solutions only for moderate problem sizes, i.e. CA parameters.

4. CAs as Cover Problems

So far, in this work we treated CAs as discrete structures appearing in combinatorial design theory, just as they are introduced in literature most frequently, i.e. being arrays with certain coverage properties. In this chapter, largely following the work in [47], we will regard CAs from a different point of view, namely under the purview of *set covers* (SCs). This and related approaches are not new and have already been used implicitly or explicitly by some authors (e.g. [35, 96]).
We proceed by first specifying the necessary notions, structures and the interplay between them, before providing mappings that translate CA parameters, to an instance of a set cover problem. As a solution to the latter can be transformed to a CA, by virtue of the appropriate backwards translation, a connection between CAs and SCs is established. Thereafter we use this connection to generalize the former mappings for generalizations of CAs, such as *variable strength covering arrays* and *weighted budgeted covering arrays* in subsections 4.2.2 and 4.2.3. The connection between these structures also enables us to import certain bounds for the quality of SC heuristics, to attain upper bounds on the number of rows in the case of CAs and a lower bound on the weight of covered tuples in the case of weighted budgeted covering arrays. Finally we will compare two greedy methods for CA generation, one having an analogue for SCs and the other one being specialized for CA generation.

4.1. Set Covers and Integer Programming

Next we define set covers and integer programs, before we describe the close relation between these two notions.

4.1.1. Set Covers

Set covers have been heavily researched by both, mathematicians and computer scientists in the past. Consequently they are subject to many works in these fields.

The next definition follows the one given in [49].

Definition 4.1 (Set Cover (SC))**.** A *set cover* (SC) of a finite set U is a set $\mathcal{S}$ of nonempty subsets of U whose union is U. In this context, we call U the *universe* and refer to the elements of $\mathcal{S}$ as *blocks*. A SC consisting of pairwise disjoint blocks is called an *exact cover*, while an SC consisting only of blocks of cardinality d is called a *d-set cover*.

The typical problems that arise with SCs are the following:

Problem 4.2 (Minimal Set Cover (MSC))**.** Given a finite set U and a set cover $\mathcal{S}$ of U, i.e. $\bigcup \mathcal{S} = U$, find one subset $\mathcal{C}$ of $\mathcal{S}$ of minimal cardinality such that $\bigcup \mathcal{C} = U$. $(U, \mathcal{S})$ is also called the *input to the MSC problem.*

Problem 4.3 (Weighted Minimal Set Cover (Weighted MSC))**.** Given a finite set U, a set cover $\mathcal{S} = \{S_1, \ldots, S_r\}$ of U and a *cost vector* $c = (c_{S_1}, \ldots, c_{S_r})$, find one subset $\mathcal{C}$ of $\mathcal{S}$ with minimal cost such that $\bigcup \mathcal{C} = U$, where the cost of a $\mathcal{C} \subseteq \mathcal{S}$ is defined as $cost(\mathcal{C}) := \sum_{S \in \mathcal{C}} c_S$. $(U, \mathcal{S}, c)$ is also called the *input to the weighted MSC problem.*

4.1.2. Integer Programming

The following definition is taken from [74].

Definition 4.4. An *integer programming problem*[1] is the problem of determining

$$\max_{x \in \mathbb{N}^r} \{cx | Ax \leq b\}, \tag{4.1}$$

where $c \in \mathbb{Z}^{1 \times r}$ is called the *cost vector*, $A \in \mathbb{Z}^{n \times r}$ and $b \in \mathbb{Z}^{n \times 1}$. the triple (c, A, b) is called the *input to the IP problem.*

Since $min_{x \in \mathbb{N}^r}\{cx|Ax \geq b\} = -max_{x \in \mathbb{N}^r}\{(-c)x|(-A)x \leq (-b)\}$, if one of them exist, we are going to use the equivalent formulation of an integer programming problem on the left hand side. Moreover we will only need to consider the following special case of integer programming problems.

[1]Note that in literature, this is also known as *linear (pure) integer program.* As this thesis focuses exclusively on this specific kind of integer programs, we elect to use this shorter, albeit possibly nonstandard terminology.

Problem 4.5 (Integer Programming (IP))**.** A $0-1$ *integer programming problem* (IP problem) is the problem of finding

$$\min_{x\in\{0,1\}^{r\times 1}}\{cx|Ax \geq b\}, \tag{4.2}$$

where $c \in \mathbb{Z}^{1\times r}$, is called the *cost vector*, $A \in \{0,1\}^{n\times r}$ and $b \in \{0,1\}^{n\times 1}$. (c, A, b) is also called the *input* to the IP problem.

4.1.3. Set Covers and Integer Programming

As subsets of a finite set with n elements can be identified with binary vectors of length n, there is a close connection between MSC problems and IP problems. We will briefly make this connection explicit and introduce some notations in the process.

For a finite set $X = \{x_1, \dots, x_n\}$ of n elements, each subset $A \subseteq X$ can be mapped to its *indicator vector* $ind_X(A) \in \{0,1\}^n$. Conversely, each $x \in \{0,1\}^n$ can be mapped to its *support* $supp_n(x) \subseteq \{x_1, \dots, x_n\}$. As $ind_X \circ supp_{|X|} = id_{\{0,1\}^{|X|}}$ and $supp_{|X|} \circ ind_X = id_{\mathfrak{P}(X)}$, the two functions:

$$\begin{aligned}
ind_X : \mathfrak{P}(X) &\to \{0,1\}^{|X|} \\
A &\mapsto ind(A) := (a_1, \dots, a_n) \text{ with } a_i = \begin{cases} 1, & x_i \in A \\ 0, & x_i \notin A \end{cases}, \\
supp_{|X|} : \{0,1\}^{|X|} &\to \mathfrak{P}(X) \\
(a_1, \dots, a_n) &\mapsto A = \{x_i | a_i = 1\}.
\end{aligned}$$

are inverse to each other. We can use these functions to map the input $(U, \mathcal{S}, c)$ of an arbitrary weighted MSC problem to the input (c, A, b) of a $0-1$ IP problem. Suppose $U = \{u_1, \dots, u_n\}$, $\mathcal{S} = (S_1, \dots, S_r)$ and c are given. We can define a matrix A via the horizontal concatenation of the column-wise indicator vectors of the blocks of $\mathcal{S}$, i.e. $A := (ind(S_1), \dots, ind(S_r)) \in \{0,1\}^{n\times r}$. Let further denote $b := \mathbb{1} := \{1\}^{n\times 1}$ the all-one vector of appropriate dimension. Now any solution x to the IP problem $\min_{x\in\{0,1\}^{r\times 1}}\{cx|Ax \geq \mathbb{1}\}$ can be mapped to a solution of the original weighted MSC problem via

$$\begin{aligned}
supp : \{0,1\}^r &\to \mathfrak{P}(\mathcal{S}) \\
x &\mapsto supp(x) =: \mathcal{C}.
\end{aligned}$$

That $\mathcal{C}$ is indeed a SC of U is ensured by $Ax \geq \mathbb{1}$ and $S_i \in \mathcal{C} \Leftrightarrow x_i = 1$, for all $i \in \{1, \dots, r\}$.

For the other direction, any input (c, A, b) to a $0{-}1$ IP problem $\min_{x \in \{0,1\}^{r\times 1}} \{cx | Ax \geq \mathbb{1}\}$, where $A = (a_1, \dots, a_r) \in \{0,1\}^{n\times r}$, can be mapped to a weighted MSC with the input $(U, \mathcal{S}, c)$, where the universe U consists of n elements, and the set of blocks $\mathcal{S} = \{supp(a_1), \dots, supp(a_r)\}$ consists of the supports of the columns of A. Then a solution $\mathcal{C}$ to the weighted MSC $(U, \mathcal{S}, c)$ can be mapped to a solution to the IP problem via $ind : \mathfrak{P}(\mathcal{S}) \to \{0,1\}^r : \mathcal{C} \mapsto ind(\mathcal{C})$, as $Ax \geq \mathbb{1}$ is equivalent to the condition that the whole universe U is covered by $\mathcal{C}$, and the minimality of the cost of $\mathcal{C}$ over all subsets of $\mathcal{S}$ is equivalent to the minimality of cx over all $x \in \{0,1\}^r$.

Remark 4.6. We say that a SC problem and an IP problem are *corresponding*, if one can be mapped to the respective other via the function *ind*, respectively *supp*, as just described.

In fact the just described connection between these two problems is so *natural* that the weighted MSC problem is often defined as its corresponding $0-1$ IP problem, as, for example, in [3]:

Problem 4.7 (Weighted MSC via IP)**.** The weighted minimal set cover problem is the problem of determining

$$\min_{x \in \{0,1\}^{r\times 1}} \{cx | Ax \geq \mathbb{1}\}, \tag{4.3}$$

where $A \in \{0,1\}^{n\times r}$, $\mathbb{1} = \{1\}^{n\times 1}$ and $c \in \mathbb{Z}^{1\times r}$.

4.2. Formulating Covering Arrays as Set Covers

In subsequent sections we consider various problem statements regarding MCAs and other generalizations of CAs. In cases where there is no chance for ambiguity, we refer to these problems as *CA problems*, although they are not exclusively stated for CAs in the strict sense of Definition 1.2, but also for generalizations of CAs. Similarly, we will refer with *SC problems* to problems regarding SCs.

4.2.1. Mapping MCAs to SCs

To the best of the author's knowledge, a formulation of the optimal CA problem in terms of SCs, was first mentioned for CAs in [88] as part of a private conversation

with D. Applegate. Later in [96] a more thorough treatment of the topic was given, as the authors also investigate the feasibility of using an IP approach to solve the optimal MCA generation problem.
In the following we will translate the OMCA Problem (c.f. Problem 1.16), to an MSC problem via an algorithmic construction. For this purpose we require two more notations, extending the notion of $\mathbf{v}$-ary t-tuples (recall Definition 1.17).

Definition 4.8. For positive integers t, k and $v_1, \ldots, v_k$ with $t \leq k$ and $\mathbf{v} = (v_1, \ldots, v_k)$, we denote the set of all $(v_1, \ldots, v_k)$-ary t-tuples with $\mathbb{T}_{\mathbf{v},t}$. We further denote with $\varphi_{\mathbf{v},t}$ the function which maps each $w \in \prod_{i=1}^{k}\{0, \ldots, v_i - 1\}$ to the set of $\binom{k}{t}$ $(v_1, \ldots, v_k)$-ary t-tuples that are covered by w (recall Definition 1.18):

$$\begin{aligned} \varphi_{\mathbf{v},t} : \prod_{i=1}^{k}\{0, \ldots, v_i - 1\} &\rightarrow \mathfrak{P}(\mathbb{T}_{\mathbf{v},t}) \\ w &\mapsto \varphi_{\mathbf{v},t}(w) := \{\tau | w \text{ covers } \tau \in \mathbb{T}_{\mathbf{v},t}\}. \end{aligned}$$

Remark 4.9. Note that each function $\varphi_{\mathbf{v},t}$ is injective on its respective domain, as for any two vectors $w_1 \neq w_2 \in \prod_{i=1}^{k}\{0, \ldots, v_i - 1\}$ there exists at least one position p where they differ, and hence the $\mathbf{v}$-ary t-tuples that involve position p and that are covered by w_1 differ from those covered by w_2 involving position p.

We refer to Example 4.10 where $\varphi_{(3,2,2),2}$ is applied to all $(3, 2, 2)$-ary 2-tuples, for an illustrative example of this concept.
We are now able to formulate the translations of CA problems to SC problems. We will first elaborate on the connection between the OMCA problem and the MSC problem. As input for an OMCA problem we require the number of columns k, the alphabet sizes $(v_1, \ldots, v_k)$, and the desired strength $t \leq k$. In Algorithm 1 this input is translated to a universe U and a set of blocks $\mathcal{S}$. The pair $(U, \mathcal{S})$ can then serve as input to an MSC problem. In a second step, we show how such a minimal SC can subsequently be mapped to an optimal MCA.
Algorithm 1, which generalizes the algorithm given in [63], transforms MCA parameters $(t, k, (v_1, \ldots, v_k))$ – the input to the OMCA problem – to a SC problem instance having as input $(U, \mathcal{S})$ and can be informally described as follows. As universe U, we consider the set $\mathbb{T}_{\mathbf{v},t}$ of all $(v_1, \ldots, v_k)$-ary t-tuples, which have to be covered by the rows of the MCA to be constructed. Via the function $\varphi_{\mathbf{v},t}$, every vector w over $(v_1, \ldots, v_k)$, which could appear as a row of the MCA, is interpreted

as a block consisting of exactly those $(v_1, \ldots, v_k)$-ary t-tuples that are covered by w. Thus all blocks, generated this way, constitute the set of blocks $\mathcal{S}$.

Algorithm 1 MCAP2SCP

```
1: INPUT: $t, k, \mathbf{v} = (v_1, \ldots, v_k)$
Require: $0 \leq t \leq k$
2: $U \leftarrow \mathbb{T}_{\mathbf{v},t}$ (the set of all $\mathbf{v}$-ary $t$-tuples)    ▷ Compute the universe
3: $\mathcal{S} \leftarrow \emptyset$
4: for $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$ do    ▷ Compute set of blocks
5:     $\mathcal{S} \leftarrow \mathcal{S} \cup \varphi_{\mathbf{v},t}(w)$
6: end for
7: return $U, \mathcal{S}$
```

Algorithm 2 SC2MCA

```
1: INPUT: $\mathcal{C}$    ▷ Set Cover
Require: $\exists\, t, (v_1, \ldots, v_k)$ s.t. $\forall B \in \mathcal{C} : B \in Im(\varphi_{\mathbf{v},t})$
2: $A \leftarrow \emptyset,\ w \leftarrow 0$
3: for all $B \in \mathcal{C}$ do
4:     $w \leftarrow \varphi^{-1}_{t,\mathbf{v}}(B)$
5:     $A \leftarrow (A; w)$    ▷ Append row $w$ to $A$
6: end for
7: $MCA \leftarrow A$    ▷ $MCA$ is a mixed-level covering array
8: return $MCA$
```

Now let us assume we are given a minimal set cover $\mathcal{C}$ of size N of the output $(U, \mathcal{S})$ of MCAP2SCP (Algorithm 1). This set cover can be mapped back to an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$, by applying $\varphi^{-1}_{\mathbf{v},t}$ to the blocks in $\mathcal{C} \subseteq Im(\varphi_{\mathbf{v},t})$. The rows obtained, can be arranged in any order, forming an array consisting of all rows that correspond to a block in $\mathcal{C}$. Note that for each block $B \in \mathcal{S}$, where $\mathcal{S}$ is part of the output of MCAP2SCP, there is a unique corresponding row $w = (w_1, \ldots, w_k)$ which covers exactly those $\mathbf{v}$-ary t-tuples that are elements of B (see Remark 4.9). As the family $\mathcal{C}$ is a set cover of $\mathbb{T}_{\mathbf{v},t}$ by assumption, and a block of $\mathcal{C}$ covers a $\mathbf{v}$-ary t-tuple if and only if the corresponding row covers the $\mathbf{v}$-ary t-tuple, it follows that the constructed array indeed constitutes an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$. See SC2MCA (Algorithm 2) for the inverse translation. Finally, the minimality (with respect to cardinality)

of the computed sub-family $\mathcal{C}$ guarantees that the corresponding MCA is optimal, i.e. the corresponding MCA has $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k))$ rows: Assume there was an MCA having less rows. Then we could map these rows to their corresponding blocks, which then constitute a set cover if U of smaller cardinality – a contradiction to the minimality of the cardinality of $\mathcal{C}$.
Note that the size of the universe $|U| = \binom{k}{t} v^t$ is exponential in t and the number of blocks $|\mathcal{S}| = \prod_{i=1}^{k} v_1 \geq 2^k$ is exponential in k, which may make the search for optimal solutions infeasible in practice. Instead of an MSC problem solver, approximations as described in [12, 36] can be applied to the SC problem to obtain upper bounds for $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k))$. We will discuss such approaches in Section 4.3.
We demonstrate this connection between optimal CAs and minimal SCs with the following example.

Example 4.10. We map the problem of computing an optimal $\mathsf{MCA}(N; 2, 3, (3, 2, 2))$ to a minimal set cover problem. The universe U consists of all $(3, 2, 2)$-ary 2-tuples. For demonstration purposes we use the, previously introduced, informal vector notation for $(3, 2, 2)$-ary 2-tuples, e.g. we denote $((2, 1), (1, 3))$ as $(2, -, 1)$ where "$-$" represents an undefined entry. In particular, using this notation we obtain the universe

$$\begin{aligned} U \;=\; \{&(0,0,-),(0,1,-),(1,0,-),(1,1,-),(2,0,-),(2,1,-),\\ &(0,-,0),(0,-,1),(1,-,0),(1,-,1),(2,-,0),(2,-,1),\\ &(-,0,0),(-,0,1),(-,1,0),(-,1,1)\}\end{aligned}$$

Each vector over $(3, 2, 2)$, that could appear as a row of an $\mathsf{MCA}(N; 2, 3, 2)$, is mapped via $\varphi_{(3,2,2),2}$ to the set of $(3, 2, 2)$-ary 2-tuple it covers (see Figure 4.1). Thus we obtain the set of blocks $\mathcal{S}$, given in Figure 4.2, which contains the minimal set cover $\mathcal{C}$. By applying $\varphi^{-1}_{(3,2,2),2}$ to these sets of $(3, 2, 2)$-ary 2-tuples, $\mathcal{C}$ corresponds to the covering array A.

$$\begin{array}{lcl@{\qquad}lcl}
(0,0,0) & \leftrightarrow & \{(0,0,-),(0,-,0),(-,0,0,)\}, & (1,1,0) & \leftrightarrow & \{(1,1,-),(1,-,0),(-,1,0,)\}, \\
(0,0,1) & \leftrightarrow & \{(0,0,-),(0,-,1),(-,0,1,)\}, & (1,1,1) & \leftrightarrow & \{(1,1,-),(1,-,1),(-,1,1,)\}. \\
(0,1,0) & \leftrightarrow & \{(0,1,-),(0,-,0),(-,1,0,)\}, & (2,0,0) & \leftrightarrow & \{(2,0,-),(2,-,0),(-,0,0,)\}. \\
(0,1,1) & \leftrightarrow & \{(0,1,-),(0,-,1),(-,1,1,)\}, & (2,0,1) & \leftrightarrow & \{(2,0,-),(2,-,1),(-,0,1,)\}. \\
(1,0,0) & \leftrightarrow & \{(1,0,-),(1,-,0),(-,0,0,)\}, & (2,1,0) & \leftrightarrow & \{(2,1,-),(2,-,0),(-,1,0,)\}. \\
(1,0,1) & \leftrightarrow & \{(1,0,-),(1,-,1),(-,0,1,)\}, & (2,1,1) & \leftrightarrow & \{(2,1,-),(2,-,1),(-,1,1,)\}
\end{array}$$

Figure 4.1.: All vectors over $(3, 2, 2)$ mapped to their corresponding sets of $(3, 2, 2)$-ary 2-tuples, according to $\varphi_{(3,2,2),2}$.

$$\begin{aligned}
\mathcal{S} \;=\; & \{\{(0,0,-),(0,-,0),(-,0,0,)\},\{(0,0,-),(0,-,1),(-,0,1,)\}, \\
& \{(0,1,-),(0,-,0),(-,1,0,)\},\{(0,1,-),(0,-,1),(-,1,1,)\}, \\
& \{(1,0,-),(1,-,0),(-,0,0,)\},\{(1,0,-),(1,-,1),(-,0,1,)\}, \\
& \{(1,1,-),(1,-,0),(-,1,0,)\},\{(1,1,-),(1,-,1),(-,1,1,)\}, \\
& \{(2,0,-),(2,-,0),(-,0,0,)\},\{(2,0,-),(2,-,1),(-,0,1,)\}, \\
& \{(2,1,-),(2,-,0),(-,1,0,)\},\{(2,1,-),(2,-,1),(-,1,1,)\}\}.
\end{aligned}
\qquad
A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 0 \\ 2 & 1 & 1 \end{pmatrix}$$

$$\begin{aligned}
\mathcal{C} \;=\; & \{\{(0,0,-),(0,-,1),(-,0,1)\},\{(0,1,-),(0,-,0),(-,1,0)\}, \\
& \{(1,0,-),(1,-,0),(-,0,0)\},\{(1,1,-),(1,-,1),(-,1,1)\}, \\
& \{(2,0,-),(2,-,0),(-,0,0)\},\{(2,1,-),(2,-,1),(-,1,1)\}\}
\end{aligned}$$

Figure 4.2.: Set of blocks $\mathcal{S}$, minimal set cover $\mathcal{C}$ of the universe U and the corresponding optimal MCA A, referring to Example 4.10.

Example 4.11. To visualize the interplay between CAs and set covers we give another example. Consider the $\mathsf{CA}(5; 2, 4, 2)$ given on the left hand side of Figure 4.3, and the matrix $E = (e_{i,j})$ on the right hand side of Figure 4.3. Each position $e_{i,j}$ of the matrix E corresponds to exactly one $(2, 2, 2, 2)$-ary 2-tuple $(i, j) = ((x_1, x_2), (p_1, p_2))$, where the row index i equals the tuple (x_1, x_2) of values, and the column index $j = (p_1, p_2)$ represents the positions. The coloring of the entry in position $e_{i,j}$ shows which row of the CA covers the respective $(4, 2)$-tuple, such that the array on the right hand side represents the universe, with the cells as its elements. Cells of the same color are covered by the same block. Each block of the cover has again its own color and corresponds to the row of the same color of the CA on the left hand side of Figure 4.3.

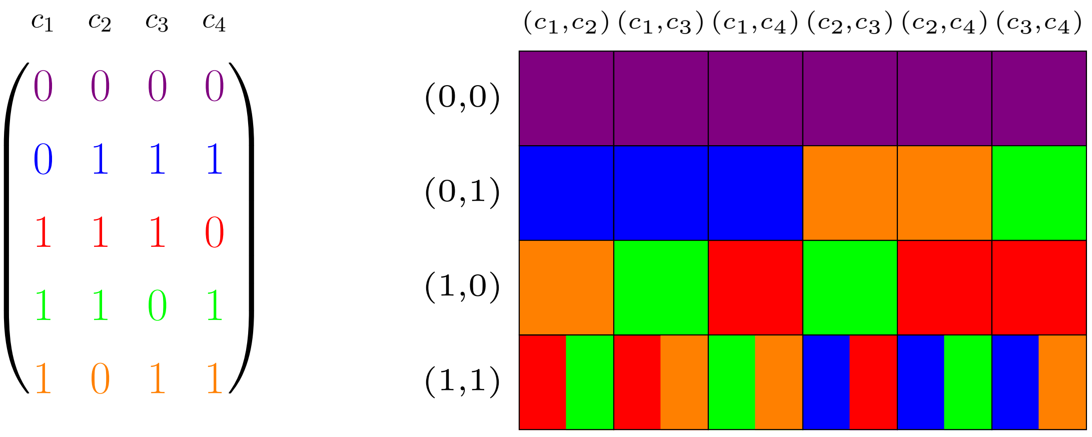


Figure 4.3.: Visualization of a CA as a SC. A colored row of the CA corresponds to a block, which covers those elements (tuples), whose entry in the matrix on the right hand side are of the same color.

Remark 4.12. Considering the general case, the universe and the set of blocks have a special structure, which depends on the MCA parameters they are generated from: $|U| = \sum_{\substack{I\subseteq\{1,\dots,k\}\\|I|=t}} \prod_{j\in I} v_j$ and $|\mathcal{S}| = \prod_{i=1}^{k} v_i$. From the fact that each $w \in \prod_{i=1}^{k}\{0,\dots,v_i-1\}$ covers exactly $\binom{k}{t}$ $(v_1,\dots,v_k)$-ary t-tuples, it follows that each block $B \in \mathcal{S}$ (which is part of the output of MCAP2SCP) has the same cardinality.

It is therefore possible to formulate each OMCA generation problem as an instance of the following specialized SC problem.

Problem 4.13 (Minimal d-Set Cover)**.** Given a universe U and a set of blocks $\mathcal{S}$, where each block has cardinality d, find a subset $\mathcal{C}$ of $\mathcal{S}$, of minimal cardinality, such that $\mathcal{C}$ is a d-set cover of U.

As will become apparent in the sequel, most problems considered in this work also translate to d-set covers for an appropriate d.

Remark. The problem of finding an OA of index unity, interpreted as a $\mathsf{CA}(v^t;t,k,v)$, can be transformed to an instance of an exact d-set cover via MCAP2SCP.

Having defined the terminology and enhanced by new terms, it is simple to review the IP formulations of the problems given in [88] and [96], which coincide with the OCA problem and OMCA problem, respectively. In fact we can identify them with the $0-1$ IP problem that is corresponding (c.f. Remark 4.6) to the output of MCAP2SCP. Following the notation of [88], Applegate's integer programming formulation for finding a $\mathsf{CA}(N;t,n,q)$, with a minimal number of rows, is given as follows. Let V denote the set of all q^n vectors that could appear as a row in the array to be constructed, S denote the set of all $\binom{n}{t}$ t-tuples of coordinates, and P be the set of all q^t q-ary vectors of length t. A covering array $D \subseteq V$ of strength t can then be specified by setting $x_v = 1$ if $v \in D$ and $x_v = 0$ if $v \notin D$ for al $v \in V$. Then, the problem of finding an optimal CA is equivalent to the following $0-1$ integer program: Choose $x_v \in \{0,1\}$ for $v \in V$, minimizing $k = \sum_{v \in V} x_v$, subject to the constraints

$$\sum x_v \geq 1, \text{ for all } s \in S,\ p \in P, \tag{4.4}$$

where the sum in (4.4) is over all $v \in V$ such that the projection of v onto s is p.
Let $(U,\mathcal{S})$ be the output of MCAP2SCP$(t,n,\mathbf{v}=(q,\ldots,q))$, with $\mathcal{S} = \{S_1,\ldots,S_{\binom{n}{t}}\}$, then Applegate's IP formulation coincides with the $0-1$ IP problem corresponding to $(U,\mathcal{S})$ (recall Remark 4.6),

$$\min\{x \cdot \mathbb{1} | Ax \geq \mathbb{1}, x \in \{0,1\}^{q^n \times 1}\}, \tag{4.5}$$

where $A = (ind(S_1),\ldots,ind(S_{\binom{n}{t}}))$ is the matrix of column-wise indicator vectors of the blocks in $\mathcal{S}$.
Similar to Applegates formulation of the OCA problem, the $0-1$ IP problem formulation of the OMCA problem, as given in [96], can be interpreted by translating the output of MCAP2SCP, to its corresponding $0-1$ IP problem.

4.2.2. Mapping Variable Strength CAs to SCs

In this subsection we first introduce a further generalization of CAs, not only considering different alphabet sizes per column as for MCAs, but also considering a generalized concept of *coverage*, leading to the notion of *variable strength covering arrays* (VCAs). Next, we introduce new mappings for VCAs to set covers that can be viewed as a natural extension of the mappings in subsection 4.2.1.

VCAs have been introduced in literature in non-uniform ways, see for example [17, 64, 80]. They can be regarded a generalization of MCAs, since not every t selection of columns, for a fixed t, has to fulfill specific cover properties. Instead the sets of columns, which have to satisfy the coverage properties, can be specified freely. A property that makes VCAs more adoptable to applications, e.g. in software testing [54], compared to CAs and MCAs. The following definition slightly deviated from the one given in [64], as we use a different nomenclature and consider mixed alphabet sizes.

Definition 4.14. For $\mathscr{I} \subseteq \mathfrak{P}(\{1,\dots,k\})$, a *variable strength covering array* (VCA) is an $N \times k$ array $(\mathbf{c}_1,\dots,\mathbf{c}_k)$, denoted as $\mathsf{VCA}(N;\mathscr{I},k,(v_1,\dots,v_k))$, with the following properties:

(i) $\forall j \in \{1,\dots,k\}$ the values in column the j-th column $\mathbf{c}_j$ arise from the set $\{0,\dots,v_j-1\}$.

(ii) $\forall \mathcal{I} \in \mathscr{I}$ the array comprised by the columns $(\mathbf{c}_i)_{i\in\mathcal{I}}$ has the property that every $|\mathcal{I}|$-tuple in $\prod_{i\in\mathcal{I}}[v_i]$ appears at least once as a row.

Remark 4.15. Using the same notation as in Definition 1.10, to avoid dealing with trivial cases and exceptions, throughout this work, we only consider VCAs where $2 \le v_i\ \forall i \in \{1,\dots,k\}$ and $\bigcup \mathscr{I} = \{1,\dots,k\}$. As $\mathscr{I}$ represents the set of all column selections, where all respective tuples need to be covered, the latter condition means that every column of a VCA appears in at least one such selection.

As was the case for MCAs, we define in a similar manner the following for VCAs.

Definition 4.16. The smallest N for which an $\mathsf{VCA}(N;\mathscr{I},k,(v_1,\dots,v_k))$ exists, is denoted as $\mathsf{VCAN}(\mathscr{I},k,(v_1,\dots,v_k))$ and is called the *variable strength covering array number* for $(\mathscr{I},k,(v_1,\dots,v_k))$. VCAs achieving this bound are called *optimal*.

Again, as for CAs and MCAs, VCAs with a small number of rows are of special interest.

Problem 4.17 (Optimal VCA (OVCA))**.** Given parameters k and $(v_1,\dots,v_k)$ and a set $\mathscr{I} \subseteq \mathfrak{P}(\{1,\dots,k\})$, find a $\mathsf{VCA}(N;\mathscr{I},k,(v_1,\dots,v_k))$ with $N = \mathsf{VCAN}(\mathscr{I},k,v_1,\dots,v_k)$.

To be able to translate problems related to VCAs to SC problems, we also need to generalize the notion of $\mathbf{v}$-ary t-tuples to a notation that allows tuples of different sizes.

Definition 4.18. For positive integers $v_1, \ldots, v_k$ and a set $\mathscr{I} \subseteq \mathfrak{P}(\{1, \ldots, k\})$ we define a $(v_1, \ldots, v_k)$*-ary* $\mathscr{I}$*-tuple* as a pair $((x_i)_{i \in \mathcal{I}}, \mathcal{I})$ with the property that $\mathcal{I} \in \mathscr{I}$ and $x_i \in \{0, \ldots, v_i - 1\}$, $\forall i \in \mathcal{I}$. For the sake of more compact writing we also use the notation $\mathbf{v}$-ary $\mathscr{I}$-tuple for a vector $\mathbf{v} = (v_1, \ldots, v_k)$.

Further notions are generalized below to apply also to VCAs.

Definition 4.19. For a family of positive integers $\mathbf{v} = (v_1, \ldots, v_k)$ and a set $\mathscr{I} \subseteq \mathfrak{P}(\{1, \ldots, k\})$ we say that a vector $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$ *covers* a $(v_1, \ldots, v_k)$-ary $\mathscr{I}$-tuple $((x_i)_{i \in \mathcal{I}}, \mathcal{I})$, if the entries of w in positions i equal x_i for all $i \in \mathcal{I}$. We say an array over $(v_1, \ldots, v_k)$ *covers* a $(v_1, \ldots, v_k)$-ary $\mathscr{I}$-tuple, if one of the rows of the array covers it. We further denote with $\varphi_{\mathbf{v},\mathscr{I}}$ the function that maps each $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$, to the set of $|\mathscr{I}|$ $(v_1, \ldots, v_k)$-ary $\mathscr{I}$-tuple that are covered by w. With $\mathbb{T}_{\mathbf{v},\mathscr{I}}$ we denote the set of all $(v_1, \ldots, v_k)$-ary $\mathscr{I}$-tuples.

$$\begin{aligned} \varphi_{\mathbf{v},\mathscr{I}} : \prod_{i=1}^{k} \{0, \ldots, v_i - 1\} &\rightarrow \mathfrak{P}(\mathbb{T}_{\mathbf{v},\mathscr{I}}) \\ w &\mapsto \varphi_{\mathbf{v},\mathscr{I}}(w) := \{\tau | w \text{ covers } \tau \in \mathbb{T}_{\mathbf{v},\mathscr{I}}\} \end{aligned}$$

Remark 4.20. Analogous to Remark 4.9 each function $\varphi_{\mathbf{v},\mathscr{I}}$ is injective on its respective domain. This can be seen, as any two vectors $w_1 \neq w_2$ over $\mathbf{v}$ differ in at least one coordinate j, and hence any $(v_1, \ldots, v_k)$-ary $\mathscr{I}$-tuple $((x_i)_{i \in \mathcal{I}}, \mathcal{I})$ with $j \in \mathcal{I}$ covered by w_1 can not be covered by w_2. $\bigcup \mathscr{I} = \{1, \ldots, k\}$ ensures the existence of at least one $\mathcal{I} \in \mathscr{I}$ with $j \in \mathcal{I}$.

Algorithm 3 transforms VCA parameters to an input to the SC problem and Algorithm 4 transforms a SC for the output of Algorithm 3 to a VCA.

Algorithm 3 VCAP2SCP

```
1: INPUT: $k, \mathbf{v} = (v_1, \ldots, v_k), \mathscr{I}$
2: $U \leftarrow \mathbb{T}_{\mathbf{v},\mathscr{I}}$ (set of all $\mathbf{v}$-ary $\mathscr{I}$-tuples)
3: $\mathcal{S} \leftarrow \emptyset$
4: for $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$ do        ▷ Compute set of blocks
5:     $\mathcal{S} \leftarrow \mathcal{S} \cup \varphi_{\mathbf{v},\mathscr{I}}(w)$
6: end for
7: return $U, \mathcal{S}$
```

Algorithm 4 SC2VCA

```
1: INPUT: $\mathcal{C}$                                          ▷ Set Cover
Require: $\exists \mathscr{I}, (v_1, \ldots, v_k)$ s.t. $\forall B \in \mathcal{C} : B \in Im(\varphi_{\mathbf{v},\mathscr{I}})$
2: $A \leftarrow \emptyset,\ w \leftarrow 0$
3: for all $B \in \mathcal{C}$ do
4:     $w \leftarrow \varphi^{-1}_{\mathbf{v},\mathscr{I}}(B)$
5:     $A \leftarrow (A; w)$                                  ▷ Append row $w$ to $A$
6: end for
7: $VCA \leftarrow A$                ▷ $VCA$ is a variable strength covering array
8: return $VCA$
```

Example 4.21. We map the problem of computing an optimal $\mathsf{VCA}(N; \mathscr{I}, 4, (2,2,2,2))$ for $\mathscr{I} = \{\{1,2,3\},\{1,4\},\{2,4\}\}$ to a minimal set cover problem. The universe U consists of all $(2,2,2,2)$-ary $\mathscr{I}$-tuples. For demonstration purposes we again use an informal vector notation for $(2,2,2,2)$-ary $\mathscr{I}$-tuples, e.g. we denote $((0,0,1),(1,2,3))$ as $(0,0,1,-)$ where "$-$" represents an undefined entry. Using this notation, we obtain the universe U:

$$\begin{aligned} U \;=\; & \{(0,0,0,-),(0,0,1,-),(0,1,0,-),(0,1,1,-),\\ & (1,0,0,-),(1,0,1,-),(1,1,0,-),(1,1,1,-),\\ & (0,-,-,0),(0,-,-,1),(1,-,-,0),(1,-,-,1),\\ & (-,0,-,0),(-,0,-,1),(-,1,-,0),(-,1,-,1)\}. \end{aligned}$$

Each $(2,2,2,2)$-ary $\mathscr{I}$-tuple that could appear as a row of a $\mathsf{VCA}(N; \mathscr{I}, 4, (2,2,2,2))$ is identified with the set of $(2,2,2,2)$-ary $\mathscr{I}$-tuples it covers:

$$\begin{aligned} (0,0,0,0) &\leftrightarrow \{(0,0,0,-),(0,-,-,0),(-,0,-,0)\},\\ (0,0,0,1) &\leftrightarrow \{(0,0,0,-),(0,-,-,1),(-,0,-,1)\},\\ &\vdots\\ (1,1,1,0) &\leftrightarrow \{(1,1,1,-),(1,-,-,0),(-,1,-,0)\},\\ (1,1,1,1) &\leftrightarrow \{(1,1,1,-),(1,-,-,1),(-,1,-,1)\}. \end{aligned}$$

We thus obtain the set of blocks $\mathcal{S}$ (see Figure 4.4), which contains the minimal set cover $\mathcal{C}$. Due to the identification of sets of $(2,2,2,2)$-ary $\mathscr{I}$-tuples with binary vectors of length four given above, $\mathcal{C}$ can be translated to the VCA array A, using SC2VCA (Algorithm 4).

$$\begin{aligned}
\mathcal{S} \;=\; & \{\{(0,0,-,-),(0,-,0,-),(0,-,-,0),(-,0,0,-),(-,0,-,0),(-,-,0,0)\},\\
& \qquad\qquad\qquad\qquad\qquad\qquad \dots\\
& \{(1,1,-,-),(1,-,1,-),(1,-,-,0),(-,1,1,-),(-,1,-,0),(-,-,1,0)\},\\
& \{(1,1,-,-),(1,-,1,-),(1,-,-,1),(-,1,1,-),(-,1,-,1),(-,-,1,1)\}\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{C} \;=\; & \{\{(0,0,1,-),(0,-,-,1)(-,0,-,1)\},\\
& \{(0,1,0,-),(0,-,-,0),(-,1,-,0)\},\\
& \{(1,0,1,-),(1,-,-,0),,(-,0,-,0)\},\\
& \{(1,1,0,-),(1,-,-,1),(-,1,-,1)\},\\
& \{(0,0,0,-),(0,-,-,1),(-,0,-,1)\},\\
& \{(1,0,0,-),((1,-,-,0),(-,0,-,1)\},\\
& \{(0,1,1,-),(0,-,-,0),(-,1,-,0)\},\\
& \{(1,1,1,-),(1,-,-,1),(-,1,-,1)\}\}
\end{aligned}
\qquad
A = \begin{pmatrix}
0 & 0 & 1 & 1\\
0 & 1 & 0 & 0\\
1 & 0 & 1 & 0\\
1 & 1 & 0 & 1\\
0 & 0 & 0 & 1\\
1 & 0 & 0 & 1\\
0 & 1 & 1 & 0\\
1 & 1 & 1 & 1
\end{pmatrix}$$

Figure 4.4.: Set of blocks $\mathcal{S}$, a minimal set cover $\mathcal{C}$ of the universe U and the corresponding optimal VCA A, referring to Example 4.21

4.2.3. Mapping Weighted Budgeted CAs to Budgeted SCs

In [35] the *testing budget problem* is formulated as the problem of constructing a $B \times k$ array over $\mathbf{v} = (v_1, \dots, v_k)$ that covers the most $\mathbf{v}$-ary t-tuples amongst all $B \times k$ arrays over $\mathbf{v}$, for given values B (the budget), k, $\mathbf{v} = (v_1, \dots, v_k)$ and a strength t. This formulation reflects the need of practitioners, for test suites generated from smaller arrays with *good coverage* properties instead of such generated from larger arrays with *complete coverage* (CAs) of $\mathbf{v}$-ary t-tuples.
We reflect this problem by defining the corresponding combinatorial object.

Definition 4.22 (Budgeted Covering Array)**.** For a positive integer B, referred to as the *budget*, a *budgeted covering array* is a $B \times k$ array $\mathsf{BCA}(B; t, k, (v_1, \dots, v_k))$ with the properties:

(i) $\forall j \in \{1, \dots, k\}$ the entries of the j-th column arise from the set $\{0, \dots, v_j - 1\}$,

(ii) $\mathsf{BCA}(B; t, k, (v_1, \ldots, v_k))$ covers the maximum number of $(v_1, \ldots, v_k)$-ary t-tuples, i.e. there is no other $B \times k$ array that covers more $(v_1, \ldots, v_k)$-ary t-tuples than a $\mathsf{BCA}(B; t, k, (v_1, \ldots, v_k))$.

In some applications, it might be the case that certain $\mathbf{v}$-ary t-tuples are more important to be covered than others. This leads to the following definition generalizing the notion of budgeted covering arrays, by additionally assigning *weights* to $\mathbf{v}$-ary t-tuples.

Definition 4.23. For a positive integer B, called the *budget*, positive integers $v_1, \ldots, v_k$ and t as well as a *weight function* $\omega : \mathbb{T}_{\mathbf{v},t} \to \mathbb{Q}$, where $\mathbf{v} = (v_1, \ldots, v_k)$, assigning a *weight* to each $\mathbf{v}$-ary t-tuple [2], a *weighted budgeted covering array* $\mathsf{WBCA}(B; t, k, (v_1, \ldots, v_k))$ is an $N \times k$ array satisfying the following criteria:

(i) $N \leq B$,

(ii) $\forall j \in \{1, \ldots, k\}$ the entries of the j-th column arise from the set $\{0, \ldots, v_j - 1\}$,

(iii) $\mathsf{WBCA}(B; t, k, (v_1, \ldots, v_k))$ maximizes the sum of weights of the $(v_1, \ldots, v_k)$-ary t-tuples that are covered by its rows, where even if a certain tuple is covered by multiple rows, its weight contributes only once to this sum.

Related structures to WBCAs have been discussed in [35, 4, 79].

Remark 4.24. Note that we do not appropriately use the notation of *covering arrays*, as WBCAs are not CAs or MCAs in a strict sense, since they do not require to cover all $\mathbf{v}$-ary t-tuples. Nevertheless, WBCAs can be considered a generalization of MCAs, as the latter appear as a special case of WBCAs, when the given budget $B \geq \prod_{i=1}^{k} v_i$ and all appearing weights are greater or equal to zero.

A definition similar to Definition 4.23 can be found in [4], where the authors introduce the notion of *ℓ-biased covering arrays*. Let us briefly review this work and relate it with the concept of WBCAs. Using the terminology introduced in this thesis, the authors of [4] consider arrays with k columns $f_1, \ldots, f_k$ over an alphabet of size v, additionally assuming that for all columns column f_i ($i \in \{1, \ldots, k\}$) and for each of its values $j \in \{0, \ldots, v-1\}$ u_j a numerical value $t_{i,u_j} \in [0; 1]$ is given. t_{i,u_j} reflects the importance of the assignment of the value u_j in column f_i of a row. In order to capture the importance of pairs, they define the importance of choosing

[2] In some cases we also consider given a set of *weights* $\mathcal{W} := \omega(\mathbb{T}_{\mathbf{v},t})$.

u_i in column f_i together with u_j in column f_j as $t_{i,u_i} \cdot t_{j,u_j}$. Then the *benefit* of a row (in isolation) is defined as the sum of all importances of pairs covered by this row. Put in the context of Definition 4.23, they assign the weight $t_{i,u_i} \cdot t_{j,u_j}$ to the $(v)_{i=1}^k$-ary 2-tuple $((u_i, u_j), (f_i, f_j))$, and the benefit of a row in isolation equals the sum of the weights of the tuples covered by the row. The *incremental benefit* of a row, with respect to a given array, is defined as the sum of all importances of pairs that are newly covered, i.e. covered by this row and not covered by any row of the given array. The *total benefit of an array* is then defined as the sum of the incremental benefits of its rows. In other words, the total benefit of an array is the sum of the benefits of all covered pairs, where the benefit of each pair contributes only once to the sum, even if covered more than once. Finally an *ℓ-biased covering array*, is defined in [4] as a $\mathsf{CA}(N; 2, k, v)$, where the benefit of the first ℓ rows is as large as possible, i.e. there exists no other $\mathsf{CA}(N'; 2, k, v)$ that has larger benefit on the first ℓ-rows than the first ℓ rows of an ℓ-biased covering array. This definition can be naturally generalized to CAs of higher strength as well as to MCAs.

Remark 4.25. The concepts of ℓ-biased covering arrays and weighted budgeted covering arrays are closely related. Since a $v^k \times k$ array, consisting of all v-ary vectors of length k in any order, is always a $\mathsf{CA}(v^k; 2, k, v)$, the first B rows of a B-biased covering array have to have maximal benefit under all possible sets of B rows, and therefore form a $\mathsf{WBCA}(B; 2, k, (v, \ldots, v))$ for budget B, where the weights of $(v)_{i=1}^k$-ary 2-tuples are assigned according to their benefits. Conversely, extending a $\mathsf{WBCA}(B; 2, k, (v, \ldots, v))$ for a given budget B and weights defined according to the importance of column-value assignments, the addition of rows until all v-ary 2-tuples are covered, always yields a B-biased covering array. This relation also holds when we consider the generalization of ℓ-biased covering arrays to higher strengths and MCAs.

This observation leads to the following problem, which builds upon these structures and naturally generalizes the testing budget problem as defined in [35].

Problem 4.26 (WBCA generation)**.** For a positive integer B (called the budget), a strength t, a k-tuple $\mathbf{v} = (v_1, \ldots, v_k)$, and a weight function $\omega : \mathbb{T}_{\mathbf{v},t} \to \mathbb{Q}$, assigning a weight to each $\mathbf{v}$-ary t-tuple, the *WBCA generation problem* is to construct a weighted budgeted covering array $\mathsf{WBCA}(B; t, k, (v_1, \ldots, v_k))$ for the budget B and the weight function ω.

Similarly to the OMCA Problem 1.16, the latter problem can be treated in terms of set covers. For this we need the following definition, which can be found among other works, in [51].

Problem 4.27 (Weighted Budgeted Set Cover (WBSC))**.** The *weighted budgeted set cover problem*[3] is defined as follows: Given a universe $U = \{u_1, u_2, ..., u_n\}$, with weights $(w_{u_i})_{i=1}^n$ associated to the elements, and a set of blocks $\mathcal{S} = \{S_1, S_2, ..., S_m\} \subseteq \mathfrak{P}(U)$, with costs $(c_{S_i})_{i=1}^m$ associated to the blocks, the goal is to find a collection of blocks $\mathcal{C} \subseteq \mathcal{S}$, such that the total cost of elements in $\mathcal{C}$ does not exceed a given budget B, and the total weight of elements covered by the blocks in $\mathcal{C}$ is maximized.

Algorithm 5 translates the input of a WBCA, with a given set of weights $\mathcal{W}$, problem to the input of a WBSC problem with unary costs.

Algorithm 5 WBCAP2WBSCP

```
1: INPUT: $t, k, \mathbf{v} = (v_1, \dots, v_k), \mathcal{W}$
Require: $t \leq k$
2: $U \leftarrow \mathbb{T}_{\mathbf{v},t}$ (the set of all $\mathbf{v}$-ary $t$-tuples)        ▷ Compute the universe
3: $\mathcal{S} \leftarrow \emptyset$
4: for $w \in \prod_{i=1}^k \{0, \dots, v_i - 1\}$ do        ▷ Compute set of blocks
5:     $\mathcal{S} \leftarrow \mathcal{S} \cup \varphi_{\mathbf{v},t}(w)$
6: end for
7: $(c_{S_i})_{i=1}^m = \mathbb{1}$        ▷ Assign unary costs
8: return $U, \mathcal{W}, \mathcal{S}, (c_{S_i})_{i=1}^m$
```

Example 4.28. Similarly to previous examples we map the problem of computing a $\mathsf{WBCA}(2; 2, 3, (3, 2, 2))$ for a given budget $B = 2$ and and weights $\mathcal{W}$ to a weighted budgeted set cover problem applying WBCAP2WBSCP (Algorithm 5). To improve readability, we omit the specification of the weights $\mathcal{W}$, instead we denote the weight of a $(3, 2, 2)$-ary 2-tuple as an exponent of its informal vector notation. The algorithm hence yields the following universe as part of its output:

$$\begin{aligned} U = \; & \{(0,0,-)^1, (0,1,-)^3, (1,0,-)^4, (1,1,-)^1, (2,0,-)^0, (2,1,-)^0, \\ & (0,-,0)^2, (0,-,1)^4, (1,-,0)^2, (1,-,1)^4, (2,-,0)^0, (2,-,1)^0, \\ & (-,0,0)^2, (-,0,1)^3, (-,1,0)^1, (-,1,1)^3\} \end{aligned}$$

[3] In literature (e.g. [51]), Problem 4.27 is known as the *Budgeted Maximum Set Cover problem.* We chose this different nomenclature to be consistent with the respective CA problems.

Each $(3,2,2)$-ary 3-tuple that could appear as a row of a $\mathsf{WBCA}(2;2,3,(3,2,2))$, is identified with the set of $(3,2,2)$-ary 2-tuple it covers. We denote the cumulative weight of the tuples covered by a row as its exponent:

$$\begin{array}{lll lll}
(0,0,0)^5 & \leftrightarrow & \{(0,0,-),(0,-,0),(-,0,0,)\}, & (1,1,0)^4 & \leftrightarrow & \{(1,1,-),(1,-,0),(-,1,0,)\},\\
(0,0,1)^8 & \leftrightarrow & \{(0,0,-),(0,-,1),(-,0,1,)\}, & (1,1,1)^8 & \leftrightarrow & \{(1,1,-),(1,-,1),(-,1,1,)\},\\
(0,1,0)^6 & \leftrightarrow & \{(0,1,-),(0,-,0),(-,1,0,)\}, & (2,0,0)^2 & \leftrightarrow & \{(2,0,-),(2,-,0),(-,0,0,)\},\\
(0,1,1)^{10} & \leftrightarrow & \{(0,1,-),(0,-,1),(-,1,1,)\}, & (2,0,1)^3 & \leftrightarrow & \{(2,0,-),(2,-,1),(-,0,1,)\},\\
(1,0,0)^8 & \leftrightarrow & \{(1,0,-),(1,-,0),(-,0,0,)\}, & (2,1,0)^1 & \leftrightarrow & \{(2,1,-),(2,-,0),(-,1,0,)\},\\
(1,0,1)^{11} & \leftrightarrow & \{(1,0,-),(1,-,1),(-,0,1,)\}, & (2,1,1)^3 & \leftrightarrow & \{(2,1,-),(2,-,1),(-,1,1,)\}
\end{array}$$

We thus obtain the set of blocks $\mathcal{S}$ (see Figure 4.5), which contains a solution $\mathcal{C}$ to the input $(U,\mathcal{S},\mathcal{W},\mathbb{1})$ to the weighted budgeted SC problem with unary costs. Due to the identification of sets of $(3,2,2)$-ary 2-tuples with $(3,2,2)$-ary 3-tuples given above, $\mathcal{C}$ can be translated to A, the WBCA to the right of Figure 4.5, using SC2MCA (Algorithm 2).

$$\begin{aligned}
\mathcal{S} \;=\; \{&\{(0,0,-),(0,-,0),(-,0,0,)\},\{(0,0,-),(0,-,1),(-,0,1,)\},\\
&\{(0,1,-),(0,-,0),(-,1,0,)\},\{(0,1,-),(0,-,1),(-,1,1,)\},\\
&\{(1,0,-),(1,-,0),(-,0,0,)\},\{(1,0,-),(1,-,1),(-,0,1,)\},\\
&\{(1,1,-),(1,-,0),(-,1,0,)\},\{(1,1,-),(1,-,1),(-,1,1,)\},\\
&\{(2,0,-),(2,-,0),(-,0,0,)\},\{(2,0,-),(2,-,1),(-,0,1,)\},\\
&\{(2,1,-),(2,-,0),(-,1,0,)\},\{(2,1,-),(2,-,1),(-,1,1,)\}\}
\end{aligned}
\qquad
A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$\mathcal{C} \;=\; \{\{(0,1,-),(0,-,1),(-,1,1,)\},\{(1,0,-),(1,-,1),(-,0,1,)\}\}$$

Figure 4.5.: Set of blocks $\mathcal{S}$, a weighted budgeted set cover $\mathcal{C}$ of the universe U and the corresponding WBCA A, referring to Example 4.28.

4.3. Algorithms

Having established the necessary concepts in the previous sections, we are now able to use the formulated problems and the respective mappings to compare sophisticated heuristic algorithms for CA generation from the view point of set covers. In particular we will review the greedy algorithm proposed in Section 3 of [13], the *deterministic density algorithm* DDA as proposed in [7], as well as an algorithm for the construction of *biased covering arrays*, presented in [5], for approximating some of the CA problems mentioned in the previous sections (e.g. Problems 1.16, 4.17 and 4.26). We will refer to the greedy algorithm of [13] with gAETG (Greedy AETG) to distinguish it from the "AETG" strategy proposed in the same work (i.e. in Section 4 of [13]). Moreover, we consider a generalized version of the gAETG algorithm for the case of MCAs.

4.3.1. gAETG: A Greedy Heuristic for MCA Generation

The authors of [13] mention a greedy algorithm which constructs a CA from an initially empty array by adding one row at a time. Although their algorithm is only formulated for CAs for strength $t = 2$, it can be generalized for higher strengths as well as for mixed alphabet sizes, i.e. for MCAs. In this thesis we generalize the greedy algorithm mentioned in [13] to work also for MCAs in Algorithm 6. Recall that $\varphi_{\mathbf{v},t}(r)$ is the set of all $\mathbf{v}$-ary t-tuples covered by the row r over $\mathbf{v} = (v_1, \dots, v_k)$. The algorithm starts with an initial empty array and proceeds in steps that append one row to the current array. Rows are chosen such that they cover the maximal number of $\mathbf{v}$-ary t-tuples that are not yet covered by the current array. Note that this algorithm uses a greedy method to maximize its profit in each step. By doing so, the algorithm also has characteristics of an exhaustive search, since, to the best of the author's knowledge, there is currently no known general method for finding the row r in step 5 of Algorithm 6, with $r = \arg\max_{r \in \prod_{i=1}^{k}[v_i]} |\varphi_{\mathbf{v},t}(r) \cap T|$, other than iterating through $\prod_{i=1}^{k}\{0, \dots, v_i - 1\} \setminus A$, which has a size exponential in k (as follows from Corollary 4.29). This is also the reason why the authors of [13] modify their algorithm towards a random greedy method for practical purposes; Instead of determining $r = \arg\max_{r \in \prod_{i=1}^{k}[v_i]} |\varphi_{\mathbf{v},t}(r) \cap T|$ in step 5, a set of *candidate* rows is computed, from which one that covers the most yet uncovered $\mathbf{v}$-ary t-tuples is selected to extend the current array. Each candidate row is generated by filling up its position in a random order, choosing a value that maximizes the number of $\mathbf{v}$-ary t-tuples covered by the partially specified row. See [13] for details.

Algorithm 6 GAETG (for MCAs)

```
1: INPUT: $t, k, \mathbf{v} = (v_1, \dots, v_k)$
Require: $t \leq k$
2: $A \leftarrow \emptyset$                                      ▷ Initial array is empty
3: $T \leftarrow \mathbb{T}_{\mathbf{v},t}$ (the set of all $\mathbf{v}$-ary $t$-tuples)    ▷ Initialize set of tuples
4: while $T \neq \emptyset$ do
5:     determine $r = \arg\max_{r \in \prod_{i=1}^{k}[v_i]} |\varphi_{\mathbf{v},t}(r) \cap T|$
6:     $A \leftarrow A \cup \{r\}$
7:     $T \leftarrow T \setminus \varphi_{\mathbf{v},t}(r)$
8: end while
9: return $A$
```

Although having a run time that is exponential in k when implemented in the described naive manner, the GAETG algorithm is important, since it is proven to produce an output that grows logarithmically in k in all cases. In other words, it constructively provides evidence that $\mathsf{MCAN}(t, k, \mathbf{v}) \in O(\log k)$.

Corollary 4.29. *For the input $t, k, (v_1, \dots, v_k)$ Algorithm 6 returns an* $\mathsf{MCA}(N; t, k, (v_1, \dots, v_k))$ *with $N \in O(\log k)$.*

Scetch of a Proof:

1.) From Theorem 1.27 we know that in each iteration of steps 4 - 8 of Algorithm 6, there exists a row that covers at least n/h $\mathbf{v}$-ary t-tuples that are not covered by the yet constructed array.

2.) Hence in step 5 a row is selected that covers at least n/h currently uncovered $\mathbf{v}$-ary t-tuples.

3.) The proof of Corollary 1.28 applies and shows that $N \in O(\log k)$. □

4.3.2. A Review of the Deterministic Density Algorithm

The authors of [7, 6] propose a *deterministic density algorithm* (DDA) as a sophisticated greedy heuristic approach to the optimal MCA generation problem that can be considered an improvement of gAETG. As mentioned in [6], the proof of the Logarithmic Guarantee (Corollary 1.28) and hence the proof of Corollary 4.29 relies on finding a row that covers at least n/h of the yet uncovered $\mathbf{v}$-ary t-tuples, and not necessarily on finding a row that covers the maximal number of uncovered $\mathbf{v}$-ary t-tuples. This observation was considered in the development of DDA, which, like gAETG, constructs an array one row at a time until all $\mathbf{v}$-ary t-tuples are covered. In contrast to gAETG, DDA aims to add a row that covers at least the average (and not necessarily maximal) number of still uncovered $\mathbf{v}$-ary t-tuples. The main improvement of DDA over gAETG is that there exists an efficient way to construct these rows, avoiding the search over the exponentially sized set of all candidate rows. The key notion that provides means to construct such a row is the notion of *density*. Informally, the density of an array is the proportion of uncovered $\mathbf{v}$-ary t-tuples to the number of all $\mathbf{v}$-ary t-tuples. Similarly, the density of a value x in a certain column p is the proportion of uncovered to total $\mathbf{v}$-ary t-tuples $((x_1, \dots, x_t), (p_1, \dots, p_t))$ containing this value-column assignment, i.e. $\exists i \in \{1, \dots, t\} : x_i = x \wedge p_i = p$. Additionally, the density of a partially specified row, as well as the factor density are introduced, enabling the algorithm to efficiently construct rows, one column at a time. The reader may be referred to [7] for more details. We give a high-level algorithmic description of DDA in Algorithm 7. Note that in [7, 6] additional layers of algorithmic designs as well as variations of Algorithm 7 are discussed, one of which, the 0-*restricted* version of DDA, is identical with the random greedy version of AETG, which is commercial available.

Algorithm 7 DDA

```
1: INPUT: $t, k, (v_1, \dots, v_k)$
2: $A \leftarrow \emptyset$
3: while There remain uncovered $\mathbf{v}$-ary $t$-tuples do
4:     $r \leftarrow ()$                                          $\triangleright$ initially empty row
5:     for i=1 to k do
6:         Set an empty position of $r$ to a value which density is above average
7:     end for
8:     append row $r$ to $A$
9: end while
10: return $A$
```

The main advantage of DDA (Algorithm 7) over gAETG (Algorithm 6) arises from the fact that the complexity of step 6 of Algorithm 7 is upper bounded by $v^t\binom{k-1}{t-1}$ up to a multiplicative constant, hence it runs in polynomial, instead of exponential time in k which the gAETG algorithm takes.
The properties of DDA for the case of homogenous alphabet sizes, i.e. $v = v_1 = \ldots = v_k$ is given in Theorem 2.2. of [7], which we restate next, and refer the reader to [7] for the proof.

Theorem 4.30. *For the input t and $v = v_1, \ldots, v_k$,* DDA *produces a* $\mathsf{CA}(N; t, k, v)$ *with $N \in O(\log k)$ in $O(k \log(k) v^t \binom{k-1}{t-1})$ time complexity, which is polynomial in k.*

We will provide an experimental comparison of gAETG and DDA in Section 4.4.

4.3.3. DDA for WBCAs

In [4] a modified version of the DDA algorithm is proposed in order to approximately generate an ℓ-biased array (see Section 4.2.3). The authors of [4] refer to such approximations as *biased covering arrays*, being covering arrays that offer a "large" total benefit in the first ℓ rows for every ℓ. Due to the similarity to the problem of approximating an optimal MCA, the necessary changes to DDA to produce biased covering arrays are minor. In effect, the algorithmic design is the same as that of DDA (Algorithm 7), with the main difference that the incremental benefit of $\mathbf{v}$-ary t-tuples is included in the computation of densities, yielding a notion of *weighted density*. The initial benefits are computed from a given set $\mathcal{T}$ of importances of of column-value assignments. Algorithm 8 gives a high-level description of this algorithm. For details see [78], where the notions of [4] are also generalized to higher strengths.
As already mentioned in Remark 4.25, the notions of ℓ-biased covering arrays and weighted budgeted covering arrays are very similar. One can therefore use Algorithm 8 as an sophisticated greedy heuristic algorithm for WBCAs by simply replacing the **while**-loop in step 3 with a **for**-loop that limits the construction of rows to the given budget B, see Algorithm 9.

4.3.4. Revisiting gAETG as a Greedy Heuristic for Set Covers

We are now going to use the established translations between CAs and SCs, to revisit gAETG from the point of set covers. This enables us to make use of a bound for

Algorithm 8 BIASEDDDA

```
1: INPUT: $t, k, (v_1, \ldots, v_k), \mathcal{T}$
2: $A \leftarrow \emptyset$
3: while There remain uncovered $\mathbf{v}$-ary $t$-tuples do
4:     $r \leftarrow ()$                                  ▷ Initialize empty row
5:     for i=1 to k do
6:         Set an empty position of $r$ to a value whose weighted density is above average
7:     end for
8:     append row $r$ to $A$
9: end while
10: return $A$
```

Algorithm 9 WBDDA

```
1: INPUT: $t, k, (v_1, \ldots, v_k), \mathcal{T}, B$
2: $A \leftarrow \emptyset$
3: for $j = 1$ to $B$ do
4:     $r \leftarrow ()$                                  ▷ Initialize empty row
5:     for $i = 1$ to $k$ do
6:         Specify an unspecified position of $r$ to a value whose weighted density is above average
7:     end for
8:     $A \leftarrow (A; r)$                              ▷ Append row $r$ to $A$
9: end for
10: return $A$
```

the size of the output of GAETG.

A known approach to approximate a solution for the SC problem is Algorithm 10. For a given universe U and a set of blocks $\mathcal{S}$, this algorithm iteratively selects a block of $\mathcal{S}$ that covers the most currently uncovered elements of the universe, and is greedy in that sense.

An analytical treatment of Algorithm 10 can be found in [45] and in a generalized version in [12], which contains a result regarding the quality of the algorithm's output, which we restate in the following theorem.

Theorem 4.31. *For any instance of an MSC problem $(U, \mathcal{S})$, Algorithm 10 returns a*

Algorithm 10 SC-GREEDY

```
1: INPUT: $U, \mathcal{S}$
2: $\mathcal{C} \leftarrow \emptyset$                                  ▷ Initial Cover Set empty
3: while $U \neq \emptyset$ do
4:     determine $b = \arg\max_{b \in \mathcal{S}} |b \cap U|$
5:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{b\}$
6:     $U \leftarrow U \setminus b$
7: end while
8: return $\mathcal{C}$
```

set cover $\mathcal{C}$ of U with $|\mathcal{C}| \leq H(d) \cdot |MSC|$, where d is the maximum of the cardinalities of the blocks in $\mathcal{S}$, $H(n)$ denotes the n-th harmonic number and $|MSC|$ the size of a minimal set cover of U with blocks from $\mathcal{S}$.

Proof:
To prove the assertion we follow the work in [12], considering the IP corresponding to a given MSC problem $(U, \mathcal{S})$, with $U = \{x_1, \ldots, x_m\}$ and $\mathcal{S} = \{b_1, b_2, \ldots, b_n\}$. So let $A = (ind(b_1), ind(b_2), \ldots, ind(b_n)) = (a_{i,j}) \in \{0,1\}^{m \times n}$ be the binary matrix of column-wise indicator vectors of the blocks of $\mathcal{S}$. Let $\mathcal{C}$ be the output of SC-GREEDY (Algorithm 10) applied to $(U, \mathcal{S})$, and $\mathcal{X} \subseteq \mathcal{S}$ an arbitrary set cover of U. We show that $|\mathcal{X}| \cdot H(d) \geq |\mathcal{C}|$ from which the assertion follows immediately for $\mathcal{X}$ being a minimal SC. Let $\mathbf{x} = (x_i)_{i=1}^n \in \{0,1\}^{n \times 1}$ denote the indicator vector of $\mathcal{X}$ and equivalently $\mathbf{c} = (c_i)_{i=1}^n \in \{0,1\}^{n \times 1}$ that of $\mathcal{C}$. Then we have $A \cdot \mathbf{x} \geq \mathbb{1}$ and $A \cdot \mathbf{c} \geq \mathbb{1}$, or equivalentely

$$\sum_{j=1}^{n} a_{i,j} x_j \geq 1 \ \forall i \in \{1, \ldots, m\}$$
$$x_j \in \{0,1\},$$

and the same for $(c_i)_{i=1}^n$. We will show that the following holds,

$$|\mathcal{C}| = \sum_{j=1}^{n} c_j \leq \sum_{j=1}^{n} H\Big(\sum_{i=1}^{m} a_{i,j}\Big) x_j, \tag{4.6}$$

and as

$$\sum_{j=1}^{n} H\Big(\sum_{i=1}^{m} a_{i,j}\Big) x_j = \sum_{j=1}^{n} H(|S_j|) x_j \leq \sum_{j=1}^{n} H(d) x_j = H(d)|\mathcal{X}|,$$

the assertion follows.
To prove (4.6) , it will suffice to find non-negative numbers $y_1, \dots, y_m$, with

$$H\Big(\sum_{i=1}^{m} a_{i,j}\Big) \geq \sum_{i=1}^{m} a_{i,j} y_i \tag{4.7}$$

and

$$|\mathcal{C}| = \sum_{j=1}^{m} c_j = \sum_{i=1}^{m} y_i. \tag{4.8}$$

As from this we get the desired:

$$\begin{aligned} \sum_{j=1}^{n} H\Big(\sum_{i=1}^{m} a_{i,j}\Big) x_j \quad &\geq \quad \sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{i,j} y_i\Big) x_j = \sum_{i=1}^{m} \underbrace{\Big(\sum_{j=1}^{n} a_{i,j} x_j\Big)}_{\geq 1} y_i \\ &\geq \quad \sum_{i=1}^{m} y_i = |\mathcal{C}|. \end{aligned}$$

The numbers $y_1, \dots, y_m$, satisfying equations (4.7) and (4.8), have an intuitive interpretation, where y_i is the price paid by the greedy heuristic for covering the point i. To be more precise, let $U^{(r)}$ denote the universe at the beginning of the r-th iteration of the algorithm, and $b_i^{(r)} := b_i \cap U^{(r)}$ the remaining uncovered elements in block b_i at the beginning of the r-th iteration, for all $i = 1, \dots, m$. Further let $w_i^r := |b_i^{(r)}|$ denote the number of these elements. Without loss of generality we may assume that $\mathcal{C} = \{b_1, \dots, b_r\}$ after r iterations, as renumbering the blocks does not have any effect on the cardinality of a set cover. Assuming that there are ℓ iterations as a whole, i.e. Algorithm 10 returns $\mathcal{C} = \{b_1, \dots, b_\ell\}$ we have that each $x_i \in U$ belongs to exactly one of the sets $b_1^{(1)}, b_2^{(2)}, \dots, b_\ell^{(\ell)}$ as they are a partition of U. Now we define for all $i \in \{1, \dots, m\}$:

$$y_i := \frac{1}{w_r^r} \Leftrightarrow x_i \in b_r^{(r)}.$$

Informally speaking we define the price of an element lower, if it is covered with many other elements by the same block. With this definition of $y_1, \dots, y_m$, we immediately can establish equation (4.8), as

$$\sum_{i=1}^{m} y_i = \sum_{r=1}^{\ell} \sum_{x_i \in b_r^{(r)}} y_i = \sum_{r=1}^{\ell} w_r^r \frac{1}{w_r^r} = \ell = |\mathcal{C}|.$$

To prove (4.7) we note that $b_j \cap b_r^{(r)} = b_j^{(r)} \setminus b_j^{(r+1)}$, from which we get

$$\begin{aligned}\sum_{i=1}^{m} a_{i,j} y_i &= \sum_{i=1}^{\ell} \sum_{x_i \in b_j \cap b_r^r} y_i \\ &= \sum_{r=1}^{\ell} (w_j^r - w_j^{r+1}) \frac{1}{w_r^r} \\ &= \sum_{r=1}^{s} (w_j^r - w_j^{r+1}) \frac{1}{w_r^r},\end{aligned}$$

where s is the largest superscript such that $w_j^s > 0$. From this, as $w_j^r \leq w_j^{r+1}$, we get

$$\begin{aligned}\sum_{r=1}^{s} (w_j^r - w_j^{r+1}) \frac{1}{w_r^r} &\leq \frac{1}{w_j^{r+1}+1} + \frac{1}{w_j^{r+1}+2} + \ldots + \frac{1}{w_j^r} \\ &= \sum_{i=1}^{s} (H(w_j^r) - H(w_j^{r+1})) = H(w_j^1),\end{aligned}$$

where

$$w_j^1 = |b_j| = \sum_{i=1}^{m} a_{i,j} \leq d = \max\{|b_j| : j = 1, \ldots, n\},$$

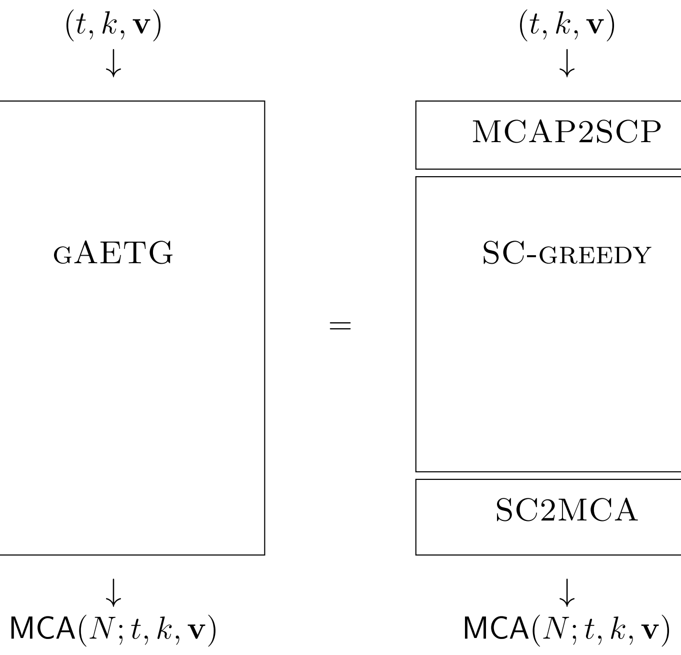
and hence the assertion holds. □

Being aware of the translation of the OMCA problem to the MSC problem (Section 4.2), it becomes apparent that the gAETG (Algorithm 6) is identical to the SC-greedy Algorithm 10 applied to the outcome of the translation MCAP2SCP (Algorithm 1) while transforming the output SC back to an MCA using SC2MCA (Algorithm 2). This connection is depicted in Figure 4.6. This implies that the number of rows of the array A returned by gAETG has the upper bound $H\left(\binom{k}{t}\right) \cdot \mathsf{MCAN}(t, k, (v_1, \ldots, v_k))$, which can be seen as a quality assurance for the output of gAETG.

A short introduction to the complexity notions used in the sequel can also be found in Chapter 11 of [49]. As complexity theory is not part of the main subject of this thesis, we refer the interested reader to [49] and limit ourselves with the description of a polynomial time algorithm as an algorithm that runs in a number of steps that is polynomially bound in the size of the input to the algorithm.

Remark 4.32. In [28] in Theorem 4.4., it is shown that unless

$$NP \subseteq TIME(n^{O(\log_2 \log_2 n)}),$$



$(t, k, \mathbf{v})$
$(t, k, \mathbf{v})$
gAETG
=
MCAP2SCP
SC-greedy
SC2MCA
$\mathsf{MCA}(N; t, k, \mathbf{v})$
$\mathsf{MCA}(N; t, k, \mathbf{v})$

Figure 4.6.: Structural decomposition of gAETG via SC-greedy.

there is no polynomial time algorithm that, for arbitrary input $(U, \mathcal{S})$, outputs a set cover $\mathcal{C}$ with $|\mathcal{C}| \leq (1-\epsilon)|MSC| \log_2 n$ for any $\epsilon > 0$, where $|MSC|$ denotes the size of a minimal set cover.

In other words, SC-greedy (Algorithm 10) is likely to be an optimal poly-time approximation algorithm for the *minimal set cover* problem. At this point, it is important to point out that this does not by any means give evidence for the optimality of gAETG. Our mapping, MCAP2SCP (Algorithm 1), generates output that is exponential in k, constructing an exponentially large (in terms of the original input) instance to a more general problem. Additionally, the blocks and universe of SC instances generated via MCAP2SCP have a specific structure (see Remark 4.12), possibly making feasible the development of more efficient algorithms.
In fact, comparing DDA (Algorithm 7) to gAETG (the latter of which is equivalent to the SC-greedy, applied to instances that are output of the mapping MCAP2SCP), it becomes apparent that DDA makes use of the inner structure of the problem at hand: $\mathbf{v}$-ary t-tuples are treated as atomic elements in gAETG, while DDA treats $\mathbf{v}$-ary t-tuples as *molecules* composed of column-value assignments. As a result, DDA is an algorithm that constructs (M)CAs of size $O(\log k)$

in a number of steps that is bounded by a polynomial in k (see [7], Theorem 2.2).

4.3.5. A Weighted Budgeted Variant of gAETG for WBCAs

In this subsection, we propose a new weighted budgeted variant of gAETG, obtained via our mappings from a greedy algorithm, for the weighted budgeted set cover problem 4.27.

Just as Algorithm 8 generalizes Algorithm 7 to handle weighted budgeted problem instances, there exists a generalization of the SC-greedy (Algorithm 10) and hence, following the observations of the last subsection, of gAETG (Algorithm 6), that approximates the weighted budgeted set cover problem (Problem 4.27). We provide a brief description of the algorithm here, while for a more detailed discussion of the topic see for example [39].

The weighted budgeted SC problem has been studied in a number of previous scientific publications, e.g. [39, 51]. wbSC-greedy (Algorithm 11) takes as input a universe U, a set of blocks $\mathcal{S}$, a set of weights $\mathcal{W}$ with a weight for each element of the universe and a budget B. The algorithm starts with an empty set $\mathcal{C}$ and in each step adds a block with maximal weight to $\mathcal{C}$. After each addition of a block to $\mathcal{C}$, the weights of all blocks in $\mathcal{S}$ need to be updated, i.e. the weight of a block gets the sum of the weights of its elements that are uncovered by the current blocks in $\mathcal{C}$. In other words, elements already covered contribute zero to the weight of the blocks they reside in. This process is iterated, until the whole universe is covered.

There exists a lot of literature on the weighted budgeted set cover problem and variations thereof [39, 51]. An analysis of this problem for non-unary costs can be found in [51].

Algorithm 11 wbSC-greedy

```
1: INPUT: $U, \mathcal{S}, \mathcal{W}, B$
2: $\mathcal{C} \leftarrow \emptyset$                              ▷ Initial Cover empty
3: for $i = 1$ to $B$ do
4:     $b = \arg\max_{b \in \mathcal{S}} w_b$                ▷ Select block with max. weight
5:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{b\}$
6:     $\mathcal{W} \leftarrow \mathcal{W}'$                          ▷ Update all weights
7: end for
```

Similar to the SC-greedy algorithm, there exists a quality assurance for the output

$\mathcal{C}$ of wbSC-greedy (Algorithm 11), which we restate next. See [39], Corollary 1 for a proof.

Theorem 4.33. *Let $U, \mathcal{S}, \mathcal{W}, B$ be an input to the weighted budgeted set cover problem. Then it holds that*

$$\frac{\omega(\mathcal{C})}{\omega(OPT)} > 1 - \frac{1}{e},$$

where $\omega(\mathcal{C})$ denotes the cumulative weight of the elements covered by the blocks in the output $\mathcal{C}$ of Algorithm 11, and $\omega(OPT)$ is the maximal cumulative weight of elements a subset of cardinality B of $\mathcal{S}$ can cover.

As Sc-greedy (Algorithm 10), also wbSC-greedy is optimal in some sense, as the following theorem, which as has been proven in [28], shows.

Theorem 4.34. *For any $\epsilon > 0$, the weighted budgeted set cover problem cannot be approximated in polynomial time within a ratio of $(1 - 1/e + \epsilon)$ unless $P = NP$.*

Using the notation of Theorem 4.33, this means that (provided $P \neq NP$) there is no polynomial time algorithm for the budgeted maximal set cover problem that produces an output $\mathcal{C}$ with $\frac{\omega(\mathcal{C})}{\omega(OPT)} > 1 - \frac{1}{e} + \epsilon$ for an arbitrary $\epsilon > 0$.
In light of Subsection 4.3.4, it is natural to describe a weighted budgeted version of gAETG (Algorithm 6) as a composition of wbCAP2wbSCP (Algorithm 5), wbSC-greedy and SC2MCA (Algorithm 2), see Figure 4.7 for the structure. We introduce wbgAETG (Algorithm 12) as follows.

Algorithm 12 wbgAETG

```
1: INPUT: $t, k, \mathbf{v} = (v_1, \dots, v_k), B, \mathcal{W}$
Require: $t \leq k$
2: $A \leftarrow \emptyset$                                         ▷ Initial array empty
3: $T \leftarrow \mathbb{T}_{\mathbf{v},t}$ (the set of all $\mathbf{v}$-ary $t$-tuples)
4: for $i = 1$ to $B$ do
5:     determine row $r$ that covers the greatest weight of $\mathbf{v}$-ary $t$-tuples in $T$
6:     $A \leftarrow (A; r)$                                   ▷ Append row $r$ to $A$
7:     $T \leftarrow T \setminus \varphi_{\mathbf{v},t}(r)$
8: end for
9: return $A$
```

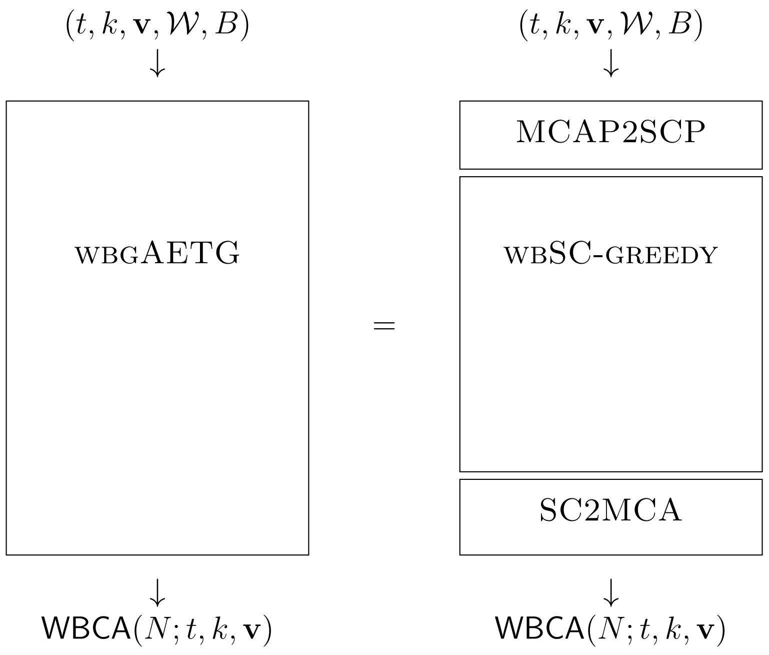

$(t, k, \mathbf{v}, \mathcal{W}, B)$
$(t, k, \mathbf{v}, \mathcal{W}, B)$
wbgAETG
=
MCAP2SCP
wbSC-greedy
SC2MCA
$\mathsf{WBCA}(N; t, k, \mathbf{v})$
$\mathsf{WBCA}(N; t, k, \mathbf{v})$

Figure 4.7.: Structure of wbgAETG via SC-greedy.

Due to the structure of wbgAETG, the quality assurance for wbSC-greedy also holds for wbgAETG, as the latter treats a subset of the instances to the former and uses the same methodology.

Theorem 4.35. *Let $t, k, \mathbf{v} = (v_1, \dots, v_k), B, \mathcal{W}$ be an input to the WBCA generation problem 4.26. Then it holds that*

$$\frac{\omega(A)}{\omega(OPT)} > 1 - \frac{1}{e}$$

where $\omega(A)$ denotes the cumulative weight of $\mathbf{v}$-ary t-tuples covered by the output array A of Algorithm 12, and $\omega(OPT)$ the maximal cumulative weight of $\mathbf{v}$-ary t-tuples an array over $(v_1, \dots, v_k)$ with B rows, i.e. a $\mathsf{WBCA}(B; t, k, (v_1, \dots, v_k))$, can cover.

Analogue to the unweighted case, Theorem 4.34 does not apply to WBCAs, as WBSC instances that are output of wbCA2wbSC (Algorithm 5), form a specialized subset of all WBSC instances. Hence, we conclude that even if $P \neq NP$, Theorem 4.34 does not provide evidence for the optimality of wbgAETG.

4.4. Experiments

In this section we evaluate the performance of the algorithms discussed in previous sections. As detailed in this chapter, CA problems can be viewed as special cases of SC problems. We aim to give empirical evidence that CA solvers (which can be considered as approximation algorithms for the OMCA Problem 1.16) benefit from the additional structure of these problems. Additionally, we evaluate the theoretical bounds (referred to as *upper bounds* throughout the remainder of this section) obtained through Theorem 4.31 and Corollary 1.28 (equation (1.6)) for CAs using practical experiments. To this end, we run the algorithms against problem instances with known covering array numbers (CANs) and compare the size of the generated CAs to this lower bound represented by the CANs, to obtain an experimental measure for the quality of the greedy heuristic employed in GAETG (Algorithm 6) as well as the more sophisticated heuristic used in DDA (Algorithm 7).

We compare implementations of GAETG (resp. WBGAETG, Algorithm 12) against DDA (and WBDDA, Algorithm 9), developed in the scope of the work presented in [47], in terms of runtime and output size. To put it differently, we compare heuristic algorithms for CA problems against their counterparts for SC problems applied to the same problem instances.

The algorithms DDA[4], WBDDA, GAETG and WBGAETG were implemented in Rust ([82]), a modern systems programming language. In all implementations, columns are sorted in descending order of their alphabet sizes, i.e. for an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$, $v_i \geq v_j \Leftrightarrow i \geq j$. In steps 6 of DDA respectively 6 of WBDDA, values are always assigned from left to right of the partially constructed rows, as the order of assignments does not affect the logarithmic guarantee (Theorem 1 in [7]).

For DDA and WBDDA, there are two variants implemented: In the variant referred to as DDA_{avg}, the first value whose density is above average is selected, whereas the variant DDA_{max} chooses a value with maximal density (if several values achieve maximal density, the last one is selected). Analogue is the case for WBDDA_{avg} and WBDDA_{max} for weighted densities.

The implementation of GAETG uses a deterministically generated list of all rows that might appear in the CA to be constructed. Recall that in each step, a row that covers the maximal number of yet uncovered $(v_1, \ldots, v_k)$-ary t-tuples is selected. To

[4]Note that we exclusively implement the *unrestricted* variant of DDA, referring to [7], where also a discussion of the effects of *restricted* density calculations is given.

enable the algorithm to find arrays with fewer rows, the starting point in the list of potential rows is randomized and the algorithm is executed 10 times.
The following tables list the best and worst obtained result under the headings gAETG$_{best}$ respectively gAETG$_{worst}$. The experiments were performed on a machine with an Intel Core i7-4770 CPU clocked at 3.40GHz with 24GB of RAM. Note that memory usage is not listed below, as the difference between implementations was negligible in all cases, due to the fact that all of them employ the same underlying structure: A continuous chunk of memory containing $v^t\binom{k}{t}$ 32-bit floating point numbers indicating the benefit of covering a specific tuple (0 if the tuple is covered).

Instance				**# Rows**				**Time (s)**			
(t, k, v)	CAN	Bound$_L$	Bound$_H$	DDA		gAETG		DDA		gAETG	
				avg	*max*	*best*	*worst*	*avg*	max	*best*	*worst*
(2, 3, 2)	4	9	7	4	4	4	5	0.00003	0.00003	0.00002	0.00002
(2, 4, 2)	5	12	12	6	6	6	6	0.00003	0.00003	0.00003	0.00003
(2, 5, 2)	6	13	17	6	6	6	6	0.00003	0.00003	0.00004	0.00004
(2, 6, 2)	6	15	19	7	7	7	7	0.00004	0.00004	0.00008	0.00011
(2, 7, 2)	6	16	21	7	7	7	7	0.00005	0.00005	0.00017	0.00018
(2, 8, 2)	6	17	23	8	8	8	8	0.00006	0.00006	0.00045	0.00051
(2, 9, 2)	6	18	25	8	8	8	8	0.00008	0.00008	0.00103	0.00108
(2, 10, 2)	6	19	26	9	9	9	9	0.00010	0.00010	0.00270	0.00281
(2, 11, 2)	7	19	32	9	9	9	9	0.00011	0.00011	0.00622	0.00640
(2, 12, 2)	7	20	33	9	9	9	10	0.00013	0.00013	0.01568	0.01610
(2, 13, 2)	7	20	34	9	9	9	10	0.00016	0.00016	0.03329	0.03637
(2, 14, 2)	7	21	35	9	9	9	9	0.00018	0.00018	0.07365	0.07470
(2, 15, 2)	7	21	36	9	9	9	10	0.00021	0.00021	0.16930	0.18516
(2, 16, 2)	8	22	42	10	10	10	10	0.00025	0.00025	0.41498	0.42252
(2, 17, 2)	8	22	43	10	10	10	10	0.00030	0.00029	0.92479	0.96072
(2, 18, 2)	8	23	44	11	11	11	11	0.00037	0.00035	2.25963	2.35605
(2, 19, 2)	8	23	45	11	11	11	11	0.00040	0.00043	5.03183	5.68974
(2, 20, 2)	8	24	46	11	11	11	11	0.00045	0.00046	11.05169	12.24347
(2, 21, 2)	8	24	47	11	11	11	11	0.00049	0.00051	24.24262	25.30102
(2, 22, 2)	8	24	48	11	11	11	11	0.00057	0.00055	52.64991	54.75317
(2, 23, 2)	8	25	48	11	11	11	11	0.00063	0.00061	112.64090	117.66975
(2, 24, 2)	8	25	49	11	11	11	12	0.00071	0.00067	252.76384	276.35139
(2, 25, 2)	8	25	50	11	11	11	11	0.00074	0.00080	532.60420	575.33891

Table 4.1.: Comparison of gAETG (Algorithm 6) and DDA (Algorithm 7 Algorithms by means of some instances of binary CAs of strength two. In the columns under the header # **Rows** the number of rows of the generated CAs is denoted, where smaller values are considered better.

Table 4.1 shows the results for various instances of the form $CA(N; 2, k, 2)$. The CAN for these instances is known (see Corollary 2.52). The table additionally lists the upper bound obtained from Corollary 4.29 respectively Corollary 1.28 equation (1.6) (computed as $\lfloor -\frac{\log v^t \cdot \binom{k}{t}}{\log(1-\frac{1}{v^t})} + 1\rfloor$) in the column Bound$_L$ as well as the one obtained from Theorem 4.31 (computed as $\lfloor H\left(\binom{k}{t}\right) \cdot CAN\rfloor$) in the column Bound$_H$. The results display some remarkable characteristics. While the CAN was only reached for two very small configurations, all computed CAs stay well under the size afforded by the upper bounds in the third and fourth column. Although gAETG tends to construct slightly smaller arrays in the general case (while consuming much more time), it is particularly badly suited for $CA(N; 2, k, 2)$ due to its constant worst-case performance, i.e. a search over all 2^k possible rows, as well as the lack of a reduction of the output size, in comparison to DDA, for the small values of k in the experiments in Table 4.1. Further experiments with DDA$_{avg}$ and DDA$_{max}$ show a similar deviation from both CAN and the upper bound, but were not included in the table due to the infeasibility of executing the required runs with gAETG.

Instance				**# Rows**				**Time (s)**			
(t, k, v)	CAN	Bound$_L$	Bound$_H$	DDA		gAETG		DDA		gAETG	
				avg	*max*	*best*	*worst*	*avg*	*max*	*best*	*worst*
(3, 5, 4)	64	411	187	107	64	65	94	0.00038	0.00023	0.00590	0.00862
(3, 5, 8)	512	4369	1499	862	768	662	712	0.00636	0.00570	1.85988	2.08355
(3, 5, 16)	4096	43496	11997	6745	4096	4952	5048	0.25057	0.15362	456.94628	469.80281
(2, 6, 5)	25	146	82	46	40	32	34	0.00019	0.00017	0.05431	0.05824
(3, 6, 5)	125	975	449	250	241	186	195	0.00171	0.00166	0.44340	0.46224
(4, 6, 5)	625	5712	2073	1216	1145	963	972	0.01648	0.01565	1.94444	1.96441
(4, 13, 2)	32	145	228	54	54	45	49	0.00826	0.00839	1.57704	1.73014
(5, 8, 2)	52	236	239	72	72	64	70	0.00113	0.00114	0.00763	0.00855
(5, 9, 2)	54	262	292	85	85	72	79	0.00276	0.00276	0.03488	0.03721

Table 4.2.: Comparison of gAETG and DDA by means of some instances of orthogonal arrays and some new known values of CANs. In the columns under the header # **Rows** the number of rows of the generated CAs is documented, where smaller values are considered better.

Table 4.2 shows output in the same format for two additional types of CA configurations: $CA(N; 3, 5, v)$ where $v \in \{4, 8, 16\}$ and $CA(N; t, 6, 5)$ where $t \in \{2, 3, 4\}$, for which the existence of an orthogonal array is guaranteed by Theorem 2.2. Additionally experiments for three instances, where the CAN was recently determined in [44] for $CA(N;4,13,2)$, $CA(N;5,8,2)$ and $CA(N;5,9,2)$, were run. Results for $CA(N;3,5,v)$ exhibit a similar pattern as those in Table 4.1, with drastically greater runtime for GAETG when v is increased. However, a much more exciting result was obtained for DDA$_{max}$, which produces either the orthogonal array (for $\log_2(v) \equiv 0 \mod 2$) or a CA with exactly $3/2 \cdot CAN$ rows (for $\log_2(v) \equiv 1 \mod 2$), in all tested instances of the type $\mathsf{CA}(3,k,2^i)$ such that an orthogonal array exists due to Theorem 2.2. A formal explanation for this behaviour is subject of future work. As the results for $CA(N;t,6,5)$ show, GAETG fares much better when v and k are kept constant and the strength is increased. For these configurations, the size advantage versus DDA is significant while runtime only increases by a small factor. This trend continues to exist also for the remaining of the test cases shown in Table 4.2. Furthermore, the number of rows continues to keep well below the upper bounds Bound$_L$ and Bound$_H$, as $0.0941 \leq \frac{min(N)}{\text{Bound}_L} \leq 0.5$ and $0.1973 \leq \frac{min(N)}{\text{Bound}_H} \leq 0.5715$ in all our experiments. This shows that, for the CA configurations we experimented with, these upper bounds are not tight. Whether this holds in general and tighter bounds can be found, is subject to further investigation.

Table 4.5 displays the results of experiments involving a weighted budgeted $MCA(N;t,10,(7,6,4^2,3^2,2^4))$ with $t \in \{3,\ldots,6\}$. Weights are assigned individually to each value of a column. The weight of a $(v_1,\ldots,v_k)$-ary t-tuple is the product of the weights of each involved value. Three different patterns of weight distributions were chosen: Unweighted (UN), just as for to the previous experiments; random weights (RAND), detailed in Table 4.3, which were generated using a pseudo-random number generator; and manual weights (MAN), shown in Table 4.4. Budgets were chosen based on preliminary experiments in order to restrict arrays to approximately half their required size. The resulting weights in Table 4.5 are rounded to two decimal places. Note that unlike the tables previously described (which list the number of rows output by the respective algorithm), higher values in the columns under the label **Weight**, are better.

Figures 4.8a and 4.8b show the weight gain per row for strength 2 for the RAND and MAN distributions (plots for higher strengths exhibit the same pattern). In term of WBCAs, the x-axis reflects the budget (number of rows) while the y-axis displays the cumulative weight.

The results are largely similar to unweighted versions, which leads to the hypothesis that weights have no significant impact on the performance of either algorithm. One

Column	**Value Weights**						
	1	2	3	4	5	6	7
1	0.81	0.56	0.14	0.41	0.8	0.32	0.86
2	0.06	0.44	0.7	0.91	0.81	0.56	
3	0.14	0.41	0.8	0.32			
4	0.86	0.06	0.44	0.7			
5	0.91	0.81	0.67				
6	0.02	0.78	0.03				
7	0.44	0.21					
8	0.9	0.74					
9	0.73	0.2					
10	0.44	0.43					

Table 4.3.: Weights of column-value assignments under RAND weight distribution.

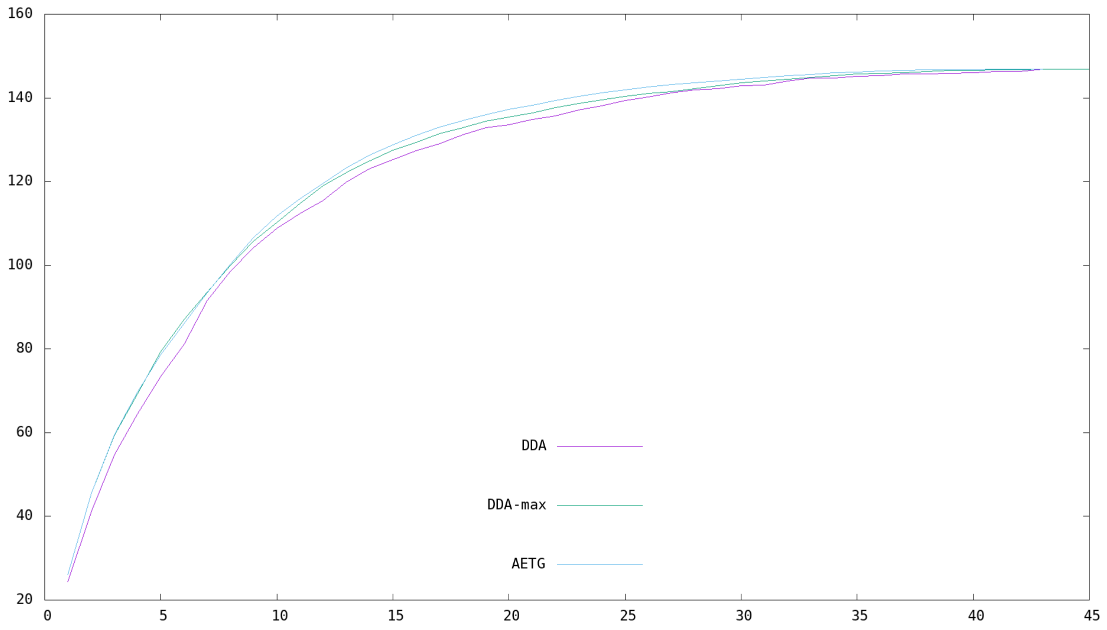
Column	**Value Weights**						
	1	2	3	4	5	6	7
1	0.75	1	0.5	1	0.5	0.75	0.5
2	0.5	0.25	0.25	0.25	0.5	0.25	
3	0.5	0.25	0.5	0.25			
4	0.5	0.25	0.5	0.25			
5	0.25	0.25	0.25				
6	0.25	0.25	0.25				
7	0.25	0.25					
8	0.5	0.5					
9	0.25	0.75					
10	0.25	0.5					

Table 4.4.: Weights of column-value assignments under MAN weight distribution.

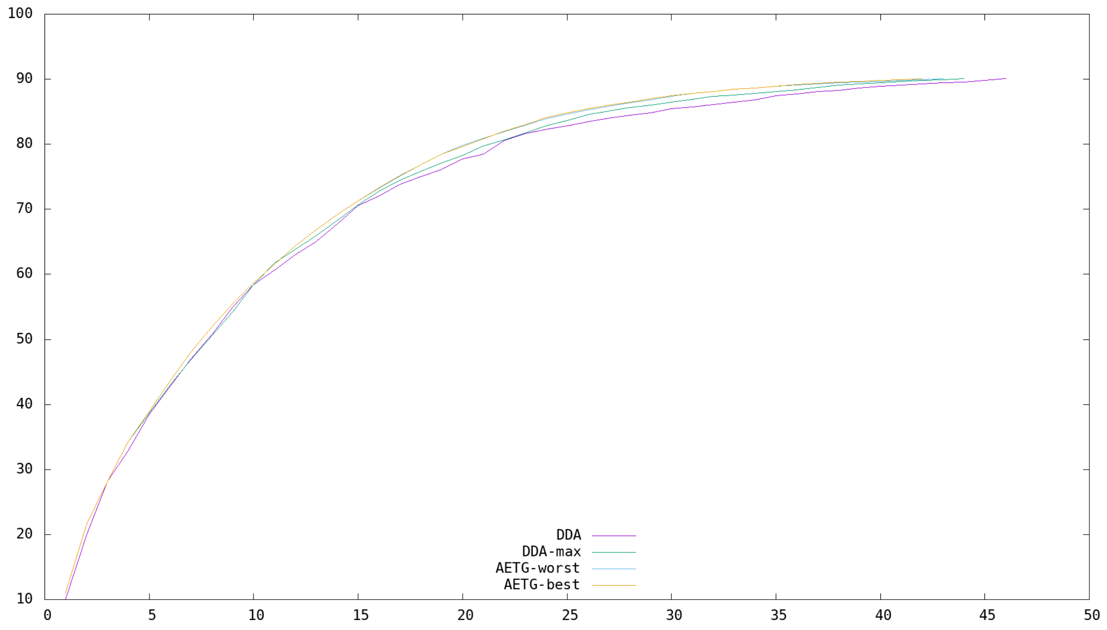
surprising effect occurs in the case of RAND weights: Despite beginning its search at random locations in the search space, it always produces arrays of the same size. This is reflected in Table 4.5 by WBGAETG$_{best}$ and WBGAETG$_{worst}$ always producing arrays covering the same weight. One explanation for this behaviour would be that the RAND weight distribution produces near-unique tuple weights (and the sum of all tuples newly covered by a row becomes similarly near-unique) such that their order of being selected by the algorithm essentially becomes deterministic. This hypothesis is also supported by the fact that for the MAN distribution, *most* executions of GAETG (but not all) result in the same output weight covered.

Instance			**Weight**			
t	Distribution	Budget	WBDDA *avg*	WBDDA *max*	WBGAETG *best*	WBGAETG *worst*
2	UN	20	113.00	117.00	119.00	117.75
2	RAND	20	133.65	135.41	137.22	137.22
2	MAN	20	77.75	78.25	79.88	79.69
3	UN	100	545.63	545.75	554.00	552.63
3	RAND	100	640.72	641.78	648.32	648.32
3	MAN	100	281.03	282.88	285.81	285.45
4	UN	400	1582.81	1587.62	1602.50	1599.94
4	RAND	400	1870.93	1874.59	1883.77	1883.77
4	MAN	400	616.29	617.57	622.49	622.42
5	UN	1250	3001.44	3005.06	3035.22	3032.12
5	RAND	1250	3563.69	3567.04	3579.06	3579.06
5	MAN	1250	863.77	864.77	872.52	871.75
6	UN	3500	3844.27	3851.59	3891.84	3889.48
6	RAND	3500	4528.13	4531.45	4544.47	4544.47
6	MAN	3500	807.72	808.25	815.53	815.18

Table 4.5.: Results for WBCA Instances.



(a) Weight growth per test for random weights (RAND).



(b) Weight growth per test for manual weights (MAN).

Figure 4.8.: Weight growth per test for different algorithms applied to weighted instances

5. A Family of Algorithms based on IFSs

In this chapter we show yet another set-based method for constructing CAs, which in this case is based on independent families of sets (IFSs). We make use of the equivalence of IFSs and binary CAs, as shown in Chapter 2, so we can interchangeably use these two structures in terms of the introduced concept and algorithmic design. Using an idea presented in [30], this set-based method is extended with *balancing properties* which can impose restrictions on the number of appearing tuples in the array corresponding to an IFS. This, among other concepts, enables the definition of different building blocks that give rise to the *IFS-family* of algorithms for constructing IFSs (and consequently also for CAs), including two new algorithms extending the method presented in [30]. Finally we give a comparison of these algorithms against the widely used algorithms of the IPO-family (previously described in Subsection 3.1.3), which bare similarities with the presented approach for constructing and extending CAs. The results show that that the IFS-family outperforms the IPO-family in many of the documented cases in terms of generating CAs with less rows.

Remark. Note that in this chapter we restrict our considerations to binary CAs and carry out arguments only for this class of objects. As in Section 2.3 we use the notation $\bar{A}$ for a variable that can either take the value A or A^C, with respect to the considered underlying set, to simplify the notation.

In this chapter we follow mainly the work in [48].

5.1. A Balancing Property

In [30] Freiman et al., proposed an algorithm that produces exponentially sized (in terms of the cardinality of the underlying set) IFSs, one set at a time. Before we

formalize this approach in Section 5.2, let us first describe the idea behind it and motivate the terminology needed to describe the algorithm formally.
It is well known that orthogonal arrays of index unity are optimal CAs (see Theorem 1.22 and also [21]), i.e. within each selection of t columns of the array each binary t-tuple appears exactly once. Also when constructing a CA with as few rows as possible, one tends to cover certain t-tuples only once rather than multiple times. Hence the objective is to cover as few t-tuples as possible more than once. Let us consider the case of a CA $(\mathbf{a}_1, \ldots, \mathbf{a}_r)$, where only few t-tuples appear more than once within a certain choice $(\mathbf{a}_{i_1}, \ldots, \mathbf{a}_{i_t})$ of t columns of that array. Since for each $(t-1)$-tuple $(u_1, \ldots, u_{t-1})$ there are exactly two binary t-tuples, that start with $(u_1, \ldots, u_{t-1})$, namely $(u_1, \ldots, u_{t-1}, 0)$ and $(u_1, \ldots, u_{t-1}, 1)$. We know that within $(a_{i_1}, \ldots, a_{i_{t-1}})$ each $(t-1)$-tuple appears at least twice, and only few of them appear more than twice. Of course this argumentation holds for each choice of $(t-1)$ columns of $(a_{i_1}, \ldots, a_{i_t})$.

Remark 5.1. Note as well that this argumentation can be iterated. From these remarks we design a necessary condition, when a column is allowed to be added to the current array. In particular, we want to ensure a minimum amount of *balance* among the columns of the array, in the sense just described.

For this reason, we introduce the notion of α-balance.

Definition 5.2. Let $\mathcal{A} = (A_1, \ldots, A_k)$ be a family of sets $A_i \subseteq [N]$, $\forall i \in [k]$ and $\alpha = (\alpha_1, \ldots, \alpha_s) \in \mathbb{N}^s$, $s \leq k$. We say that A *is tuple-balanced with respect to* α *(or* α*-balanced for short), if* $\forall i \in \{1, \ldots, s\}$, $\forall \{j_1, \ldots, j_i\} \subseteq \{1, \ldots, k\}$ *and* $\forall \bar{A}_{j_r} \in \{A_{j_r}, A_{j_r}^C\}$, *we have:*

$$|\bigcap_{r=1}^{i} \bar{A}_{j_r}| \geq \alpha_i. \tag{5.1}$$

Note that if a family of sets is tuple-balanced w.r.t. $\alpha = (\alpha_1, \ldots, \alpha_s)$ and $\alpha_s \geq 1$, it is also s-independent.
Due to the one to one correspondence between subsets of $[N]$ and binary vectors of length N (see also Theorem 2.46), this definition translates naturally to the language of binary arrays. Therefore, in the following, it makes sense to consider *α-balanced arrays.* Inherited from the corresponding family of sets, $(\alpha_1, \ldots, \alpha_s)$-balanced arrays have the property that within each choice of $i \leq s$ columns, each binary i-tuple appears at least α_i times. We also introduce the following notion.

Definition 5.3. Let $B \subseteq [N]$, $\mathcal{A} = (A_1, \dots, A_k)$ be a family of sets $A_i \subseteq [N]$, $\forall i \in \{1, \dots, k\}$ and $\alpha = (\alpha_1, \dots, \alpha_s) \in \mathbb{N}^s$. We say that B *is tuple-balanced with respect to* A *and* α *(or simply* B *is* α*-balanced with respect to* A*), if the family of sets* $(A_1, \dots, A_k, B)$ *is tuple-balanced with respect to* α*.*

As above, we use the same terminology for binary arrays, i.e. we say a vector $\mathbf{b}$ is tuple-balanced w.r.t. $(\mathbf{a}_1, \dots, \mathbf{a}_k)$ and α if $(\mathbf{a}_1, \dots, \mathbf{a}_k, \mathbf{b})$ is α-balanced. We illustrate the introduced concepts with the following example.

Example 5.4. Let $\alpha = (6, 3, 1)$, $\mathcal{A} = (A_1, A_2)$ and

$$\begin{aligned} A_1 &= \{1, 2, 3, 4, 9, 10\}, \\ A_2 &= \{1, 2, 5, 6, 9, 11\}, \\ B_1 &= \{1, 3, 5, 7, 10, 12\}, \\ B_2 &= \{1, 3, 5, 7, 10, 11\} \end{aligned}$$

One can easily verify that $\mathcal{A} = (A_1, A_2)$ is an IFS over $\{1, \dots, 12\}$ of strength 2 and is tuple-balanced with respect to α. One may also verify that the family of sets (A_1, A_2, B_1) is in fact 3-independent. Nevertheless B_1 is not tuple-balanced with respect to $\mathcal{A}$ and α, since $|A_2 \cap B_1| = |\{1, 5\}| < 3 = \alpha_2$, which violates condition (5.1). B_2 provides an example that is tuple-balanced with respect to $\mathcal{A}$ and α, since (A_1, A_2, B_2) fulfills condition (5.1): each set as well as its complement has at least 6 elements, intersections of any two sets (or their complements) contain at least 3 elements, and the intersection of the three sets (where some of them might be replaced by their complement) contains at least one element.

Vector notation might make this more visible. Let $\mathbf{a_i}$ denote the indicator vector of A_i for $i = 1, 2$ and $\mathbf{b}_i$ that of B_i respectively. Therefore we can consider the array $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1)$ of column-wise indicator vectors of A_1, A_2 and B_1

$$(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1) = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}^T,$$

and the array $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_2)$ of column-wise indicator vectors of A_1, A_2 and B_2

$$(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_2) \;=\; \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}^T$$

First of all one can verify that the arrays $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1)$ and $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_2)$ both are binary CAs of strength 3, but only $\mathbf{b}_2$ is a tuple-balanced vector w.r.t. $(\mathbf{a}_1, \mathbf{a}_2)$ and $(6, 3, 1)$, since additionally within every selection of two columns of $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_2)$, each 2-tuple appears at least 3 times, and within every column each value appears at least 6 times.

The previous example exemplifies that $(\alpha_1, \ldots, \alpha_t)$-balance with $\alpha_t \geq 1$ is a stronger property than t-independence. In the sequel we will use the notion of α-balance as a decision criterion in a heuristic for constructing IFSs.

5.2. A Family of Algorithms for Generation of IFSs

In this section we propose a variety of algorithms, IFS-ORIGIN, IFS-GREEDY and IFS-SCORE, based on independent families of sets. We call these algorithms collectively the family of IFS-Algorithms. In particular the method described earlier, is formalized and extended, in terms of a combinatorial algorithmic design. The design is composed of the following five *building blocks*: *store*, *select*, *admissible*, *extend* and *update*, which we state below.

– *Store:* The *store* is a data structure that serves as a resource, from which the sets to build the target IFS are chosen. It may be static or dynamic.

– *Select:* A procedure which returns one element of the *store*, e.g. randomly or via a scoring function.

– *Admissible:* This procedure decides whether a certain element is allowed to be added to the current IFS or not, under certain admissible criteria, which can be based for example on the concept of α-balance.

– *Extend:* A procedure which extends the current IFS.

– *Update:* A procedure which updates the *store* in case it is dynamic.

Building blocks \ Algorithm	IFS-ORIGIN	IFS-GREEDY	IFS-SCORE
Store	F_2	F_2	F_2
Select	SELECTRANDOM	SELECTNEXT	SELECTSCORE
Admissible	ADMISSIBLE$_\alpha$	ADMISSIBLE$_\alpha$	ADMISSIBLE
Extend	EXTEND	EXTEND	EXTEND
Update	UPDATE$_\alpha$	-	UPDATE

Table 5.1.: Composition of the IFS-family algorithms.

A comprehensive overview of the proposed algorithms via their building blocks, is given in Table 5.1.

5.2.1. IFS-Origin

Initially we give a short algorithmic description of the method proposed in [30] and extended in the previous subsection. We refer to it and its implementation as IFS-ORIGIN (Algorithm 13). The algorithm takes as input the size N of the base set and the strength t of the to be constructed IFS. The initial STORE, S_0, is set to be equal to the maximal 2-independent family of sets $\mathcal{F}_2(N)$, as described in Theorem 2.51, as it contains all sets that are balanced w.r.t $(\lfloor N/2 \rfloor)$, i.e. each set individually is optimally balanced. The initial set of the IFS, A_1, is set to be a random element of the STORE. This random initialization is justified because picking a different initial element comes down to permuting the elements of $[N]$, which respects Definition 2.43, and keeps $\mathcal{F}_2(N)$ invariant. After the initialization step ($i = 1$), in the i-th step, IFS-ORIGIN traverses through the whole STORE S_{i-1} given at that time, updating it by removing all non ADMISSIBLE$_\alpha$ elements from it, which yields S_i. For the check of admissibility the algorithm requires a vector $\alpha = (\alpha_1, \dots, \alpha_t)$, which encodes the desired balance of i-tuples for $i = 1, \dots, t$. When the target is to construct a t-independent family of sets, then $\alpha_t \geq 1$ has to be ensured. After the update S_i is left with only ADMISSIBLE$_\alpha$ elements, a random element is chosen and added to the IFS at hand, yielding A_{i+1}. The algorithm terminates when the STORE is empty. Using the building blocks from Table 5.1, we express IFS-ORIGIN in pseudocode form (see Algorithm 13).

Note that the nature of this algorithm immediately yields an upper bound on the number of calls of the subprocedure UPDATE (line 15), occurring during one run of the algorithm. Let k be the size of the returned IFS, A, then the number of calls of UPDATE is also exactly k, since the STORE gets updated once after each element that is added to the IFS, which is about to be constructed.

5.2.2. IFS-Greedy

When being familiar with IFS-ORIGIN described above, one will realize, that this version, as was originally described in [30], lacks a method to decide which one of the elements in the remaining STORE should be added to the current array. Particularly in IFS-ORIGIN this is done via a random pick, which in retrospect makes the UPDATE of the STORE, determining *all* ADMISSIBLE$_\alpha$ elements, unnecessary. The newly proposed IFS-GREEDY version (see Algorithm 14) bypassess this decision problem by simply picking the next found ADMISSIBLE$_\alpha$ element of the STORE, with the advantage of the fact that the STORE never needs to be updated. The initialization stays the same as in IFS-ORIGIN. After that, IFS-GREEDY traverses the STORE only once, adding the first set that is ADMISSIBLE with respect to the already constructed IFS and α (recall Definition 5.3), i.e. the STORE never gets updated.

5.2.3. IFS-Score

The overall structure of IFS-SCORE is the same as that of IFS-ORIGIN, but in lines 10 and 29 of Algorithm 15 different sub procedures SELECTSCORE and ADMISSIBLE are defined. To circumvent the problem of IFS-ORIGIN of picking a *random element* from the updated STORE, we calculate a score for each element of the STORE and add the one (or one of them, since ties may occur) with the lowest score (see procedure SELECTSCORE of Algorithm 15). Each element is initialized with a score of zero. In the i-th step of the algorithm an individual score for each element of the current STORE S_{i-1} is computed. This score can be based on the tuple-balance of the family (A_i, b) for example. This also has the advantage that IFS-SCORE does not requiring an input for α. Since we compute a score for each element, we already encounter the tuple balance of (A_i, b) to our selection, and we do not need to dictate a-priori via α how often certain i-tuples have to appear. Therefore IFS-SCORE is

Algorithm 13 IFS-ORIGIN(N, t)

```
 1: $S = \mathcal{F}_2(N)$                                  ▷ S initializes a STORE
 2: $A \leftarrow$ SELECTRANDOM(S)
 3: while $S \neq \emptyset$ do
 4:     $S \leftarrow$ UPDATE$(S, A, t)$
 5:     if $S \neq \emptyset$ then
 6:         $b \leftarrow$ SELECTRANDOM$(S)$
 7:         $A \leftarrow$ EXTEND$(A, b)$
 8:     end if
 9: end while
        return $A$

10: procedure SELECTRANDOM$(S)$
        return random element of $S$
11: end procedure

12: procedure EXTEND$(A, b)$
13:     $A \leftarrow [A, b]$
        return $A$
14: end procedure

15: procedure UPDATE$_\alpha(S, A, t)$
16:     for b in S do
17:         if not ADMISSIBLE$_\alpha(A, b, t)$ then
18:             $S \leftarrow S \setminus \{b\}$
19:         end if
20:     end for
21: end procedure

22: procedure ADMISSIBLE$_\alpha(A, b, t)$
Require: $\alpha_{1\times t}$ with $\alpha_t \geq 1$            ▷ $\alpha$ is a vector of length $t$
23:     if [A,b] is $\alpha$-balanced then
        return True
24:     else
        return False
25:     end if
26: end procedure
```

Algorithm 14 IFS-GREEDY(N, t)

```
1: $S = \mathcal{F}_2(N)$                                  ▷ S initializes a STORE
2: $b \leftarrow$ SELECTNEXT$(S, \emptyset)$
3: $A \leftarrow [b]$
4:
5: while b has next do
6:     $b \leftarrow$ SELECTNEXT$(S, b)$
7:     if ADMISSIBLE$_\alpha(A, b, t)$ then
8:         $A \leftarrow$ EXTEND$(A, b)$
9:     end if
10: end while
        return $A$

11: procedure SELECTNEXT$(S, b)$
12:     if $b = \emptyset$ then
        return first element of $S$
13:     else
        return next element to $b$ from $S$
14:     end if
15: end procedure

16: procedure EXTEND$(A, b)$
17:     $A \leftarrow [A, b]$
        return $A$
18: end procedure

19: procedure ADMISSIBLE$_\alpha(A, b, t)$
Require: $\alpha_{1\times t}$ with $\alpha_t \geq 1$                ▷ $\alpha$ is a vector of length $t$
20:     if [A,b] is $\alpha$-balanced then
        return True
21:     else
        return False
22:     end if
23: end procedure
```

the only algorithm in the proposed IFS-family that does *not* require an input of α. Consequently, in the decision criterion of ADMISSIBLE, we require only that (A, b) is t-independent, instead of it being α-balanced.

5.3. Results

As a proof of concept of the algorithmic design (presented in Section 5.2), we compare the implementations of the IFS-family of algorithms for $t = 3$ to two of the most commonly used greedy algorithms of the IPO-family, namely IPOG [66] and IPOG-F [29]. In addition, we evaluate the results versus the *current* best known upper bounds for $\mathsf{CAK}(N; 3, 2)$ retrieved from [19]. To the best of the author's knowledge the algorithms of the IPO-family are the only ones that generate CAs using a horizontal extension step similar to the one proposed in the IFS-family of algorithms.
Table 5.2 shows the number of columns a binary CA of strength 3 can attain by either the respective algorithm or according to [19]. The table starts with $N = 8$, since there are at least eight rows needed to cover all eight binary 3-tuples. It shows that the IFS-family of algorithms improves significantly over IPOG and IPOG-F in almost every case presented, as well as IFS-GREEDY and IFS-SCORE improve over IFS-ORIGIN. It is also worth pointing out that during the computations larger families were obtained, when running IFS-ORIGIN and IFS-GREEDY on more restrictive α-vectors, than when running them on less restrictive α-vectors. We believe the concept of admissibility via α-balance (and its requirement per different IFS algorithms) makes the difference compared to IPOG and IPOG-F, since these algorithms lack a balancing strategy during horizontal extension. Regarding our results, we want to highlight that IFS-SCORE is able to deliver almost the same size of output IFS as IFS-GREEDY without the need of an α-vector as input. On the other hand, IFS-SCORE is more complex than IFS-GREEDY and even IFS-ORIGIN due to the computation of scores.
We ran the algorithms IPOG and IPOG-F as they are implemented in ACTS (Version 2.93), a CA generation tool provided by NIST [99], [76]. For the input values of N in Table 5.2, IPOG and IPOG-F were considerably faster than all three algorithms of the IFS-family. The extra computations are justified in so far that, the IFS-family of algorithms outperforms IPOG and IPOG-F, in 14 out of the 18

Algorithm 15 IFS-SCORE(N, t)

```
1: $S = \mathcal{F}_2(N)$                                   ▷ S initializes a STORE
2: $A \leftarrow$ SELECTSCORE(S)
3: while $S \neq \emptyset$ do
4:     $S \leftarrow$ UPDATE$(S, A, t)$
5:     if $S \neq \emptyset$ then
6:         $b \leftarrow$ SELECTSCORE$(S)$
7:         $A \leftarrow$ EXTEND$(A, b)$
8:     end if
9: end while
       return $A$

10: procedure SELECTSCORE$(S, A, N, t)$
11:     $M \leftarrow \emptyset$
12:     $min \leftarrow \infty$
13:     for $b \in S$ do
14:         if SCORE$(b, A, N, t) < min$ then
15:             $min \leftarrow$ SCORE$(b, A, N, t)$
16:             $M \leftarrow [b]$
17:         else if SCORE$(b, A, N, t) = min$ then
18:             $M \leftarrow [M, b]$
19:         end if
20:     end for
        return random element of M
21: end procedure

22: procedure UPDATE$(S, A, t)$
23:     for b in S do
24:         if not ADMISSIBLE$(A, b, t)$ then
25:             $S \leftarrow S \setminus \{b\}$
26:         end if
27:     end for
28: end procedure

29: procedure ADMISSIBLE$(A, b, t)$
30:     if [A,b] is t-independent then
        return True
31:     else
        return False
32:     end if
33: end procedure

34: procedure SCORE$(A, b, N, t)$
35:     return $score$              ▷ $score$ is calculated from the tuple-balance of $(A, b)$
36: end procedure
```

documented cases in terms of output size of produced IFS (or columns of produced CAs, c.f. Theorem 2.46) and achieves the same size values in the other four.

N	IPOG-F	IPOG	IFS-ORIGIN	IFS-GREEDY	IFS-SCORE	Colbourn Tables
8	4	4	4^a	4^a	4	4
9	4	4	4^a	4^a	4	4
10	4	4	4^a	5^a	5	5
11	5	4	4^a	5^a	5	5
12	5	6	11^b	11^b	11	11
13	5	6	6^b	11^b	11	11
14	6	6	6^b	11^b	11	11
15	6	6	7^b	11^b	11	12
16	7	7	8^c	14^c	14	14
17	9	7	10^c	14^c	14	16
18	11	8	12^c	17^c	16	20
19	12	8	13^c	17^c	16	22
20	13	10	11^d	19^d	19	23
21	15	10	15^c	19^c	19	25
22	16	12	18^c	21^c	21	26
23	16	13	19^c	23^c	22	30
24	19	13	23^d	26^d	25	38
25	21	14	24^c	28^a	26	44

Table 5.2.: Comparison of the number of columns attained for N rows by different CA algorithms. The (currently) best lower bound for $\mathsf{CAK}(N;3,2)$ is provided by the Tables provided by Colbourn [19], and is given in the last column. The superscripts are representing the different α-vectors, which were used as input for the computation that yield the output IFS, where a=(4,2,1), b=(6,3,1), c=(8,4,1), d=(10,5,1).

6. Conclusion and Future Work

This thesis provided an introductory overview of the topic of *covering arrays* (CAs) and generalizations, highlighting specific aspects of this field. The first part of this thesis concerned theoretical results related to CAs and furthermore connections to other fields of discrete mathematics, such as two constructions for special classes of optimal CAs, binary CAs of strength two and CAs over prime fields. Other results, such as the logarithmic growth of the covering array number (CAN) in the number of columns, were proven for the general case. As so often in mathematics, the former mentioned result only proves the existence of an optimal CA with $O(\log k)$ rows and k columns, but gives no method of construction for such a CA. Thus, in the second part of this thesis, the focus is shifted to algorithms for CA generation. After a brief summary of popular CA generation methods, two of them were analyzed in detail. In particular, after the preliminaries given in Chapter 1, in Chapter 2 we described various combinatorial constructions for CAs. A construction for orthogonal arrays $\mathsf{OA}_1(q^t; t, q+1, q)$ for prime powers q, based on properties of finite fields, was described first. Next, we presented a construction of CAs using group actions that act on certain matrices, which essentially reduces the problem of generating the desired CA to the problem of finding an appropriate group G and a matrix M with the necessary properties, such that after *developing* M by G a CA can easily be constructed. Finding other constructions involving groups acting on matrices, such that CAs with a possibly small number of rows are generated, is subject to future work, especially considering that such constructions can help determining covering array numbers. After that, we shifted the focus to plug-in constructions, proving that the *plug-in* of a family of CAs into a given CA yields again a CA. This property enabled us to formulate two constructions for *nested CAs* and *refined nested CAs*, where the latter makes use of a more sophisticated plug-in construction. Additionally, analogues and applications of such nested CAs to modular software designs were explained. A second plug-in construction making use of *perfect hash families* revealed connections

of CAs to other classes of designs and to the field of *Error-Correcting Codes*. The last combinatorial construction considered in Chapter 2 makes use of the representation of binary CAs as *independent families of sets*. We gave a theorem which determines the size of a maximal 2-independent family of sets and also describes how such families can be constructed. Due to the proven equivalence of binary CAs and independent families of sets, the problem of generating $\mathsf{CA}(N; 2, k, 2)$ with the smallest possible N is solved completely for arbitrary k.
In Chapter 3 we gave an overview of popular algorithms for CA generation, depicting greedy algorithms growing CAs *horizontally*, *vertically* or in two dimensions, metaheuristic algorithms based on *hill climbing*, *simulated annealing* or *tabu search*, as well as hybrid and exact approaches.
In Chapter 4 we discussed connections between CAs and *set covers* (SCs) and showed how the former can be considered a special instance of the latter using the appropriate mappings and translation algorithms. This enabled us to apply known results for SCs to CAs and to their respective generalizations. A number of experiments show that the CA specific algorithms – DDA (Algorithm 7) and WBDDA (Algorithm 9) – generate solutions similar in quality (number of rows respectively weight of covered tuples under budget constraints) to the algorithms obtained from SC heuristics, namely GAETG (Algorithm 6) and WBGAETG (Algorithm 12). While the latter tend to produce slightly smaller (or higher weighted) output, producing arrays that would generally be regarded as *better*, the CA specific strategies consume significantly less time, particularly for CAs with a large number of columns or values. Experiments with the weighted versions of the algorithms led to similar observations. A formal explanation for the behavior of DDA_{max} for the tested CA parameters $(3, 5, 2^i)$, $i \in \mathbb{N}$ can be regarded as future work, just as the question of the existence of tighter upper bounds for general covering array numbers is subject to further investigation.
In Chapter 5 we presented a family of combinatorial algorithms for constructing independent families of sets, and hence, due to Theorem 2.46, binary covering arrays. As the presented algorithmic design is modular, its building blocks can give rise to further algorithms not presented in this thesis. We introduced the concept of α-balance, which can impose certain restrictions on the number of tuples that can appear in such arrays, as a means of generating higher quality covering arrays. As a proof of concept of this approach, the implementations of the proposed family

were compared against existing greedy algorithms. Enhancing the functionality of the presented algorithms via improving the scoring function based on α-balance, generalizations for larger and mixed alphabet sizes as well as conducting more experiments for higher strength covering arrays are considered part of future work.
To conclude, the results presented in the first part of this thesis raise the question of further connections between (optimal) CA constructions and other fields of mathematics. Similarly, improvements of the algorithms discussed in the second part can be considered as part of future work. The fact that the complexity of the optimal CA generation problem is yet unknown poses interesting additional challenges and is subject to future investigation. The fact that CAs find applications in automated software testing and other domains further reinforces the need to answer these questions, especially considering the growing amount of software artifacts in modern information society.

List of Acronyms

CA(s)	Covering Array(s)
CAN	Covering Array Number
IFP	Independent Family of Partitions
IFS	Independent Family of Sets
IFSs	Independent Families of Sets
MCA(s)	Mixed level Covering Array(s)
MCAN	mixed level Covering Array Number
OA(s)	Orthogonal Array(s)
PHF	Perfect Hash Family
VCA	Variable strength Covering Array
VCAN	Variable strength Covering Array Number
WBCA	Weighted Budgeted Covering Array

List of Tables

List of Figures

List of Algorithms

Bibliography

[1] N Alon. Explicit construction of exponential sized families of k-independent sets. *Discrete Mathematics*, 58(2):191 – 193, 1986.

[2] Ian Anderson. *Combinatorics of finite sets.* Courier Corporation, 1987.

[3] Egon Balas and Manfred W. Padberg. Set partitioning: A survey. *SIAM review*, 18(4):710–760, 1976.

[4] Renée C. Bryce and Charles J. Colbourn. Test prioritization for pairwise interaction coverage. In *Proceedings of the 1st International Workshop on Advances in Model-based Testing*, A-MOST '05, pages 1–7, New York, NY, USA, 2005. ACM.

[5] Renée C. Bryce and Charles J. Colbourn. Prioritized interaction testing for pair-wise coverage with seeding and constraints. *Information and Software Technology*, 48(10):960 – 970, 2006. Advances in Model-based Testing.

[6] Renée C. Bryce and Charles J. Colbourn. The density algorithm for pairwise interaction testing. *Software Testing, Verification and Reliability*, 17(3):159–182, 2007.

[7] Renée C. Bryce and Charles J. Colbourn. A density-based greedy algorithm for higher strength covering arrays. *Softw. Test. Verif. Reliab.*, 19(1):37–53, March 2009.

[8] Renée C. Bryce, Charles J. Colbourn, and Myra B. Cohen. A framework of greedy methods for constructing interaction test suites. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 146–155, New York, NY, USA, 2005. ACM.

[9] Renée C. Bryce, Charles J. Colbourn, and Myra B. Cohen. A framework of greedy methods for constructing interaction test suites. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 146–155, New York, NY, USA, 2005. ACM.

[10] Kenneth A Bush et al. Orthogonal arrays of index unity. *The Annals of Mathematical Statistics*, 23(3):426–434, 1952.

[11] M. A. Chateauneuf, Charles J. Colbourn, and D. L. Kreher. Covering arrays of strength three. *Designs, Codes and Cryptography*, 16(3):235–242, May 1999.

[12] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[13] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The aetg system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[14] G. D. Cohen, S. Litsyn, and C. Zemor. On greedy algorithms in coding theory. *IEEE Transactions on Information Theory*, 42(6):2053–2057, Nov 1996.

[15] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn. Constructing test suites for interaction testing. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 38–48, May 2003.

[16] Myra B. Cohen, Charles J. Colbourn, and Alan C.H. Ling. Constructing strength three covering arrays with augmented annealing. *Discrete Mathematics*, 308(13):2709 – 2722, 2008. Combinatorial Designs: A tribute to Jennifer Seberry on her 60th Birthday.

[17] Myra B Cohen, Peter B Gibbons, Warwick B Mugridge, Charles J Colbourn, and James S Collofello. A variable strength interaction testing of components. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 413–418. IEEE, 2003.

[18] Charles J Colbourn. Covering Array Tables for t=2,3,4,5,6. Available at `http://www.public.asu.edu/~ccolbou/src/tabby/catable.html`, Accessed on 2018-01-09.

[19] Charles J Colbourn. Table for CAN(3,k,2) for k up to 10000. Available at `http://www.public.asu.edu/~ccolbou/src/tabby/3-2-ca.html`, Accessed on 2018-01-09.

[20] Charles J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche*, LIX(I-II):125–172, 2004.

[21] Charles J Colbourn and Jeffrey H Dinitz. *Handbook of combinatorial designs*. CRC press, 2006.

[22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Np-completeness. In *Introduction to algorithms*, pages 1048–1053. The MIT Press, 3rd edition, 1990.

[23] Jacek Czerwonka. Pairwise testing in the real world: Practical extensions to test-case scenarios. In *Proceedings of 24th Pacific Northwest Software Quality Conference, Citeseer*, pages 419–430, 2006.

[24] S. R. Dalal, A. Jain, G. Patton, M. Rathi, and P. Seymour. Aetgsm web: a web based service for automatic efficient test generation from functional requirements. In *Proceedings. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*, pages 84–85, 1998.

[25] Siddhartha R. Dalal and Colin L. Mallows. Factor-covering designs for testing software. *Technometrics*, 40(3):234–243, 1998.

[26] Peter Danziger, Eric Mendelsohn, Lucia Moura, and Brett Stevens. *Covering Arrays Avoiding Forbidden Edges*, pages 296–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[27] P. Erdös, CHAO KO, and R. Rado. Intersection theorems for systems op finite sets. *Quart. J. Math. Oxford Ser.(2)*, 12:313–320, 1961.

[28] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.

[29] Michael Forbes, Jim Lawrence, Yu Lei, Raghu N Kacker, and D Richard Kuhn. Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5):287, 2008.

[30] G. Freiman, E. Lipkin, and L. Levitin. A polynomial algorithm for constructing families of k-independent sets. *Discrete Mathematics*, 70(2):137 – 147, 1988.

[31] Fred Glover. Tabu search–part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[32] Fred Glover. Tabu search–part II. *ORSA Journal on computing*, 2(1):4–32, 1990.

[33] Mats Grindal and Jeff Offutt. Input parameter modeling for combination strategies. In *Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering*, SE'07, pages 255–260, Anaheim, CA, USA, 2007. ACTA Press.

[34] Alan Hartman. Software and hardware testing using combinatorial covering suites. In MartinCharles Golumbic and IrithBen-Arroyo Hartman, editors, *Graph Theory, Combinatorics and Algorithms*, volume 34 of *Operations Research/Computer Science Interfaces Series*, pages 237–266. Springer US, 2005.

[35] Alan Hartman and Leonid Raskin. Problems and algorithms for covering arrays. *Discrete Mathematics*, 284(1):149 – 156, 2004. Special Issue in Honour of Curt Lindner on His 65th Birthday.

[36] Refael Hassin and Asaf Levin. A better-than-greedy approximation algorithm for the minimum set cover problem. *SIAM Journal on Computing*, 35(1):189–200, 2005.

[37] A Samad Hedayat, Neil James Alexander Sloane, and John Stufken. *Orthogonal arrays: theory and applications*. Springer Science & Business Media, 2012.

[38] Brahim Hnich, Steven D. Prestwich, Evgeny Selensky, and Barbara M. Smith. Constraint models for the covering test problem. *Constraints*, 11(2):199–219, Jul 2006.

[39] Dorit S. Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6):615–627, 1998.

[40] Iiro Honkala. A graham-sloane type construction for s-surjective matrices. *Journal of Algebraic Combinatorics*, 1(4):347–351, Dec 1992.

[41] W Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2010.

[42] Thomas W Hungerford. Algebra. 1974. *Grad. Texts in Math*, 1974.

[43] IEEE. IEEE standard for software unit testing. *ANSI/IEEE Std 1008-1987*, pages 1–28, June 1987.

[44] Idelfonso Izquierdo-Marquez and Jose Torres-Jimenez. New covering array numbers. *arXiv preprint arXiv:1711.10040*, 2017.

[45] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[46] L. Kampel, B. Garn, and D. E. Simos. Combinatorial methods for modelling composed software systems. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 229–238, March 2017.

[47] Ludwig Kampel, Manuel Leithner, Bernhard Garn, and Dimiris E. Simos. Problems and algorithms for covering arrays via set covers. *Preprint*, 2017.

[48] Ludwig Kampel and Dimitris E. Simos. Set-based algorithms for combinatorial test set generation. In Franz Wotawa, Mihai Nica, and Natalia Kushik, editors, *Testing Software and Systems*, pages 231–240, Cham, 2016. Springer International Publishing.

[49] Petteri Kaski, Patric RJ Östergård, and RJ Patric. *Classification algorithms for codes and designs*, volume 15. Springer, 2006.

[50] Gy Katona. Intersection theorems for systems of finite sets. *Acta Mathematica Hungarica*, 15(3-4):329–337, 1964.

[51] Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[52] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[53] P. Kitsos, D. E. Simos, J. Torres-Jimenez, and A. G. Voyiatzis. Exciting fpga cryptographic trojans using combinatorial testing. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 69–76, Nov 2015.

[54] K. Kleine and D. E. Simos. Coveringcerts: Combinatorial methods for x.509 certificate testing. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 69–79, March 2017.

[55] Kristoffer Kleine and Dimitris E. Simos. An efficient design and implementation of the in-parameter-order algorithm. *Mathematics in Computer Science*, Dec 2017.

[56] Daniel J Kleitman and Joel Spencer. Families of k-independent sets. *Discrete Mathematics*, 6(3):255–262, 1973.

[57] János Körner and Gábor Simonyi. A sperner-type theorem and qualitative independence. *Journal of Combinatorial Theory, Series A*, 59(1):90 – 103, 1992.

[58] R. Krishnan, S. Murali Krishna, and P. Siva Nandhan. Combinatorial testing: Learnings from our experience. *SIGSOFT Softw. Eng. Notes*, 32(3):1–8, May 2007.

[59] D.R. Kuhn, R.N. Kacker, and Y. Lei. Practical combinatorial testing. NIST Special Publication 800-142, 2010.

[60] D.R. Kuhn, R.N. Kacker, and Y. Lei. *Introduction to Combinatorial Testing*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. Taylor & Francis, 2013.

[61] V. V. Kuliamin and A. A. Petukhov. A survey of methods for constructing covering arrays. *Programming and Computer Software*, 37(3):121, 2011.

[62] Victor Kuliamin and Alexander Petukhov. Covering arrays generation methods survey. In *Proceedings of the 4th International Conference on Leveraging*

Applications of Formal Methods, Verification, and Validation - Volume Part II, ISoLA'10, pages 382–396. Springer-Verlag, 2010.

[63] B. Garn L. Kampel and D. E. Simos. Covering arrays via set covers. *to appear in Electronic Notes in Discrete Mathematics*, 2018.

[64] B. Stevens L. Moura, S. Raaphorst. The lovász local lemma and variable strength covering arrays. *to appear in Electronic Notes in Discrete Mathematics*, 2017.

[65] Jim Lawrence, Raghu N Kacker, Yu Lei, D Richard Kuhn, and Michael Forbes. A survey of binary covering arrays. *the electronic journal of combinatorics*, 18(1):P84, 2011.

[66] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. Ipog: A general strategy for t-way software testing. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pages 549–556, March 2007.

[67] Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, and James Lawrence. Ipog/ipog-d: efficient test generation for multi-way combinatorial testing. *Software Testing, Verification and Reliability*, 18(3):125–148, 2008.

[68] Robert Mandl. Orthogonal latin squares: An application of experiment design to compiler testing. *Commun. ACM*, 28(10):1054–1058, October 1985.

[69] Sosina Martirosyan and van Tran Trung. On t-covering arrays. *Designs, Codes and Cryptography*, 32(1):323–339, May 2004.

[70] Karen Meagher, Lucia Moura, and Latifa Zekaoui. Mixed covering arrays on graphs. *Journal of Combinatorial Designs*, 15(5):393–404, 2007.

[71] Karen Meagher and Brett Stevens. Group construction of covering arrays. *Journal of Combinatorial Designs*, 13(1):70–77, 2005.

[72] Lucia Moura, John Stardom, Brett Stevens, and Alan Williams. Covering arrays with mixed alphabet sizes. *Journal of Combinatorial Designs*, 11(6):413–432, 2003.

[73] Peyman Nayeri, Charles J. Colbourn, and Goran Konjevod. Randomized post-optimization of covering arrays. *European Journal of Combinatorics*, 34(1):91 – 103, 2013. Combinatorics and Stringology.

[74] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[75] Kari J. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics*, 138(1):143 – 152, 2004. Optimal Discrete Structures and Algorithms.

[76] National Institute of Standards and Technology (NIST). Downloadable tools. Available at `https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software/Downloadable-Tools#acts`, Accessed on 2018-01-21.

[77] Martyn A Ould and Charles Unwin. *Testing in software development*. Cambridge University Press, 1986.

[78] X. Qu and M. B. Cohen. A study in prioritization for higher strength combinatorial testing. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 285–294, March 2013.

[79] X. Qu, M. B. Cohen, and K. M. Woolf. Combinatorial interaction regression testing: A study of test case generation and prioritization. In *2007 IEEE International Conference on Software Maintenance*, pages 255–264, Oct 2007.

[80] Sebastian Raaphorst. *Variable strength covering arrays*. University of Ottawa (Canada), 2013.

[81] Derek JS Robinson. *A Course in the Theory of Groups*, volume 80. Springer Science & Business Media, 2012.

[82] Rust. The Rust Programming Language. Available at `https://www.rust-lang.org/en-US/`, Accessed on 2018-02-01.

[83] Jennifer Seberry and Mieko Yamada. Hadamard matrices, sequences, and block designs. *Contemporary design theory: a collection of surveys*, pages 431–560, 1992.

[84] Gadiel Seroussi and Nader H Bshouty. Vector sets for exhaustive testing of logic circuits. *IEEE Transactions on Information Theory*, 34(3):513–522, 1988.

[85] Dennis E. Shasha, Andrei Y. Kouranov, Laurence V. Lejay, Michael F. Chou, and Gloria M. Coruzzi. Using combinatorial design to study regulation by multiple input signals. a tool for parsimony in the post-genomics era. *Plant Physiology*, 127(4):1590–1594, 2001.

[86] D. E. Simos, R. Kuhn, A. G. Voyiatzis, and R. Kacker. Combinatorial methods in security testing. *IEEE Computer*, 49:40–43, 2016.

[87] Neil J. A. Sloane. A library of orthogonal arrays. Available at `http://neilsloane.com/oadir/`, Accessed on 2018-01-09.

[88] Neil JA Sloane. Covering arrays and intersecting codes. *Journal of combinatorial designs*, 1(1):51–63, 1993.

[89] Emanuel Sperner. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift*, 27(1):544–548, Dec 1928.

[90] M. H. Stone. The theory of representation for boolean algebras. *Transactions of the American Mathematical Society*, 40(1):37–111, 1936.

[91] G. Tassey. *The economic impacts of inadequate infrastructure for software testing.* National Institute of Standards and Technology, 2002.

[92] J. Torres-Jimenez and I. Izquierdo-Marquez. Survey of covering arrays. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 20–27, Sept 2013.

[93] Jose Torres-Jimenez and Eduardo Rodriguez-Tello. New bounds for binary covering arrays using simulated annealing. *Information Sciences*, 185(1):137 – 152, 2012.

[94] Walter D Wallis. *A beginner's guide to graph theory.* Springer Science & Business Media, 2010.

[95] A. W. Williams and R. L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In *Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on*, pages 246–254, Oct 1996.

[96] Alan W. Williams and Robert L. Probert. *Formulation of the Interaction Test Coverage Problem as an Integer Program*, pages 283–298. Springer US, Boston, MA, 2002.

[97] J. Yan and J. Zhang. Backtracking algorithms and search heuristics to generate test suites for combinatorial testing. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, volume 1, pages 385–394, Sept 2006.

[98] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn. Acts: A combinatorial test generation tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 370–375, March 2013.

[99] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn. Acts: A combinatorial test generation tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 370–375, March 2013.

[100] Eric R Ziegel. Experimental design for combinatorial and high throughput materials development. *Technometrics*, 45(4):365–365, 2003.