TU WIEN Informatics

Combinatorial Design Theory and Applications for Software Testing

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

MSc Ludwig Kampel, BSc
Registration Number 0826015

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dr. Dimitris E. Simos

The dissertation has been reviewed by:

Prof. Ph.D. Charles J. Colbourn	Prof. Dr. Ana R. Cavalli

Vienna, 15th August, 2024

Ludwig Kampel

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Combinatorial Design Theory and Applications for Software Testing

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

MSc Ludwig Kampel, BSc
Matrikelnummer 0826015

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Dimitris E. Simos

Diese Dissertation haben begutachtet:

Prof. Ph.D. Charles J. Colbourn	Prof. Dr. Ana R. Cavalli

Wien, 15. August 2024

Ludwig Kampel

Erklärung zur Verfassung der Arbeit

MSc Ludwig Kampel, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. August 2024

Ludwig Kampel

Acknowledgements

First and foremost, I want to express my profound gratitude to my advisor Priv.-Doz. Dr. Dimitris Simos. Further, I want to thank the members of my dissertation committee for accepting the invitation and dedicating their time and expertise. My gratitude goes especially to the international experts for agreeing to review my thesis. Last but not least, I want to thank everybody who co-authored a paper with me in the past years, and I especially want to emphasize my gratitude to the members of the MATRIS research group.

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit dem Zusammenhang von Combinatorial-Design-Theorie und dem Testen von Software. Einerseits wird das Problem des Generierens von Softwaretests für blackbox-testen von Software deren Input vektoriell modelliert werden kann, abstrahiert und als Gegenstand der Combinatorial-Design-Theorie behandelt – auch in einem weiteren Sinn als Gegenstand der diskreten Mathematik und der theoretischen Computerwissenschaft. Andererseits werden in dieser Arbeit Combinatorial Designs zur Lösung von Problemen im Bereich des Softwaretestens angewendet. Der genannte Zusammenhang zwischen Combinatorial-Design-Theorie und dem Testen von Software manifestiert sich gegenwärtig in Combinatorial Testing, einem Zugang zum Softwaretesten, welcher auf der sogenannten "t-way coverage" des Eingaberaumes basiert. Um diesen Zusammenhang auszubauen und zu verstärken, werden in der vorliegenden Arbeit einzelne Schritte des Combinatorial-Testing-Prozesses weiterentwickelt. Entsprechend der beiden Themengebiete und der gewählten Vorgehensweise hat diese Arbeit zwei Abschnitte: Der zweite Teil behandelt die konkrete Anwendung von Combinatorischen Designs für das Testen von Software und softwaregestützen Systemen anhand von Fallstudien und Beispielen aus der Industrie. Der erste Teil betrifft die theoretischen Aspekte, zum Beispiel jene Strukturen der Combinatorial-Design-Theorie, die für das Testen von Software verwendet werden. Es werden deren Eigenschaften und Erzeugung sowie damit einhergehende Probleme untersucht. Abschließend wird erfasst, wie die einzelnen Beiträge den jeweiligen Schritt des Combinatorial-Testing-Prozesses weiterentwickeln.

Abstract

This thesis pertains to the interplay of combinatorial design theory and software testing. On the one hand, the problem of generating test sets for black-box testing of software systems which are modeled via vectorial input can be *abstracted* and treated as subject matter of combinatorial design theory and, in a wider sense, as part of discrete mathematics and theoretical computer science. On the other hand, it is subject to this thesis to *apply* combinatorial designs to solve problems occurring in the realm of software testing. The aforementioned interplay manifests currently in *Combinatorial Testing*, a software testing methodology centered around test sets that achieve full t-way coverage of a software's input space. In order to further and strengthen the interconnections between combinatorial design theory for software testing, we will contribute to and advance individual parts of the *combinatorial testing process*. According to the overall methodology, this thesis is structured in two parts. The first part pertains to the theoretic aspects of this work, such as objects of combinatorial designs, their properties, generation and related problems; the second part comprises concrete applications of combinatorial design theory for testing software and software-aided systems, including dedicated case studies and industrial applications. In the conclusion we capture the individual contributions described throughout this work and outline how they advance the combinatorial testing process, thereby extending the interplay of combinatorial design theory and software testing.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Introduction 1
1.2 Motivation for Combinatorial Design Theory and Software Testing . . 5
1.2.1 Combinatorial Testing: The Early Years 6
1.2.2 Failure Report Analyses - Towards an Empirical Basis for Combinatorial Testing 8
1.2.3 Combinatorial Testing in Brief 10
1.3 Preliminaries: Notions, Notations and Definitions 12
1.3.1 Notions Related to Combinatorial Design Theory 13
1.3.2 Notions Related to Software Testing 18
1.4 State of the Art: An Overview of the Combinatorial Testing Cycle . . 22
1.4.1 Input Model 22
1.4.2 Test Set Generation 24
1.4.3 Combinatorial Fault Localization 27
1.5 Methodology of Combinatorial Design Theory and Software Testing . 30
1.5.1 Structure of the Thesis 31
1.5.2 Publications Underlying the Thesis 32

I Contributions to Combinatorial Design Theory **35**

2 Combinatorial Test Sets as Objects of Theoretical Computer Science and Discrete Mathematics **37**
2.1 Related Literature 38
2.2 Survey of the Computational Complexity of Optimal Covering Array Problems 40
2.2.1 Complexity Problems of CAs and their Relations 43
2.2.2 Complexity Problems of MCAs and their Relations 47

2.2.3 Complexity Problems of VCAs and their Relations 49
2.2.4 Complexity Problems of Bshouty-Seroussi (BS) Arrays: Their Relations and Effects 50
2.2.5 Other Complexity Problems Related to Covering Arrays 56
2.2.6 An Overview of the Current State of the Art 62
2.3 Review and Extension of Problems and Algorithms for Covering Arrays via Set Covers 63
2.3.1 Encoding Covering Array Problems as Set Cover Problems . . 64
2.3.2 A Weighted Density Notion for Covering Arrays 65
2.4 Modeling Optimal Covering Array Generation via Computational Commutative Algebra 68
2.4.1 Algebraic Models for CAs of Arbitrary Strength 69
2.4.2 An Algebraic Model for Column Extensions of CAs 72
2.4.3 Experiments Using Gröbner Bases and Supercomputing 75
2.5 Balanced Covering Arrays: Definition and Classification 78
2.5.1 Definition of Balanced Covering Arrays 78
2.5.2 Related Work 81
2.5.3 Preliminaries 84
2.5.4 Properties for $\boldsymbol{\lambda}$-balance and $\boldsymbol{y}$-balance 89
2.5.5 Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced Covering Arrays 96
2.6 Automata Theoretic Models for Sequential Combinatorial Testing . . . 111
2.6.1 Preliminaries Related to Finite Automata 114
2.6.2 Connection Between Sequential Combinatorial Testing and Known Products of Automata 123
2.6.3 Introducing a Reduced-Shuffle Product 124
2.6.4 A Reduced Shuffle Automaton for Linear Automata 132
2.7 Summary 141

3 Algorithmic Techniques for Covering Array Generation 143
3.1 Related Literature 144
3.2 Artificial Neural Networks for CAs Generation 145
3.2.1 Background to Artificial Neural Networks and Combinatorial Optimization 146
3.2.2 Preliminaries to Artificial Neural Networks and Graphs 147
3.2.3 Boltzmann Neural Networks for Covering Array Generation . . 149
3.2.4 Learning Methods for Boltzmann Neural Networks for Covering Array Generation 155
3.2.5 Hopfield Neural Networks for Covering Arrays 161
3.2.6 Experimental Evaluation 163
3.3 The sliced AETG Algorithm 177
3.3.1 Review of Greedy AETG 177
3.3.2 Memory Saving Variants of Greedy AETG 179
3.3.3 Discussion on Related Data Structures 183

3.3.4 Evaluation . . . 184
3.4 The IPO-Q Algorithm . . . 187
3.4.1 Preliminaries . . . 187
3.4.2 IPO-Q: A Quantum-inspired IPO Algorithm . . . 189
3.4.3 Evaluation of Different Configurations of the IPO-Q Algorithm 191
3.4.4 Comparison with IPOG-F and best known upper Bounds for CAN 193
3.5 Summary . . . 195

II Contributions to Combinatorial Software Testing 197

4 Applications of CT to Software Testing 199
4.1 Related Work . . . 200
4.2 Applying Combinatorial Testing to BACnet Protocol Conformance Testing 202
4.2.1 Introduction and Background to BACnet, Building Automation and the Siemens APOGEE Insight® Product . . . 203
4.2.2 Combinatorial Testing of the BACnet Conformance of the Harmonization Tool of the APOGEE Insight® Workstation . . . 205
4.2.3 Testing Results and Remarks . . . 207
4.3 Applying Combinatorial Testing to Adobe Analytics . . . 208
4.3.1 Introduction and Background to Adobe Analytics . . . 208
4.3.2 IPM Creation and CA Construction . . . 210
4.3.3 Application and Results . . . 213
4.4 Applying Combinatorial Testing to Test Consistency of LLMs . . . 215
4.4.1 Background on Recent Developments of AI and LLMs . . . 216
4.4.2 Related Work on (Consistency) Testing of LLMs . . . 218
4.4.3 Instantiating the Combinatorial Testing Process for Testing the Consistency of LLMs . . . 219
4.4.4 Experimental Evaluation . . . 223
4.4.5 Threats to Validity . . . 226
4.4.6 Future Work and Ideas to Address the Threats to Validity . . . 227
4.5 Summary . . . 228

5 Applications of CT-FLA 231
5.1 Related Work . . . 232
5.2 Fast CT-FLA for Hardware Trojan Location . . . 236
5.2.1 Background on Hardware Testing . . . 236
5.2.2 Related Work . . . 241
5.2.3 Linkage Between Hardware Trojan Location and Combinatorial Testing . . . 244
5.2.4 Location Algorithms . . . 245
5.2.5 Case Study . . . 253
5.2.6 Experiments . . . 254
5.2.7 Threats to Validity . . . 263

5.2.8 Summary and Outlook 264
5.3 Exploring CT-FLA for XAI 265
5.3.1 Background on Explainable Artificial Intelligence 266
5.3.2 Properties of XAI 267
5.3.3 Remarks Related to the Development of XAI 269
5.3.4 CT-FLA for XAI 272
5.3.5 Reflection of CT-FLA Methods for XAI 274
5.4 CT-FLA for Automated Driving Functions 277
5.4.1 Background on Automated and Autonomous Driving Systems . 278
5.4.2 Underlying Works on Automated Driving Functions and Combinatorial Testing 279
5.4.3 Methodology 279
5.4.4 Case Study 282
5.4.5 Results 285
5.4.6 Threats to Validity 288
5.4.7 Review of a Domain Expert 288
5.4.8 Epilogue – Analysis of an Individual Crash Scenario 291
5.5 Summary 293

6 Conclusion and Outlook **295**
6.1 Advancement of the Combinatorial Testing Cycle 295
6.2 Outlook 298

List of Figures **303**

List of Tables **309**

List of Algorithms **317**

Bibliography **319**
Appendix 358
Binary Balanced CAs 358
Ternary Balanced CAs 389

CHAPTER 1

Introduction

1.1 Introduction

Software testing is an integral part of software engineering and contributes to the development of higher quality software artifacts. There are several aspects to software testing, which can play different roles in the software development process and can be understood in different ways. Going along the lines of Ammann and Offutt [1], who adapt from Beizer [2], software testing can be understood as (i) identical with debugging; (ii) showing that software works; (iii) showing that software does not work; (iv) reducing the risk of using some software; and (v) a mental discipline that helps developing higher quality software. For some, the importance of software testing may be self-evident, considering that more and more aspects of contemporary society are affected *by* and our daily lives are increasingly interwoven *with* software. We may be impacted directly, as users or operators, as well as indirectly, since software is deployed in many areas of our lives. We may think, for instance, of traffic control systems that can indirectly affect arbitrary citizens of a town, not only actual traffic participants. Another example would be software used for trading at the stock exchange, whose function or malfunction can have global impacts. For others, the importance of software testing is not as obvious, but can be clarified in different ways. On the one hand, through *infamous software failures* causing tragic accidents that could have been avoided through better testing. Examples are the catastrophic crash of the Ariane 501 satellite launch [3], or the fatal plane crashes of major avionic companies [4], [5]. On the other hand, through *famous quotes* that remind us that software testing does not stand in contrast to, let alone is replaceable by, other methods for increasing software quality such as formal software verification. The probably most famous quote is attributed to Donald Knuth: “Beware of bugs in the above code; I have only proved it correct, not tried it.” [6], [7]. Failing all the above, people might best be convinced of the importance of software testing by illustrating the economical and monetary consequences if no or inadequate testing is done. A prominent

and frequently cited 2002 report [8] of the U.S. National Institute of Standards and Technology (NIST) stresses these consequences, stating that "based on the software developer and user surveys, the national annual costs of an inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion." Similarly, a more recent report [9] is putting the costs of poor software quality into figures, stating that for the year 2022 the costs for finding and fixing bugs amount to $608 billion in the U.S. alone.

Over the past decades, multiple testing methodologies have been developed and are now co-existing, each of them having their own characteristics, advantages and disadvantages. Giving an in-depth overview is not within the scope of this thesis, especially since excellent introductions to methods of software testing exist [1], as well as comprehensive surveys [10]. However, we list a few testing methodologies below; these do not necessarily exclude each other but can often be used in combination:

- Model-based testing [11], [12] where tests are derived from models that are based on system requirements.
- Search-based software testing [13], deploying metaheuristic search techniques for automated and optimized test generation.
- Metamorphic testing [1, Chapter 14.2.4], making use of specific input variations to test against an expected program behavior.
- Mutation testing [14], for generating new tests and assessing the quality of existing tests based on their capability of detecting seeded faults.
- Coverage criteria [1, Chapter 2.4], such as code coverage (e.g., based on functional, statement, modified condition decision coverage), or input-space coverage (e.g., based on parameter value coverage or pairwise coverage).
- V-model for software testing [1, Chapter 2.3], including unit-testing, functional testing, integration testing, system testing and acceptance testing.
- Regression testing [1, Chapter 13] [15], seeking to ensure that evolving software still performs as expected after modification.

Many software artifacts are too complex to be *tested completely*, with the meaning differing from case to case. Typically, the number of inputs and the number of internal states suffer from combinatorial explosion. Therefore, one notorious question of software testing is: *When to stop (or pause) testing?* – a question that researchers and engineers have been dealing with for decades [16], [17]. Although this question cannot be answered in general – and often not even in specific cases – coverage criteria can aid test engineers, as they provide a means to quantify the testing progress – at least with respect to these specified criteria. A prominent input space coverage criterion (or measure) is *pairwise*

coverage [18], which pertains to the occurrence of pairs of input parameter values in the tests of a test set.

Combinatorial design theory can be described as the study of discrete, finite mathematical structures – combinatorial designs – that obey certain intersection-, balance- or coverage properties. We will encounter these three properties at essential points in this thesis. Typically, a combinatorial design can have multiple phenomena – as it may appear as an element of various disciplines of discrete mathematics – such as finite geometry, finite fields – and may be represented as a set-system, or a matrix when considering the incidence structure of the former.

The theory of combinatorial designs is rich and several notions play important roles in other mathematical disciplines such as statistics (in in the *Design of Experiments*), cryptography, or error-correcting codes. There are many important notions of combinatorial designs that can serve as representative examples with notable connections to, e.g., finite geometry, finite fields, or group theory. A representative example would be the Fano plane, which is the smallest finite projective plane, also constituting a Steiner (triple) system $S(2,3,7)$ and being tightly connected to an orthogonal array $\mathsf{OA}_1(2,3,7)$, see also [19]. Another example are *Hadamard matrices* which are tightly connected to error-correcting codes, which in turn are the mathematical structures and methods used to transmit data over noisy channels. Without being aware of it, we are using these methods, e.g., whenever we employ telecommunication. A particular example is the Hadamard matrix H_{12}, tightly connected to the *Golay code* $\mathcal{G}_{24}$ [20, Chap. 2.6] which was used for data transmission in NASA's Voyager missions [21].

However, we try to give a more detailed understanding of combinatorial designs and their interconnections by means of a notion that gained popularity in a surprisingly explicit and pure form: *Latin squares*, which underlie *Sudokus* – a number puzzle that is a popular mental exercise being widely spread in newspapers and on online platforms. Most frequently, Sudokus take the form of 9×9 arrays that are partially filled with the numbers 1 to 9. The task is to fill the missing entries of the array, such that each number appears exactly once per column and once per row. Additionally further constraints can be considered, such as the frequently used condition that in specific 3×3 sub-arrays each symbol has to appear exactly once. As an object of combinatorial design theory, a (solved) Sudoku is nothing else than a 9×9 *Latin square* [19, Part III], with possibly additional constraints concerning the 3×3 sub-arrays. Generally, a Latin square of side n is an $n \times n$ array whose entries arise from an alphabet of size n, where each symbol of the alphabet appears exactly once per column and once per row. An example of a 9×9 Latin square can be found in Figure 1.1. The combinatorial puzzles we call Sudokus are formally known as *partial Latin squares* and ask to be completed to Latin squares. In fact, it was shown in 1984 that deciding whether a given partial Latin square can be completed to a Latin square is an NP-complete problem [22]. In this light, when we are solving Sudokus, we are solving a sub-problem – typically of size 9×9 – of a search problem that in its general form is NP-hard.

Another characteristic of combinatorial designs is their tight interconnection, whether

9		6		7		4		3
			4			2		
	7			2	3		1	
5						1		
	4		2		8		6	
		3						5
	3		7				5	
		7			5			
4		5		1		7		8

(a) Partial Latin square/Sudoku puzzle.

9	2	6	5	7	1	4	8	3
3	5	1	4	8	6	2	7	9
8	7	4	9	2	3	5	1	6
5	8	2	3	6	7	1	9	4
1	4	9	2	5	8	3	6	7
7	6	3	1			8	2	5
2	3	8	7			6	5	1
6	1	7	8	3	5	9	4	2
4	9	5	6	1	2	7	3	8

(b) Almost completed puzzle.

4	9
9	4

9	4
4	9

(c) Two ways for completion.

Figure 1.1: A 9×9 Sudoku puzzle from [23] on the left, and the two (unique) solutions for its completion on the right [23].

```
111111111222222222333333333444444444555555555666666666777777777888888889999999999
123456789123456789123456789123456789123456789123456789123456789123456789123456789
926571483351486279874923516582367194149258367763149825238794651617835942495612738
```

Figure 1.2: The transposed orthogonal array $\mathsf{OA}(81, 3, 9, 2)$ corresponding to the Latin square in Figure 1.1.

directly or by means of combinatorial constructions. For example, when we denote a Latin square as something else than an $n \times n$ array and list its entries as triples (r, c, v) where the first entry specifies a row, the second a column and the third the value found in the Latin square, then we obtain a design called *orthogonal arrays*. In the particular case of a 9×9 Latin square, we obtain an $\mathsf{OA}(81, 3, 9, 2)$. The orthogonal array corresponding to the Latin square given in Figure 1.1 is depicted in Figure 1.2 where we selected the upper way shown in Figure 1.1c to complete the Latin square in Figure 1.1b. In general, an orthogonal array $\mathsf{OA}(N, k, s, t)$ is defined as an $N \times k$ array over an alphabet of size s, where each $t \times N$ sub-array comprised of any t distinct columns has the property that each t-tuple over the alphabet appears equally often, that is $\lambda = N/s^t$ times, see also [24]. The parameter t is called the *strength* of the orthogonal array, and λ its *index*. Orthogonal arrays are also denoted as $\mathsf{OA}_\lambda(N, k, s, t)$.

If we take one further step and weaken the defining property of orthogonal arrays, i.e., that each t-tuple has to appear *exactly once*, and merely impose that they have to appear *at least once*, then we obtain a generalization of orthogonal arrays known as *Covering Array (CA)*, see also [19]. This is the combinatorial design notion that will guide us throughout this thesis, as it its the essential ingredient for devising software tests with the desired coverage of the input space.

This thesis pertains to the interplay of combinatorial design theory and software testing. That is, applications of combinatorial design theory to software testing are subject to

this thesis; and vice versa, challenges arising in software testing will be abstracted and formalized to define objects or characterize problems of combinatorial design theory in order to be further investigated as such. At first sight, these two domains may seem very distinct; however, previous efforts brought to light a remarkable connection between them and showed their mutual enrichment. It is my aim to advance this symbiotic relationship and to further develop combinatorial design theory as an approach for software testing.

Aim of the Thesis. The aim of this thesis is to develop and further combinatorial design theory for software testing. This is done by pursuing two paths. First, we will strengthen the combinatorial design theoretical foundations which underlie combinatorial testing, and second, we will advance applications of combinatorial testing by devising new methods that extend its applicability in known domains as well as open up new application domains.

1.2 Motivation for Combinatorial Design Theory and Software Testing

The interplay between combinatorial design theory and software testing has not been fully examined, but individual connections promise fruitful synergies between them which we extend and broaden in this thesis, as detailed below. Paradigmatic for the mutual enrichment of these two domains is the emergence of *combinatorial software testing – Combinatorial Testing (CT)* for short – that has experienced a dynamic development since the end of the 20^{th} century. A thorough introduction to CT can be found in [25]. In the following, we outline the key developments and aspects.

We have already introduced the concept of *pairwise coverage*, which quantifies over pairs of parameter-values of the System Under Test (SUT)'s input parameters and their appearance in the tests of a test set. Clearly, in order to make use of this coverage criterion an appropriate model of the SUT is required in the first place, i.e., one that models the input to the SUT by means of parameters that can take values from respective finite domains. Such models are known as *Input Parameter Model (IPM)* [26]. We will consider this concept in more detail in the following section. When we apply *pairwise testing*, we are testing an SUT with a test set that achieves full pairwise coverage for a specific IPM, i.e., every possible parameter-value pair of the IPM appears in at least one executed test. Thus, we are testing for *failures* that are triggered by combinations of two parameter values. Test sets that achieve full pairwise coverage are also called *pairwise test sets*.

A natural generalization of pairwise coverage is *combinatorial t-way coverage* (also *t-way coverage* for short) [25, Chapter 7], which pertains to the appearance of parameter-value combinations of up to t input parameters of an SUT, i.e., t-tuples of parameter-values. When we deploy test sets that achieve full combinatorial t-way coverage for some integer t, this is called *combinatorial testing*, i.e., we are testing an SUT with a test set that achieves

full combinatorial t-way coverage for a specific IPM. This further means that every, due to the IPM possible, t-tuple of parameter-values occurs in at least one test. We are testing the SUT for failures that are triggered by combinations of up to t parameter-values. Test sets that achieve full combinatorial t-way coverage are also called *t-way test sets*. There exist terms synonymous with combinatorial testing, such as t-way testing [27] and combinatorial interaction testing [28], or *t-way testing* for short. In its general form, combinatorial testing does not depend on an SUT's internals, but merely depends on the modeled input space, thus it represents a black-box testing methodology that tests against (undesired) interactions of input parameters.

In the following, we will revisit the history and developments towards combinatoral testing as it is known today.

1.2.1 Combinatorial Testing: The Early Years

Identifying the very first applications of t-way test sets for any value of t is difficult, especially since the concept of parameter-value coverage (also known as *each choice* or 1-*wise* [18]) as well as pairwise coverage and pairwise testing were widely known and likely used by many engineers.

Logic Circuit Testing. First applications of test sets with full t-way coverage are documented in scientific publications from the 1980s in the domain of logic circuit testing, where test sets are referred to as *test vectors*. For example, Tang and Woo proposed in their 1983 work [29] to use test vectors which are obtained via a construction that collects all vectors of a certain constant weight in order to achieve full t-way coverage. Similarly, in 1988 Seroussi and Bshouty present [30] a study of theoretical aspects of test vectors with full t-way coverage, again in the realms of logic circuit testing. We will revisit their work in more detail in Section 2.2.

Software Testing. Early investigations and applications of test sets with full t-way coverage in the realms of software testing can be roughly separated into three stages, depending on the combinatorial designs used for deriving the test sets. The first respective publications date back to the 1980s and make use of Latin squares or mutually orthogonal Latin squares (MOLS), i.e., a set of Latin squares where any two distinct Latin squares give rise to yet another Latin square by superposition [19].

MOLS Period - Mutually Orthogonal Latin Squares for Software Testing:

In early applications of t-way test sets in the domain of software testing the t-way test sets were derived from Latin squares or sets of mutually orthogonal Latin squares [19]. For example, in 1985 Mandl [31] proposed to do so for compiler testing. Exemplified by testing four (binary) ordering operators that can be applied to four values, a 4×4 Latin square is used to derive 16 tests with three parameters (or factors), achieving full pairwise coverage of paramter-value assignments. In the same way, Williams [32]

proposed to use Latin squares, respectively sets of mutually orthogonal Latin squares, for testing network interfaces in telecommunication software in 1996. The method of using MOLS for deriving test sets with full pairwise coverage results in v^2 tests in case there are v different values and at most $v + 1$ input parameters in the IPM of the SUT. However, both authors have also acknowledged the limitations of using MOLS, i.e., that the method is only applicable for pairwise testing ($t = 2$), if the number of parameters is at most the number of values increased by one, and, in addition, is primarily suited when all parameters in the IPM of the SUT have the same number of values. This is because there exist at most $v - 1$ MOLS over v symbols, see also [19].

Orthogonal Array Period - A Tool for Orthogonal Array Construction to Derive t-way Test Sets:

It is likely that the shortcomings of the approach using MOLS have driven the investigation of orthogonal arrays (OAs) for software testing. In the early 1990s, articles from researchers of the AT&T Bell Laboratories were published in which OAs are used for designing software and system tests for telecommunication applications. Brownlie, Prowse and Phadke [33] used OAs for robust testing of a new e-mail product. In their paper they report – to the best of my knowledge – for the first time the use of a tool, which for given input parameters and their values (aims to) construct an appropriate OA and automatically generates the test set. As mentioned in [33] and also in [34], the OATS tool (Orthogonal Array Testing System tool) was developed by Chintapalli, Hegde, and Phadke at AT&T Bell Laboratories.

While the notion of OAs also allows to derive t-way test sets for higher values of $t \geq 2$, OAs still obey certain restrictions pertaining to the number of input parameters, domain sizes, etc. In particular, the existence of OAs for arbitrary parameter numbers, domain sizes, and strength t is not given, see [19, Part III, Section 6.2] for a compact overview or [24] for a thorough treatment of OAs and their existence.

Covering Array Period - A Tool for Generating General t-way Test Sets:

Early investigations that consider "general t-way test sets" – i.e., t-way test sets where the theoretical restrictions pertaining to existence inherited from OAs are lifted – are the works of Cohen et al. [35] in 1994 and in 1997 [36]. In their work, they propose the Automatic Efficient Test Generator (AETG), a tool for generating t-way test sets for, theoretically speaking, an arbitrary number of input parameters, each of which can assume an arbitrary number of values. The AETG tool has been developed and deployed at Bellcore [37]. In these works we also find descriptions of early applications of combinatorial t-way testing, where test sets obtained via AETG were deployed, including applications to test a database system at Bellcore [35], and for protocol conformance testing [38] in 1994.

A further pioneering work from Dunietz et al. [39] from 1997 investigates t-way test sets for software testing and evaluates them with regards to two types of code coverage

– branch coverage and structural (or block) coverage – for different values for t. Their empirical results suggest that there is a direct functional dependency between t-way coverage and code coverage. This observation may be effectively used to reduce redundant tests, at least in terms of certain types of code coverage. Further, the test sets with full t-way coverage also improve on random test sets of the same size in terms of code coverage.

To the best of my knowledge, the term *covering array* was introduced by Sloane in his seminal paper *Covering arrays and intersecting codes* in 1993 [40], who motivated the study of covering arrays also by applications in (hardware) testing. A further remarkable work is that of Stevens and Mendesohn from 1998 [41]. They mention the derivation of pairwise test sets from CAs and present new methods for their construction.

1.2.2 Failure Report Analyses - Towards an Empirical Basis for Combinatorial Testing

A sequence of empirical investigations, primarily driven by Kuhn et al. at the turn of the century, paved the way and, at the same time, provides an explanation for the success of testing with t-way test sets. Along with the empirical findings outlined below, the number of applications of t-way test sets in the domain of software testing increased, and the term *combinatorial testing* (CT) was coined [25].

In 1996 researchers at Bellcore (Cohen et al.) reported that an empirical study of user interface software showed that most field faults were caused by either single values or by an interaction of pairs of values [42].

In 2001 Wallace and Kuhn published a report [43] investigating failures of medical devices of various types, recorded over a period of 15 years by the U.S. Food and Drug Administration. A total of 342 records contained enough information about the failures to be considered in their analysis. A number of 109 failure reports contained enough information in order to analyze what conditions are required to trigger a fault and subsequently result in the respective failure of the medical device, while the remaining 233 failure reports did not. Their analysis showed that a large majority (106) of the failures were caused by single value conditions or pairs of values. The remaining three failures required interactions of three or more conditions to cause the failure, and the most complex one was caused by an interaction of four values [43].

Kuhn and Reilly [44] conducted a similar analysis of bug reports of open-source software projects in 2003 – 194 bugs of the Mozilla web browser and 171 bugs of the Apache web server were reviewed. They created a classification of the bugs based on the number of values required to trigger a failure. Their classification of the bugs shows that they are triggered by single values as well as combinations of up to six values, where a majority of failures, over 96%, could be detected by a test set with full 4-way coverage. However, both SUTs (browser and server), had also reported bugs triggered by a combination of 6 values, i.e., a 6-way test set could have revealed all these failures.

Kuhn et al. extended the previously mentioned empirical studies in 2004 to include the analysis of 329 error reports from a distributed scientific database developed at NASA Goddard Space Flight Center [45]. The results show that all investigated failures can be triggered by a 4-way test set, where again most failures could be triggered by 1-way or 2-way combinations of values, and only 2% of failures require a trigger that is a combination of three or four conditions. The results of [45] were augmented by reports about failures of the Remote Agent Experiment software on NASA's Deep Space 1 mission [46] and those of an operating system function [47]. In particular, Kuhn et al. [45] mention that testing all pairs of input values could not detect all failures of these SUTs, leaving open the investigation of higher-degree interactions. If experience shows that all failures of a specific type of software artifacts can be triggered by t or less parameter-value combinations, then t-way testing represents a way for "pseudo-exhaustive" testing. Since the software failures reviewed by Kuhn et al. were triggered by parameter-value combinations of at most four to six parameter values – depending on the software artifacts – they conclude that combinatorial testing with 4-way to 6-way test sets appear to be appropriate to resemble pseudo-exhaustive testing. At the same time, they mention that "many more empirical studies of other classes of software are needed to evaluate the applicability of combinatorial testing for other classes of systems" [45].

The previously formulated hypothesis held true in a 2016 follow-up study [48]. The authors investigated the relation between different bug types and the t-way interactions triggering them for a well documented open-source relational database management system. Of the 242 analyzed bug reports, approximately 75% contained enough information to determine the number of parameter-value assignments required to trigger them. Amongst other results, it was concluded that for these bugs even the most complex ones could have been detected when testing with a 5-way test set.

In addition, research groups other than Kuhn et al. have conducted empirical analyses of software failures and bug reports. For example, the results of Bell's PhD thesis [49] were included by Kuhn et al. [50] in their overall aggregated 2016 analysis and presentation. They showed that a failure analysis of a TCP/IP internet protocol implementation also fits the overall picture.

Fögen and Lichter present a case study analyzing 434 bug reports which describe failures of a financial enterprise application. Although they identified 38 bugs that required specific timing or orderings to be triggered, 176 bugs that were integration failures triggered by *any* test input, and 8 bug reports did not contain enough information for their analysis; the remaining 212 bug reports were suitable for analyzing the interacting parameter-values required to trigger the respective failure. The result of this analysis fits again into the overall picture outlined by the studies of Kuhn et al. The majority of failures are triggered by a single parameter value or by a pair of values, progressively fewer failures are triggered by an interaction of three or four parameter-values, and none of the reported failures required an interaction of more than four parameter-values [51].

In another remarkable work, Cotroneo et al. pose the question *how bugs surface* and investigate bug manifestation and its characteristics. They conduct an empirical study

analyzing 666 bug reports of two applications: the MySQL open-source relational database management system and the Apache open-source web server [52]. Their models of the subject SUTs include – in addition to the respective application – external entities such as the user and the execution environment. Consequently, they consider different types of failure triggers from a higher level and define *macro-conditions* that may influence bug manifestation, such as: workload, execution environment, the application's initial states, or the user behavior. Notably, Cotroneo et al. consider failure triggers other than pure combinations of input parameters, for example sequences of inputs which, in form of request sequences, are especially relevant for failures of MySQL. Part of their results pertain to the number of triggers required to expose a bug [52, Fig. 9]. Their results show that combinations of macro-conditions are necessary to reveal most of the failures. For example, their *Finding #16* states that 45.7% of the bugs require two macro-conditions to surface, and in 37.9% of the cases two workload conditions are required. Furthermore, *Finding #18* states that 7.6% of the bugs require 4 macro-conditions in combination in order to surface, while 1% require a combination of 5 macro-conditions [52].

The results of Cotroneo et al. show again that, despite being a potential additional abstraction step, combinations of (macro-)conditions are critical for triggering the analyzed bugs. Their results were included in the – at the time of writing – most recent report of Kuhn et al. [53], summarizing the analysis of empirical data regarding the relationship between software failures and the number of conditions required to be combined in order to trigger them. To summarize, the aggregated empirical data shows that most failures are triggered by a single input parameter value or by pairs of parameter values. Gradually fewer failures are triggered by a combination of three or more parameter values; however, none of the investigated failures required a combination of more than six parameter values. Kuhn et al. refer to this circumstance as the "*interaction rule*":

> "**Interaction Rule:** Most failures are induced by single factor faults or by the joint combinatorial effect (interaction) of two factors, with progressively fewer failures induced by interactions between three or more factors." [25]

This empirically discovered rule provides the foundation for combinatorial testing. Noteworthy is that this finding is closely related to the *effect sparsity principle*, a foundational principle appearing in Design of Experiments, which says that "*the number of relatively important factors in a factorial experiment is small*" [54]. For an overview of design and analysis of experiments, we refer the interested reader to [55]. Although we cannot expect that the interaction rule holds for all SUTs, we may hypothesize that it generalizes also to other systems to be tested, being well aware that this "interaction hypothesis" may have to be rethought, reformulated or adapted to new findings. However, such methodological processes are anyways integral parts of both, science and engineering.

1.2.3 Combinatorial Testing in Brief

Combinatorial testing is frequently used as an umbrella term for any (software) testing activity that achieves full t-way coverage of the modeled input space of an SUT. In

addition to achieving full t-way coverage, it is characterized by the aim of doing so with the minimal number of tests, i.e., the smallest test set. In short, we can describe CT as testing with optimized test sets that achieve full t-way coverage. The notion of t-way coverage is left ambiguous on purpose, as depending, e.g., on the specific use case and the SUT it can pertain to combinations or sequences of inputs. In this way, the above description may be understood as a characterization of *combinatorial testing in a wider sense*. This includes testing with (optimized) t-way test sets coming from CAs and their generalizations, related structures, or any actions geared towards combinatorial fault localization, i.e., the identification of failure-triggering parts of the input, which we further detail below. We may understand CT in a wider sense as the application of combinatorial designs and accompanying methods for (software) testing. In this thesis we consider primarily *combinational combinatorial testing*, thereby referring to applications of CT to SUTs which are modeled by taking vectorial input. In such cases the notion of t-way coverage refers to combinatorial t-way coverage [25, Chapter 7], as mentioned previously. Further, we use the term *Combinatorial Testing Fault Localization (CT-FLA)* to refer to combinational combinatorial testing augmented by methods for identifying failure-triggering parts in the input vectors, i.e., those combinations of parameter-value assignments that are responsible for triggering failures. With *sequential combinatorial testing* we refer to applications of CT to SUTs which are modeled as taking sequential input. This is of subordinate importance to this thesis, being subject only to Section 2.6. *Throughout this thesis, if not closer specified, we use the term* ***combinatorial testing*** *(CT) to refer to combinational combinatorial testing. If referring to* ***sequential combinatorial testing****, we will explicitly write* ***sequential CT****.*

The notion of *covering arrays* [19] of strength t is of central importance for CT as it allows to derive software tests that achieve full combinatorial t-way coverage.

To apply combinatorial testing based on CAs, it is required to model the input space of an SUT by means of an input parameter model (IPM) [26]. This can be done by identifying parameters, also called factors, and the respective values these parameters can attain. The resulting IPM is a discrete model of the SUTs input space and allows to express an input to the SUT as a vector representing parameter-value assignments. On the other hand, CAs are structures appearing in combinatorial design theory, mostly represented as matrices whose rows have certain properties regarding the appearance of tuples in subarrays. More precisely, CAs of *strength* t over a given alphabet have the defining characteristic that any sub-array comprised of any t distinct columns has the following property: every t-tuple over the alphabet appears at least once as a row of the sub-array. Hence, when using the rows of a CA to construct a combinatorial test set for an SUT, the strength t of the CA translates to the combinatorial t-way coverage of the test set. In other words, for any combination of t parameters modeling the SUT, every possible input assignment to these parameters is tested once the whole test set based on the CA is executed.

The properties of CAs, together with the empirical findings of Kuhn et al., provide a possible explanation for the success of CT in practice [28], [56], [57]. Promising to

constitute pseudo-exhaustive test suites that are generic enough to be applied for wide range of software systems, CAs have attracted the attention of many researchers, both theorists and practitioners. However, the interplay between combinatorial design theory and software testing is not limited to practical applications of CAs in software testing. The enrichment between the two fields is mutual since further applications of CT for software testing have led to additional demands for combinatorial designs. Subsequently, a vital research area and numerous new notions of combinatorial designs were established.

Especially the construction of CAs with a small number of rows – which translates to small test suites – turned out to be a very challenging and computationally difficult problem which is subject to ongoing research as of this writing. Significant effort has also been made in the development of combinatorial methods for testing (software) systems that have sequential dependencies on their inputs, i.e., where tests are modeled as an ordered sequence of events. In any case, the hurdles when deploying combinatorial testing are diverse, making several requirements that the rather "rigid" structure of CAs cannot fulfil. This led to generalizations of CAs, i.e., new combinatorial design structures such as

1. *Mixed-level covering arrays* for heterogeneous alphabet sizes in the IPM [58],
2. *Variable strength covering arrays* that can model different interaction strengths between groups of input parameters [59],
3. *Constrained covering arrays* when constraints in the input space are present [60],
4. *ℓ-biased covering arrays* aiming to maximize failure-per-test ratio on earlier scheduled tests [61],
5. *Locating arrays* and *detecting arrays* for combinatorial test sets for fault localization [62],
6. *Sequence covering arrays* for testing SUTs with sequential dependencies in the input [63].

Once they are generated, such structures can be embedded in the software testing process, using them to derive software tests [64]. At the same time, this raises the problem of generating these structures [65]. Throughout this thesis, we will meet several of the above-listed structures: mixed-level covering arrays are applied in Section 4.4, variable strength covering arrays are generated in Section 2.3, and detecting arrays are applied in Section 5.2.

1.3 Preliminaries: Notions, Notations and Definitions

In the following, we present definitions, notions, notations and terminologies used throughout this thesis. First, those pertaining to combinatorial design theory, mainly covering arrays and their generalizations. Second, those pertaining to software testing. At the end of this subsection we also include a list of the acronyms used.

1.3.1 Notions Related to Combinatorial Design Theory

We begin with the notations of some elementary concepts. We frequently use the abbreviation $[v]$ for the integer interval $\{0, 1, \ldots, v-1\}$, for some $v \in \mathbb{N}$. These integer intervals will mostly appear as alphabets underlying the columns of the considered arrays. The rows of the considered arrays again will be indexed starting at $1 \in \mathbb{N}$, mostly by $\{1, \ldots, N\}$, for some $N \in \mathbb{N}$, while columns are mostly indexed by $\{1, \ldots, k\}$ for $k \in \mathbb{N}$. By $\binom{k}{t}$ we denote the binomial coefficient and, we use the notation $\binom{\{k\}}{t}$ to denote the set of all subsets of $\{1, \ldots, k\}$ having cardinality t. In case an array with k columns is given, these can be mapped to selections of t columns of the array. For two sets A, B we denote their Cartesian product as $A \times B := \{(a, b) : a \in A, b \in B\}$, for sets $A_1, \ldots, A_n$ we denote their Cartesian product as $\bigtimes_{i=1}^{n} A_i$, and we use the power notation to denote n-tuples over a set: $A^n := \{(a_1, \ldots, a_n) : \forall i \in \{1, \ldots, n\}, a_i \in A\}$. Further, we denote the power set of a set A, i.e., the set of all subsets of A as $\mathscr{P}(A)$.

The definitions and notations below follow the publications underlying this thesis, primarily those of [66], [67], and [68]. CAs are combinatorial objects most frequently introduced as follows, see also [19]:

Definition 1.3.1 (CA) *A* covering array $\mathsf{CA}(N; t, k, v)$ *is an* $N \times k$ *array over a* v*-ary alphabet* Σ *that has the property that in any* $N \times t$ *sub-array, comprised of* t *different columns of the covering array, every* t*-tuple in* Σ^t *appears at least once as a row. The parameter* t *is called the* strength *of the covering array.*

Although CAs can be considered over arbitrary alphabets, we restrict our attention mostly to arrays over integer intervals $\Sigma = [v]$. We can do so without loss of generality since the properties of CAs we consider here merely depend on the size of the alphabet rather than its actual elements. We already make use of this simplification in the definition of Mixed-level Covering Arrays (MCAs) shown below, following the one given in [19], but phrasing it slightly different. MCAs constitute a natural generalization of CAs where the i-th column can take v_i different values.

Definition 1.3.2 (MCA) *A* mixed-level covering array $\mathsf{MCA}(N; t, k, (v_1, v_2, \ldots, v_k))$ *is an* $N \times k$ *array* $(\mathbf{c}_1, \ldots, \mathbf{c}_k)$ *with the following properties:*

(i) For all $j \in [k]$ *the values in the* j*-th column* $\mathbf{c}_j$ *belong to the set* $\{0, \ldots, v_j - 1\}$

(ii) For each selection $\{\mathbf{c}_{j_1}, \ldots, \mathbf{c}_{j_t}\} \subseteq \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ *of* t *different columns, the subarray that is comprised by the columns* $\mathbf{c}_{j_1}, \ldots, \mathbf{c}_{j_t}$*, has the property that every* t*-tuple in* $\bigtimes_{r=1}^{t} \{0, \ldots, v_{j_r} - 1\}$ *appears at least once as a row.*

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

We say that an $N \times k$ array $A = (\mathbf{c}_1, \ldots, \mathbf{c}_k)$ having property (i) of Definition 1.3.2 is an array *over* $(v_1, \ldots, v_k)$ for short.

The values t, k and v are also referred to as the CA parameters, analogue $N, t, k, (v_1, \ldots, v_k)$ are the parameters of the MCA. When the parameters $(N, t, k, (v_1, \ldots, v_k))$ and $(N; t, k, v)$ play no role in the specific context, and there is no threat for ambiguity, we will simply refer to these structures with the abbreviations MCA and CA, respectively. When we want to construct a $\mathsf{CA}(N; t, k, v)$ of arbitrary N for given t, k, v, then we also say that we are given a *CA instance*. For both, CAs and MCAs, the parameter t is referred to as *strength*. The number of rows N is often referred to as the *size* of a $\mathsf{CA}(N; t, k, v)$, which must not be confused with its number of columns k.

From the definition, it is clear that for $t' \leq t$ a CA of strength t is also a CA of strength t'. Thus, a CA can have several strengths, respectively we can consider a CA for different strengths. If we want to have a well-defined notion of *the strength* of a CA, then we have to consider its maximal strength: Given a $\mathsf{CA}(N; 2, k, v)$, say A, there is a t with $2 \leq t \leq k$ such that A is a CA of strength t but not a CA of strength $t + 1$, i.e., $t \in \mathbb{N}$ is maximal with the attribute *strength of* A. In this case the phrasing "t is *the* strength of A" would be justified. However, we may use the informal formulations such as "the strength t of a CA" even in cases where we do not know that the CA is not of strength $t + 1$. The same holds for MCAs.

To avoid dealing with trivial cases, throughout this thesis we consider merely CAs of strength $t \geq 2$, and over alphabets of size at least two, i.e., $v \geq 2$ for CAs and respectively $v_i \geq 2$ for all $i = 1, \ldots, k$ for MCAs.

Example 1.3.3 *An example of a* $\mathsf{CA}(12; 3, 11, 2)$ *is given in Table 1.1. As it is a CA of strength* 3*, it has the property that when selecting any three different columns* b_i*,* b_j*,* b_k *with pairwise different* $i, j, k \in \{1, \ldots, 11\}$*, every binary* 3*-tuple (i.e., binary word of length three) appears at least once in the array* (b_i, b_j, b_k) *comprised of these three columns. Further, we know (e.g., from [69]) that there exists no array with less rows that also has this property, i.e., there exists no* $\mathsf{CA}(11; 3, 11, 2)$*.*

Definition 1.3.4 (CAN) *The smallest positive integer* N *for which a* $\mathsf{CA}(N; t, k, v)$ *exists is denoted as* $\mathsf{CAN}(t, k, v) := \min\{N \in \mathbb{N} : \exists \mathsf{CA}(N; t, k, v)\}$*, and is called the* covering array number *(CAN) for* (t, k, v)*.*

A $\mathsf{CA}(N; t, k, v)$ attaining the smallest possible number of rows, i.e., a $\mathsf{CA}(N; t, k, v)$ with $N = \mathsf{CAN}(t, k, v)$, is called *optimal*. In fact, the CAN is well defined and finite for all $2 \leq t \leq k$ and alphabet sizes $2 \leq v$, since the full product $[v]^k$ always constitutes a CA for the respective parameters. Some interesting problems immediately arise from these definitions:

1. Given CA parameters (t, k, v), determine $\mathsf{CAN}(t, k, v)$.
2. Determine an optimal $\mathsf{CA}(N; t, k, v)$.

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}
1	0	1	0	0	1	0	1	1	0	1
1	1	0	0	1	0	1	1	0	0	1
1	0	0	0	1	1	1	0	1	1	0
0	1	1	0	0	0	1	0	1	1	1
1	1	1	1	0	1	1	0	0	0	0
1	0	1	1	1	0	0	0	0	1	1
1	1	0	1	0	0	0	1	1	1	0
0	0	1	1	1	0	1	1	1	0	0
0	1	1	0	1	1	0	1	0	1	0
0	1	0	1	1	1	0	0	1	0	1
0	0	0	1	0	1	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0

Table 1.1: An example of an (optimal) $\mathsf{CA}(12; 3, 11, 2)$

.

We will take a closer look at these as well as related problems in Section 2.2 by formulating them as computational complexity problems.

In the context of column extension algorithms for CA generation (as will be the case in Section 2.5), it appears more natural to consider the maximum number of columns k for a given number of rows N for which a $\mathsf{CA}(N; t, k, v)$ exists.

Definition 1.3.5 (CAK) *The maximum number of columns k for which a $\mathsf{CA}(N; t, k, v)$ exists is denoted as $\mathsf{CAK}(N; t, v) := max\{k : \exists \mathsf{CA}(N; t, k, v)\}$ and abbreviated referred to as* CAK.

Clearly, the CAN and CAK are connected, as $\mathsf{CAN}(t, k, v) = \min\{N : \mathsf{CAK}(N; t, v) \geq k\}$ and $\mathsf{CAK}(N; t, v) = \max\{k : \mathsf{CAN}(t, k, v) \leq N\}$, see also [70].

Definition 1.3.6 *The smallest positive integer N for which an $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ exists is denoted as $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k)) := \min\{N \in \mathbb{N} : \exists \mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))\}$, and is called the* mixed covering array number *(MCAN) for $(t, k, (v_1, \ldots, v_k))$.*

An $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$ with $N = \mathsf{MCAN}(t, k, (v_1, \ldots, v_k))$ is called *optimal*.

The following notion formalizes the concept of t-tuples that are covered by the rows of CAs, it can be also found in [71].

Definition 1.3.7 (t-way interaction) *For a given alphabet size v, we define a v-ary* t-way interaction of k parameters *as a set of pairs $\tau = \{(p_1, u_1), \ldots, (p_t, u_t)\}$ with $1 \leq p_1 < p_2 < \ldots p_t \leq k$ and $u_i \in [v]$, $\forall i = 1, \ldots, t$. Usually, the underlying alphabet and the value of k are clear from the context and we speak of* t-way interactions *for short.*

We refer to t also as the length *of the t-way interaction. The set of all t-way interactions is denoted by $\mathbb{T}_{v,k,t}$.*

We further define a notion of *at-most t-way interactions*, a notion also considered in [62].

Definition 1.3.8 ($\leq t$-way interaction) *For a given alphabet size v, a* v-ary $\leq t$-way interaction of k parameters*, or $\leq t$-way interaction for short, is a set of $s \leq t$ pairs $\{(p_1, v_1), \ldots, (p_s, v_s)\}$ with the property that $v_i \in \{0, \ldots, v-1\}$, $\forall i \in \{1, \ldots, s\}$ and $1 \leq p_1 < \ldots < p_s \leq k$.*

Example 1.3.3 (continuing from p. 14) *The pairs that constitute a t-way interaction represent the parameter-value assignments that are fundamental for combinatorial testing. To give examples, for $k = 11$ and $v = 2$ a 3-way interaction would be $\{(3,1),(4,0),(11,1)\}$ and a ≤ 3-way interaction could be the 2-way interaction $\{(2,1),(5,1)\}$.*

For MCAs we use the analogue notation for heterogeneous alphabets.

Definition 1.3.9 *For positive integers t, k and $v_1, \ldots, v_k$ with $t \leq k$, we define a* $(v_1, \ldots, v_k)$-ary t-way interaction *as a set of pairs $\tau = \{(p_1, u_1), \ldots, (p_t, u_t)\}$ with $1 \leq p_1 < p_2 < \ldots p_t \leq k$ and $u_i \in [v_i]$, $\forall i = 1, \ldots, t$. For the sake of brevity, we also use the notation $\mathbf{v}$-ary t-tuple for a vector $\mathbf{v} = (v_1, \ldots, v_k)$. The set of all $\mathbf{v}$-ary t-way interactions is denoted by $\mathbb{T}_{\mathbf{v},k,t}$.*

We should not confuse the notion of $\mathbf{v}$-ary t-tuples, with v-ary k-tuples, where the latter simply means an element of $\{0, \ldots, v-1\}^k$. We can visualize a $(v_1, \ldots, v_k)$-ary t-tuple as a vector of length k with only t entries specified at positions p_i for $i = 1, \ldots, t$, with values chosen from their specific alphabets. For example, we can illustrate the $(3,2,2,2)$-ary 2-way interaction $\{(1,2),(3,1)\}$ as $(2,-,1.-)$, where "$-$" represents an undefined entry, see also Example 2.3.2. We use $(v_1, \ldots, v_k)$-ary t-tuples to encode a column selection $(p_1, \ldots, p_t)$ of a vector of length k, together with a t-way interactions $(x_1, \ldots, x_t)$ encoding the entries in these positions. This also motivates the following definition:

Definition 1.3.10 *For positive integers t, k and $v_1, \ldots, v_k$ with $t \leq k$, a (row) vector $w \in \bigtimes_{i=1}^{k} [v_i]$* covers *a $(v_1, \ldots, v_k)$-ary t-tuple $\{(p_1, u_1), \ldots, (p_t, u_t)\}$, if the entries of w in positions p_i equal u_i for all $i = 1, \ldots, t$: $w_{(p_i)} = u_i$, $\forall i \in \{1, \ldots, t\}$.*

We further say an $N \times k$ array *covers* a t-way interaction $\{(p_1, u_1), \ldots, (p_t, u_t)\}$, if one of its rows has the entry u_i in column p_i for all $i \in \{1, \ldots, t\}$. With this notion CAs can be characterized as arrays whose rows cover all t-way interactions for given t, k and v.

Example 1.3.3 (continuing from p. 14) *The 3-way interaction $\{(3,1),(4,0),(11,1)\}$ is covered by the $\mathsf{CA}(12; 3, 11, 2)$ in Table 1.1, in its first and fourth row.*

Variable strength covering arrays have been introduced in the literature in different ways, see [72, 73, 59]. They can be considered a generalization of MCAs which allow for formulation of more general coverage properties. We deviate from the terminology used in [59] and consider also mixed alphabet sizes as is the case in [73].

Definition 1.3.11 (VCA) *For $\mathcal{I} \subseteq \mathscr{P}(\{1,\dots,k\})$ with $\bigcup \mathcal{I} = \{1,\dots,k\}$, a* variable strength covering array (VCA) *is an $N \times k$ array $(\mathbf{c}_1,\dots,\mathbf{c}_k)$, denoted as $\mathsf{VCA}(N;\mathcal{I},k,(v_1,\dots,v_k))$, with the properties*

(i) For all $j \in [k]$ the values in the j-th column $\mathbf{c}_j$ belong to the set $\{0,\dots,v_j-1\}$

(ii) $\forall I \in \mathcal{I}$ the array comprised of the columns $(\mathbf{c}_i)_{i\in I}$ has the property that every $|I|$-tuple in $\prod_{i\in I}\{0,\dots,v_i-1\}$ appears at least once as a row.

We also refer to $\mathcal{I}$ as the set of column selections *with regards to a VCA.*

Definition 1.3.12 (VCAN) *The smallest N for which a $\mathsf{VCA}(N;\mathcal{I},k,(v_1,\dots,v_k))$ exists is denoted as $\mathsf{VCAN}(\mathcal{I},k,(v_1,\dots,v_k)) := \min\{N \in \mathbb{N} : \exists \mathsf{VCA}(N;\mathcal{I},k,(v_1,\dots,v_k))\}$, and is called the* variable strength covering array number (VCAN) *for $(\mathcal{I},k,(v_1,\dots,v_k))$. VCAs achieving this bound are also called* optimal.

In the following we introduce locating arrays (LAs) and detecting arrays (DAs) analogue to [62] and [74]. For this definition we require the following notation. For an array A and a t-way interaction τ, we denote by $\bar{\rho}(A,\tau)$ the *set of rows of A that cover τ*, and by $\rho_A(\tau)$ we denote the *number of rows of A that cover τ* (which will be used in Section 2.5). Further, for a set of t-way interactions $\mathcal{T} \subseteq \mathbb{T}_{v,k,t}$, we define $\bar{\rho}(A,\mathcal{T}) := \bigcup_{\tau\in\mathcal{T}} \bar{\rho}(A,\tau)$.

Definition 1.3.13 (LA) *For an integer $d \in \mathbb{N}$ a $\mathsf{CA}(N;t,k,v)$, say A, is a* (d,t)-locating array*, if and only if, any two d-sets $\mathcal{T}_1 \neq \mathcal{T}_2$ of t-way interactions can be distinguished by means of A through the sets of rows their elements are covered in, i.e.*

$$\forall \mathcal{T}_1,\mathcal{T}_2 \subseteq \mathbb{T}_{v,k,t} \text{ with } |\mathcal{T}_1| = |\mathcal{T}_2| = d : \mathcal{T}_1 \neq \mathcal{T}_2 \Rightarrow \bar{\rho}(A,\mathcal{T}_1) \neq \bar{\rho}(A,\mathcal{T}_2).$$

Definition 1.3.14 (DA) *For an integer $d \in \mathbb{N}$ a $\mathsf{CA}(N;t,k,v)$, say A, is a* (d,t)-detecting array*, if and only if, for any d-set $\mathcal{T} \subseteq \mathbb{T}_{v,k,t}$ of t-way interactions, and any t-way interaction τ not in $\mathcal{T}$, there exists a row that covers τ, but no element of $\mathcal{T}$, i.e.*

$$\forall \mathcal{T} \subseteq \mathbb{T}_{v,k,t} \text{ with } |\mathcal{T}| = d, \forall \tau \in \mathbb{T}_{v,k,t} \setminus \mathcal{T} : \bar{\rho}(A,\tau) \not\subseteq \bar{\rho}(A,\mathcal{T})$$

The following result was presented in [62, Theorem 8.5]:

$$A \ \mathsf{CA}(N;t+d,k,v) \text{ with } d < v \text{ is a } (d,t)\text{-detecting array.} \tag{1.1}$$

Thus, the $\mathsf{CA}(12; 3, 11, 2)$ in Table 1.1 represents an example of a $(1, 2)$-detecting array.

Lastly, with regards to sequential CT we introduce the notion of *Sequence Covering Arrays (SCAs)*, analogue to [63] and [75].

Definition 1.3.15 (SCA) *A sequence covering array* $\mathsf{SCA}(N; t, v)$ *is an* $N \times v$ *array with entries from a set* Σ *of cardinality* v*, where each row is a permutation of the elements in* Σ*, such that each permutation of any* t *elements of* Σ *is covered by at least one row of the array.*

1.3.2 Notions Related to Software Testing

Similar introductions to the following notions can be found in [25] or [76].

System Under Test (SUT). A (software) systems which is subject to application of CT is referred to as *system under test* and abbreviated as SUT. In order to apply the combinational CT methods discussed in this thesis, an SUT must fulfill several requirements. First and foremost, the SUT must have an observable behavior, and we assume that it is deterministic, in particular, for identical inputs the SUT returns identical outputs. While the latter may be restrictive in practice, on the other hand, we generally do not require any knowledge about the SUT's internals for the application of CT. That is, the SUT can be considered as a *deterministic black box*.

Input Parameter Model (IPM). Another prerequisite for the application of CT is the availability of an input parameter model of the SUT. The IPM can model the input space or the configuration space of the SUT [25, 1.2]. More precisely, we assume that the SUT is modeled by means of an IPM, as it is considered in [26]. In its simplest form, an IPM enumerates parameters (also called factors [25] or categories [77]) and their respective domains, i.e., the values they can assume. Each parameter value represents a non-empty partition of the input space. Certain combinations of paramter-value assignments may be invalid. In order to avoid invalid test cases it can be obligatory to formulate constraints between the input parameters and their values. Such restrictions can be expressed, e.g., by explicitly listing invalid combinations of parameter-value assignments or, more generally, by means of constraints expressed in some logic. For example, this can be a grounded quantifier free expression in first order logic, in a language with relations such as $=, \leq, >, \neq$, etc. In other words, constraints are formulated using logic connectives, connecting relations of multi-valued variables (e.g., $p_1 = 3 \rightarrow (p_2 > 1 \vee p_4 = p_5)$). An example of how constraints can be specified is given in [78, Grammar 1]. The resulting restriction of the input space renders it a subset of the plain product space of the parameter domains. Some applications allow a further differentiation in *soft* and *hard* constraints, respectively representing constraints that *can be satisfied* and *have to be satisfied* by a test case in order to be successfully executed. The assumption of a finite number of parameters seldomly appears limiting, as many systems may be modeled naturally this way. However, in practical applications we

may very well encounter parameters with (theoretically) infinite domains. For example, parameters modeling time, length or speed: In such cases *equivalent class partitioning* [79] or boundary value analysis [80] can be deployed in order to discretize the domain of the parameters. Consequentially, more formally, we may denote an IPM consisting of k parameters as a tuple $(D_1, D_2, \ldots, D_k, \mathscr{C})$ where D_i denotes the finite domain of the i-th parameter and $\mathscr{C}$ is a set of constraints.

As mentioned above, according to [25, 1.2] there are two basic approaches to conduct CT: (i) considering combinations of *configuration parameters* values or (ii) combinations of *input parameter* values. In the interest of conciseness, we may consider the configuration space of an SUT also as an input space, justified by making one additional step of abstraction and assuming an SUT which has input parameters that each specify a configuration option of the former SUT. Therefore, we also use the term IPM when referring to models of the configuration space of SUTs.

Test and Test Set. A *test* for an SUT and a given IPM is an assignment of values to each parameter in the IPM, coming from its respective domain. Formally, for an IPM $(D_1, D_2, \ldots, D_k, \mathscr{C})$ a test u is represented as a vector or row $u = (u_1, \ldots, u_n) \in \bigtimes_{i=1}^{k} D_i$. Therefore, we also adopt the terminology for rows and say that a test *covers* a t-way interaction $\{(p_1, u_1), \ldots, (p_t, u_t)\}$ when the value u_i appears in position p_i of the test. In case of given constraints ($\mathscr{C} \neq \emptyset$) we may distinguish a *valid test*, which respects all constraints in $\mathscr{C}$, from an *invalid test* which violates at least one constraint in $\mathscr{C}$. As explained above, the tests considered in this thesis represent *abstract tests* if not closer specified. Such a test most likely requires additional pre-processing in order to yield an *executable test*, i.e., to be readily executed against the SUT. Executable tests are always specific to the concrete SUT and the case study. We will encounter them in the second part of the thesis. The first part of this thesis studies abstract combinatorial test suites as objects of combinatorial design theory.

A *test set* contains several tests. Although it is called "set", it typically is considered to be ordered, i.e., actually a list of tests. As such, a test set is frequently represented as a vertical juxtaposition of tests, resulting in an array of tests. In the context of this thesis, test sets will arise from combinatorial designs represented as arrays.

Failures and Faults. In our understanding of the concepts of failures and faults of an SUT we follow Ammann and Offutt [1]: A *failure* is any incorrect behavior of the SUT with respect to the requirements or another description of the expected behavior, see [1, Definition 1.3], while a *fault* is a static defect in the software [1, Definition 1.1]. Since CT is a black-box testing methodology, we generally can recognize failures only through the SUT's output.

Testing Oracle (Oracle). A *testing oracle* – or *oracle* for short – is a method through which the tester can recognize failure, i.e., deviation from the intended behaviour, of the SUT for a given input. We can formally define an oracle as a function o that maps

from the test space to the set $\{pass, fail\}$, in particular it *assigns* to each test either *pass* or $fail$. While conceptually simple, practical realizations of testing oracles can be very cumbersome. Barr et al. dedicated a survey to this topic and discuss *the oracle problem in software testing*, i.e., "the challenge of distinguishing the (...) desired, correct behaviour from potentially incorrect behavior" [81] of an SUT for a given input. However, in this thesis we do not address this problem in general, but only describe oracles for the particular application considered in the respective section.

Oracles may be realized, for example, via a (certified) reference implementation or a second, independent implementation of the same functionality of the SUT (also called *pseudo-oracle* [81]). Another option is to use, if possible, *implicit oracles* [81] and observe the execution environment for undesired anomalies, such as buffer overflows or increased energy consumption. Finally, an oracle can be based on human inspection, although this does not align with test automation.

Passing and Failing Fest. Given an SUT with an IPM, a tester can use an oracle to distinguish *passing tests* from *failing tests*. If the oracle recognizes a failure of the SUT for a specific test, we also say the test *triggers a failure* of the SUT, and we call the test a *failing test*. On the other hand, if the oracle does not recognize a failure of the SUT for a specific test, i.e., the outcome of the test is in accordance with the intended behavior, then the test is called a *passing test*.

Failure-Inducing t-way Interactions. Finally, we introduce a notion of failure-inducing tuples which arise in CT and can, thus, be considered a notion appearing in the intersection of the domains of software testing and combinatorial design theory.

Witnessing a failure of the SUT by means of the oracle and a failing test, we have revealed the *existence* of a fault in the SUT – also according to Offutt and Ammann [1]. As mentioned in [76] as well, this is an important goal of software testing. A further step towards correcting the fault is to isolate or characterize it more closely. In the black-box setting of combinatorial testing we aim to isolate a parameter-value combination of the failing test that is leading to the failure. More precisely, we want to identify a t-way interaction covered by the failing test which will always lead to a failure of the SUT when executing a test that covers the t-way interaction.

Assume we are given an SUT with an IPM consisting of k parameters and a failing test $f = (f_1, \ldots, f_k)$. In a straightforward manner, we can consider f a k-way interaction that triggers the failure. Aiming to characterize the underlying fault more closely, we are interested in finding a *shorter* t-interaction covered by f that will also result in a failure, regardless of how the remaining parameters that are not involved in the interaction are specified. In other words, interpreting the failing test f as a k-way interaction (see Definition 1.3.7), in CT-FLA, we aim to find a t-way interaction $\tau \subseteq f$, minimizing the length of τ while maintaining the property that any test that covers τ is a failing test. We use the term *failure inducing tuple (FIT)* or, more formally, *failure inducing t-way interaction of length t (t-FIT)* when we emphasize the length t of an FIT for such

sub-combinations of input parameter-value assignments. We formally define them as follows.

Definition 1.3.16 *For a given SUT, an IPM with domain sizes $v_1, \ldots, v_k$ and an oracle o, a* failure-inducing t-way interaction (t-FIT) *is a t-way interaction τ that induces the failure of any test that covers τ. More formally, a t-way interaction $\tau \in \mathbb{T}_{\mathbf{v},k,t}$ is a t-FIT, if and only if*

$$\forall f \in \bigtimes_{i=1}^{k} [v_i] : \tau \subseteq f \Rightarrow o(f) = fail.$$

Further, a t-FIT τ is called minimal *when it does not contain a FIT of smaller length. Formally, $\tau \in \mathbb{T}_{\mathbf{v},k,t}$ is a minimal FIT if and only if, it is a FIT and*

$$\forall \tau' \in \bigcup_{i=0}^{|\tau|} \mathbb{T}_{\mathbf{v},k,i} : \tau' \subsetneq \tau \Rightarrow \left(\exists u \in \bigtimes_{i=1}^{k} [v_i] : \tau' \subseteq u \wedge o(u) = pass \right).$$

We note that several other names for the very same concept appear in the literature, e.g., *failure-causing schemas* in [82], *failure-inducing interactions* in [83], *failure-inducing combinations* in [84] and *faulty interactions* in [85] and [86]. Despite different names, the goal of combinatorial fault localization is always the identification of failure-inducing t-way interactions in the input to the SUT. In the interest of accurately naming the concept defined above, we are refining existing terminology to *failure-inducing t-way interaction*. We do not draw on the other terms found in the literature (see above), as they do allow for ambiguity. For example, when considering them through the lens of software testing, the term "faulty interactions" does not apply, because the interactions are not *faulty* per se, they only induce a failure. For other terms there might be a lack of clarity with regards to other concepts in CT, such as "failure-inducing combinations", since in CT the term *combinations* usually refers to a combination of parameters, not yet specifying their values [87]. While some t-way interactions involving a combination of parameters may be inducing failure, this does not necessarily mean that they all are. Other previous works introduce additional names for (seemingly) existing concepts, such as "failure-causing *schemas*" [82], whereas the notion of (t-way) interactions is widely used in CT, see, e.g., [88], [71] and even in [82]. In that light, the term "failure-inducing t-way interaction (t-FIT)" as proposed in this thesis is merely building on the existing term "failure-inducing interactions" [83], combining it with the established term "t-way interaction" appearing in the realms of combinatorial design theory. This is reasonable from the point of view of software testing, as they induce *failures*, in the sense of [1].

The notion of t-way interactions of combinatorial design theory maps exactly to the concept of FITs in CT-FLA. Combinatorial designs like locating arrays or detecting arrays have properties that allow to identify sets of t-way interactions, if they fulfill respective preconditions. In a typical problem faced in CT-FLA, the set of FITs is generally unknown, and we want to identify them through testing. Therefore, in the

following we may adopt the terminology of CT and speak about *a set of FITs* – instead of *a set of t-way interactions* – when we want to highlight that we do not have information about the position and the values of these t-way interactions.

Further, we note that a list of frequently used acronyms is provided at the end of the thesis in Section 6.2. Therein the first use of each acronym is listed. However, we may recall some acronyms by re-introducing them at appropriate points of the document.

1.4 State of the Art: An Overview of the Combinatorial Testing Cycle

As described in the motivation, the empirical investigations of failure reports and their explanation by means of t-way interactions have led to an increased interest in using combinatorial designs for software testing in general. In particular, they led to an increased use of CAs as (abstract) software test sets.

However, the aspects of software testing that are affected by CT are not limited to test generation. In order to illustrate this, we build on the "Combination Strategies Testing Process" as described by Grindal and Offutt [26] and slightly adapt it. Figure 1.3 presents the *combinatorial testing process* and visualizes those parts of a simplified testing cycle which are influenced by and in turn influence combinatorial design theory (highlighted in red). In the following, we present a compact overview of these parts, as well as the state of the art of combinatorial testing.

It should be noted that providing a detailed survey of all individual parts would go beyond the scope of this thesis. Still, in order to provide context of the individual contributions presented in this thesis, we will discuss related works in the respective chapters.

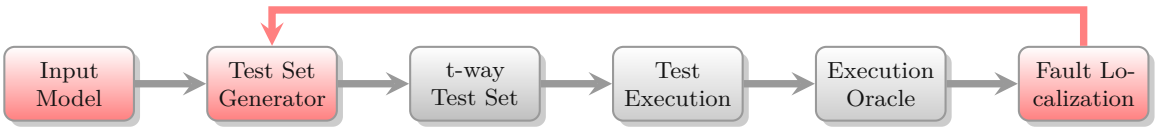


Figure 1.3: The *combinatorial testing process/cycle*: a simplified testing process (CT-affected parts in red) for a given SUT. See the "Combination Strategies Testing Process" in [26].

1.4.1 Input Model

As mentioned before, devising an IPM is a fundamental precondition for applying CT to an SUT and is, thus, the important first step in the combinatorial testing cycle. Yet, it is often difficult to create meaningful IPMs, as they require a lot of domain knowledge and often heavily depend on human interaction. In their fundamental work [26], Grindal

and Offutt suggest a structured modeling method for deriving IPMs for combination strategies. They propose two methods for doing so – in particular for black-box testing applications – both of which do not require any knowledge about the SUT's internals. The *interface-based input parameter modeling* uses a one-to-one mapping between the interface parameters and those of the IPM. The advantage is that the identification of parameters can be straightforward, whereas a disadvantage can be that existing requirements or constraints are not reflected in the interface parameters. In contrast, *functionality-based input parameter modeling* aims to identify parameters and their values based on the intended functionality of the SUT, facilitating the inclusion of more semantic information in the IPM [26]. Since specifications can be interpreted in various ways, existing works aim to guide and support testers without limiting their creativity, see for example [77]. Although considered a creative task, input parameter modeling is subject to automation, e.g., in [89] a rule-based semi-automatic approach is proposed to assist test designers creating IPMs from Unified Modeling Language (UML) diagrams.

State of the Art

In existing work, researchers have made significant efforts to expand on the connections between IPMs and combinatorial testing strategies. For example, the authors of [79] consider the process of input space modeling as split into two steps: (i) input structure modeling, and (ii) input parameter modeling. They propose a method for combinatorial test set construction based on the input structure. Further, the authors of [90] consider the construction of combinatorial test sets for composed systems whose modules operate sequentially. They propose a method where test suites for modules are constructed in a way that the composition of the test suites yields a valid test plan for the sequential execution of a second module upon the output of the first module. Borazjany et al. [91] consider an input space modeling methodology dedicated to CT which divides the derivation of an IPM into two steps. The first step aims at capturing structural relationships among different components in the input space, and the second step identifies parameters, their values, and relations and constraints for the individual components. Focusing on input structure modeling, the work proposes a strategy for how to perform unit and integration testing based on the input space structure model. Segall et al. [92] describe recurring properties of the modeled input spaces for combinatorial testing, independent from the application domain of the SUT. They argue that, since these patterns are often hard to identify, they lead to common pitfalls in the derivation of IPMs. They discuss five modeling patterns aiming to avoid such pitfalls and to facilitate the derivation of IPMs that are *correct* (i.e., correctly capture the modeled input space), *complete* (i.e., do not omit an important part of the modeled input space), and *not redundant* (i.e., do not explicitly enumerate different cases that are actually equivalent). Further, I have elaborated on the connection between modular software architecture and combinatorial testing in my previous work: [93] shows how CT can be applied to systems that are modeled as a composition of components. Based on a combinatorial plug-in construction, together with structural models of the composed SUT and t-way test suites for its components, a unified t-way test suite for the entire SUT is constructed. In other

words, the work in [93] shows how to leverage combinatorial testing from a lower level to a higher level – e.g., how to get from integration testing to systems testing – showing that CT has the potential to be homeomorphically embedded in the software development and testing processes. Finally, the authors of [94] promote the automated derivation of IPMs for their use in CT from ontologies which are formally introduced as a sets of concepts together with attributes and relations. They present an algorithm that converts ontologies to combinatorial test sets and exemplify it by applying it to a sample ontology.

Significant effort has also been devoted to devise combinatorial methods for SUTs that have models capturing the sequential dependencies in the SUTs' input. For example, in [75], the authors present combinatorial methods for testing SUTs where inputs can be modeled by a fixed number of *events* that appear successively, leading to the notion of *sequence covering arrays*. The SUTs' behavior then depends on the order of these events. The authors of [95] go one step further and merge combinational and sequential combinatorial methods. That means, while the input to the SUTs' input is still represented through an IPM, the outlined modeling methodology can capture sequential dependencies on top of combinational dependencies of input parameters. Hence, a test specifies which value is given to each parameter and in what order they are used.

1.4.2 Test Set Generation

There are several objects found in combinatorial design theory that can be used to derive combinatorial test sets. These include the already mentioned variable strength covering arrays, locating arrays, detecting arrays and sequence covering arrays, to name a few. Their common characteristic is the exhaustive *coverage* of a specific set of t-way interactions in all sub-arrays specified by a set of column selections of the array. However, in applications of combinatorial testing, the problem of combinatorial test set generation is often understood as being identical with the problem of (constrained) covering array generation. In other words, CT with test sets that are based on CAs is the most prominent representative amongst combinatorial testing techniques. This is also the reason why **CAs take a central role in this thesis**.

As mentioned in the preliminaries (Section 1.3), for given t, k, v the existence of a CA is always guaranteed, since the full-factorial design, i.e., the array comprised of all row vectors of length k, constitutes a CA with $N = v^k$ rows. The notorious problem that arises for CAs, is that of finding optimal CAs, i.e., given t, k, v we are asking for a $\mathsf{CA}(N; t, k, v)$ with N being as small as possible, i.e., $N = \mathsf{CAN}(t, k, v)$. Interpreted in terms of software testing - since the rows of CAs give rise to the tests - finding optimal CAs translates to finding minimal test sets that cover all t-way interactions of the input parameters. The analogue holds for generalizations of CAs. For mixed-level covering arrays,the restriction of an uniform alphabet is lifted, i.e., the values in different columns can take values from different domains [58]. For *variable strength covering arrays* the coverage properties are generalized, i.e. quantification over subarrays is no longer over *all* arrays comprised by t different columns, but instead a set $\mathscr{I}$ specifies in which subarrays the respective tuples must appear as rows [59]. The possible existence of *soft* and *hard*

constraints – which manifest in tuples that *need not* and *must not* appear [61] – leads to the notion of *covering arrays with constraints* [96]. Finally, sequence covering arrays $\mathsf{SCA}(N, S, t)$ [75], which can be applied for combinatorial event sequence testing are defined as $N \times S$ arrays, with entries coming from the alphabet S with the defining property that each permutation of any t symbols from S appears at least once as a sub-sequence of one of the rows of the array.

State of the Art

CAs can be regarded as a generalization of orthogonal arrays [24] and as such they have been studied as a topic of combinatorial design theory. Especially the problem of finding *small* covering arrays that represent the backbone of CT has attracted attention from many researchers, theorists as well as applied researchers. Despite the expended effort, especially the generation of *optimal* CAs remains a challenging problem and much research has been devoted to finding approximations to optimal CAs.

As of this writing, for the general case there is no known strategy that efficiently constructs optimal CAs, nor is a method known that determines the covering array number $\mathsf{CAN}(t, k, v)$, for given parameters t, k, v. Current research efforts focus on the generation of CAs with a *minimized* number of rows, approaching the goal of optimal CAs from above. Many different approaches have been followed in order to generate such *optimized* CAs, i.e. such with a small number of rows. Thus, the state of the art in terms of generating optimized CAs is not provided by a single algorithm, but is represented by a collection of different approaches, including mathematical approaches, combinatorial constructions, meta-heuristic algorithms, and last but not least combinatorial algorithms. This is also nicely illustrated by the plurality of methods that constitute the state of the art in optimal and optimized CA generation, recorded under [97]. The method that was used to derive the respective state of the art result is indicated in the column headed by "**Source**". Independently from this, the work in [98] provides the most recent survey of such CA generation methods.

Although unsolved in general, there exist some cases where *optimal CAs* are known. For example the binary case of strength two is solved entirely, i.e. for any given $k \in \mathbb{N}$, it is known how many rows an optimal $\mathsf{CA}(N; 2, k, 2)$ has and how such can be generated [99, 100]. Since orthogonal arrays of index unity represent optimal CAs, the so called *Bush construction* is capable to generate optimal CAs [101]. It is an algebraic construction and uses properties of finite fields to construct orthogonal arrays of strength t, by evaluating all polynomials of degree up to $t - 1$ over a finite field at all elements of the finite field. Thus, the cardinality of the alphabet must be a prime power and the number of columns is restricted. See also [65]. Further, some individual cases can be closed using exact approaches such as SAT solving or constraint satisfaction problem (CSP) encodings [69], others by making use of combinatorial constructions [102]. A thorough study of CAs and their connections to other combinatorial designs is given by [103], presenting amongst others, basic bounds, direct constructions, recursive constructions for CAs.

Combinatorial constructions can be very potent for *optimized CA* construction, due to good scalability to larger problems, but they are often limited to work only when specific conditions, e.g. regarding the alphabet size, are met. The works in [104], [105] and [106] rely mainly on cyclic permutations of vectors with specific properties in order to generate CAs, while other methods rely on connections between CAs and other combinatorial structures, such as *covering perfect hash families* [107], [108]. Their good scalability makes it so that combinatorial constructions populate the leader board for optimized CA generation for larger CA parameters.

Meta-heuristic approaches currently provide smallest known CAs in several cases. Often they yield promising results only for smaller problem instances as otherwise they are inefficient, suffering under the combinatorial explosion, when applied to larger problem instances, i.e., for CAs with larger parameters. Many different meta-heuristic algorithms have been applied for CA generation, including population-based meta-heuristics, such as genetic algorithms [109] and particle swarm optimization [110]. The current state of the art however shows, that single-solution meta-heuristics such as simulated annealing produce the best results, holding several records on currently best known upper bounds on covering array numbers [111], [112]. In some works, such as [113], [114] and [115] meta-heuristic approaches are paired with combinatorial constructions, which produces the smallest known CAs so far for several cases of given parameters t, k and v. Similarly, two-stage algorithms combining a randomized approach with greedy or heuristic algorithm, as presented in [116] currently hold the record for producing the smallest CAs for several CA instances.

Combinatorial algorithms, such as [36], [27] and [71] are amongst the most prominent algorithms for CA generation. Although they are currently not necessarily contributing to the state of the art of optimized or optimal CA generation, the majority of existing tools for CA generation ([36], [117], [78]) implement such algorithms. This is mostly explained due to combinatorial algorithms having fairly small execution time and being flexibile with regards to (mixed) alphabet sizes and constraints in the IPM. This makes them preeminent to provide CAs that can be readily used for combinatorial test set generation and practical applications.

Notably, the deterministic density algorithm (DDA) presented in [118] for CAs of strength two, and later generalized to the density-based greedy algorithm for higher strength CAs in [71], represents a mile-stone in research on optimal CA generation, as it represents the first (and currently only) algorithm for generating a $\mathsf{CA}(N; t, k, v)$ with $N \in O(\log k)$ that has computational time complexity polynomial in k for fixed t and v.

While optimal CA generation appears to be a challenging task, we do not know about its actual computational complexity. However, in several publications the problem is termed as NP-hard, without any reference or referring to the hardness of a related, but not identical problem, e.g., works like [119], [120], [121] and [30].

Regarding sequential combinatorial test set generation the pioneering works are [75] and [63]. The work in [75] presents combinatorial methods for event sequence testing of SUTs and present a greedy algorithm for SCA generation. This algorithm considers a set of randomly generated permutations of the events and selects in each step a permutation that maximizes the number of newly covered t-way sequences of events. This permutation, together with its reversal are added to the SCA to be constructed. This is repeated until all t-way sequences are covered by the rows of the array. In [63] SCAs, respectively the equivalent notion of *completely t-scrambling sets* coming from combinatorial design theory, are formally investigated. The authors present a polynomial time algorithm that generates SCAs whose number of rows stays within logarithmic function of the number of events when the strength t is fixed. They also present a combinatorial product construction for strength $t = 3$ for SCAs for $v \cdot w$ events starting from SCAs for v and w events. Finally, there also exists a pioneering work on sequential CT where test sequences are modeled as *sequences* of events, rather than *permutations* of events [122]. Several algorithms to generate such test sequences are proposed and compared analytically and experimentally. The same notion of test sequences for sequential CT is considered in [123], where an automata theoretic construction is proposed for their generation.

1.4.3 Combinatorial Fault Localization

Combinatorial fault localization is an aspect of combinatorial testing that aims to *identify failure-inducing t-way interactions* (FITs) in the input of an SUT. Instead of mere detection (i.e., verification of presence) of failure caused by t-way interactions, we are interested in identifying *which* t-way interactions triggered this failure. We want to reiterate that in CT the SUT is considered a black-box and that CT-FLA can, thus, not tell us *why* a specific t-way interaction is causing the failure, but only that it does so. To that extent, CT-FLA methods are clearly distinguished from "traditional" software fault localization methods as part of the debugging process. Ammann and Offutt [1] define debugging as "the process of finding a fault given a failure"; also, there is a broad understanding that debugging includes resolving the identified faults, see for example [124] or [125]. Traditional methods – such as testing-based fault-localization [126], [127], or delta-debugging [128] – typically require information about the program execution, i.e., they are white-box or grey-box testing methods. This seems necessary when we consider resolving of faults as part of the debugging process, e.g., by finding and fixing faulty statements in the source code. In contrast, in CT we generically consider the SUT as a black-box and cannot assume any knowledge about its internals. In this context, one may understand CT-FLA as a method that identifies the manifestation of failure-triggers already in the input of a program instead of its source code. Still, research has shown that the knowledge of FITs can facilitate the location of faulty statements in source code [129], making CT-FLA an important final step in establishing a closed combinatorial testing cycle (see Figure 1.3) that represents a useful and universal tool for developers as well as a crucial step towards the automation of combinatorial testing.

State of the Art

CT-FLA methods can be divided into *adaptive* and *non-adaptive* approaches. Adaptive approaches, such as [130] and [83] rely on online communication to the SUT during testing. The test set is generated during test execution, whereby results of earlier tests influence the generation of later tests. Through alternating test execution with test generation, the set of potential FITs is reduced and concretized iteratively. We will review these works in more detail in Section 5.1. Generally speaking, the adaptive CT-FLA methods have the advantage to not make strong assumptions about the SUT; however, they do depend on communication with the SUT. A survey on adaptive combinatorial fault localization is given in [131].

Non-adaptive CT-FLA methods, on the other hand, do not require an online communication with the SUT. Fault localization solely relies on properties of the underlying combinatorial design – giving rise to the test set – and on the result of test execution in form of the pass/fail assignment obtained from the testing oracle. In that regard, non-adaptive methods can be studied as part of combinatorial design theory, independent from concrete SUTs. The combinatorial designs used are generic and can be deployed for testing of various SUTs. As a result, combinatorial designs such as *locating arrays*, *detecting arrays* [62], *error-locating arrays* [86], and *constrained locating arrays* [132] came into the focal point of combinatorial testing. It is clear that these combinatorial designs are closely related to CAs, considering that the location of a t-FIT via testing requires the execution of the very same t-way interaction in the first place. For example, (d, t)-detecting arrays, as introduced in [62], are covering arrays of strength t with the additional property that for any set $\mathcal{T}$ of d different t-way interactions any other t-way interaction $\tau \notin \mathcal{T}$ can be distinguished only by the rows of the array, see also Definition 1.3.14. This is ensured as any such τ must be covered in at least one row of the array that does not cover any element of $\mathcal{T}$. Such a row can be considered a "witness" for τ, not being an element of $\mathcal{T}$. When applied for software testing, these combinatorial properties ensure that detecting arrays can distinguish failure-inducing t-way interactions from those which do not induce failures. However, this is only possible when the number and the strength of the FITs of the SUT meet the combinatorial properties of the deployed combinatorial design. For example, for the case of (d, t)-detecting arrays we can locate the FITs of an SUT that has exactly d different FITs of strength t.

Remark 1.4.1 *When applied for software testing, detecting arrays allow for the following localization procedure (as originally described in [62]): iterate over all passing tests; mark all t-way interactions covered by passing tests as non-failure-inducing; the set of all failure-inducing t-way interactions is retrieved as the remaining un-marked set.*

However, it must be noted that this location procedure is guaranteed to work only if the number and the strength of the FITs of the SUT meet with the properties of the deployed combinatorial design. In the case of (d, t)-detecting arrays we can locate the FITs of an SUT that has exactly d-different FITs of strength t. Still, there exist similar structures

– $(\bar{d}, \bar{t})$-detecting arrays – that can be used for the location of *up to* d failure-inducing $\leq t$-way interactions. Notions like $(\bar{d}, \bar{t})$-detecting arrays generalize this constraint, as they can locate all FITs of SUTs with at most d FITs of length *up to* t (recall Definition 1.3.8). A brief formal definition of these combinatorial structures is given in the Preliminaries (Section 1.3). For a detailed introduction and discussion, the interested reader is referred to [62].

Combinatorial designs for non-adaptive CT-FLA have been studied increasingly after their definition and introduction in [62]. For example, the optimal LAs and their construction was investigated in [133], focusing on the case of $(1, t)$-LAs for the location of a single t-FIT. The authors of [133] derive a lower bound for the size of such LAs and describe a combinatorial construction for LAs attaining this bound that uses transversal designs. The work in [134] explores randomized methods for the generation of LAs and DAs, and derandomized variants thereof using conditional expectation methods and column resampling. The resulting algorithms are used to derive LAs and DAs with up to 100 columns for the location of a single 2-FIT. A related approach is followed in [135] where column resampling is combined with search space partitioning for the check if an array is locating. The resulting algorithm can therefore be used for generating LAs, but also allows to verify if a given array is an LA. We also note that a notion of *constrained detecting arrays* has been studied [136] which facilitates fault localization of SUTs with constraints on the input. This work further proposes to generate these structures using a satisfiability modulo theories solver.

Generally speaking, we can say that non-adaptive CT-FLA methods do have the disadvantage that they make relatively strong assumptions regarding the number and the strength of the FITs present in the SUT; on the other hand, their advantage is that they are independent from communication with the SUT, in contrast to adaptive approaches. Additionally, once derived, the combinatorial designs can be used as abstract test sets for arbitrary SUTs as long as they suit the IPM. Colbourn and Syrotiuk [76] present a framework for combinatorial fault characterization. Amongst other things, this work discusses probabilistic methods for the generation of locating and detecting arrays, and it provides a thorough review of the development of non-adaptive combinatorial fault localization, putting it into context with combinatorial group testing, compressive sensing, and statistical design of experiments. Furthermore, they present combinatorial constructions of detecting arrays after giving a compact introduction to the topic of non-adaptive CT-FLA, including a survey of prior work in [137].

Open problems of (non-adaptive) CT-FLA are also mentioned in [76], amongst others the absence of effective tools dedicated to the construction of combinatorial designs for combinatorial fault localization. Further, as discussed in [76], an efficient recovery strategy that quickly determines FITs requires more structure than a naive strategy. Compared to CAs that merely cover all t-way interactions, i.e., detect the presence of FITs, combinatorial structures that also allow for identifying (a specific number of) FITs do have a higher number of rows. This is due to the fact that in order to *locate* we require more information than to *detect*, see also [76]. Strengthening the combinatorial

requirements underlying locating arrays in such a way that allows for a more efficient FIT location strategy leads to the definition of detecting arrays. However, the location strategy still requires the enumeration of all t-way interactions covered by all passing tests which can become infeasible, especially for SUTs with a large number of input parameters.

1.5 Methodology of Combinatorial Design Theory and Software Testing

Advancing the combinatorial testing cycle. As noted previously, the aim of this thesis is to develop and further combinatorial design theory for software testing. We are pursuing this aim by advancing the combinatorial testing cycle, illustrated in Figure 1.3. This means that we strive to contribute to input modeling, test set generation and fault localization in order to improve and extend the CT process. This includes devising methods that can advance automation and application of CT for testing SUTs in novel domains or large scale systems.

The overall methodology of *combinatorial design theory for software testing* can be described on a high-level as a combination of *abstraction*, *solution*, and *implementation*.

We approach a real-world problem, in our case pseudo-exhaustive (combinatorial) software testing, and try to solve it by considering a formal analog. The formal analog is obtained by abstracting the essential characteristics of the problem at hand and by rephrasing it as a formal problem of discrete mathematics or theoretical computer science. In this way, we make it susceptive to formal methods, allowing to solve the formal problem via a theoretical construction, the proof of a theorem, its realization in an algorithmic procedure, or by developing an algorithm. Having devised a solution to the theoretical problem, the next step is to transform it so that it can be applied to the initial real-world problem. This includes: the implementation of algorithms as computer programs, the realization of abstract procedures as real-life procedures, the application of the derived solution to the problem at hand (often requiring appropriate modeling techniques), and the translation of abstract mathematical objects – i.e., combinatorial designs – to executable test suites. An overview of the overall methodology is depicted in Figure 1.4.

Due to the nature of the involved domains – Combinatorial Design Theory and Software Testing – the methodologies used in this thesis incorporate methods of *formal sciences*, in particular of discrete mathematics and theoretical computer science, and *empirical methods*, including empirical algorithmics, proof of concept applications, and case studies. The thesis is structured in two parts. The first part focuses on the contributions to combinatorial design theory, and the second part pertains to the applications of combinatorial design theory for software testing. As a result, the above-mentioned methodologies split roughly along these two parts.

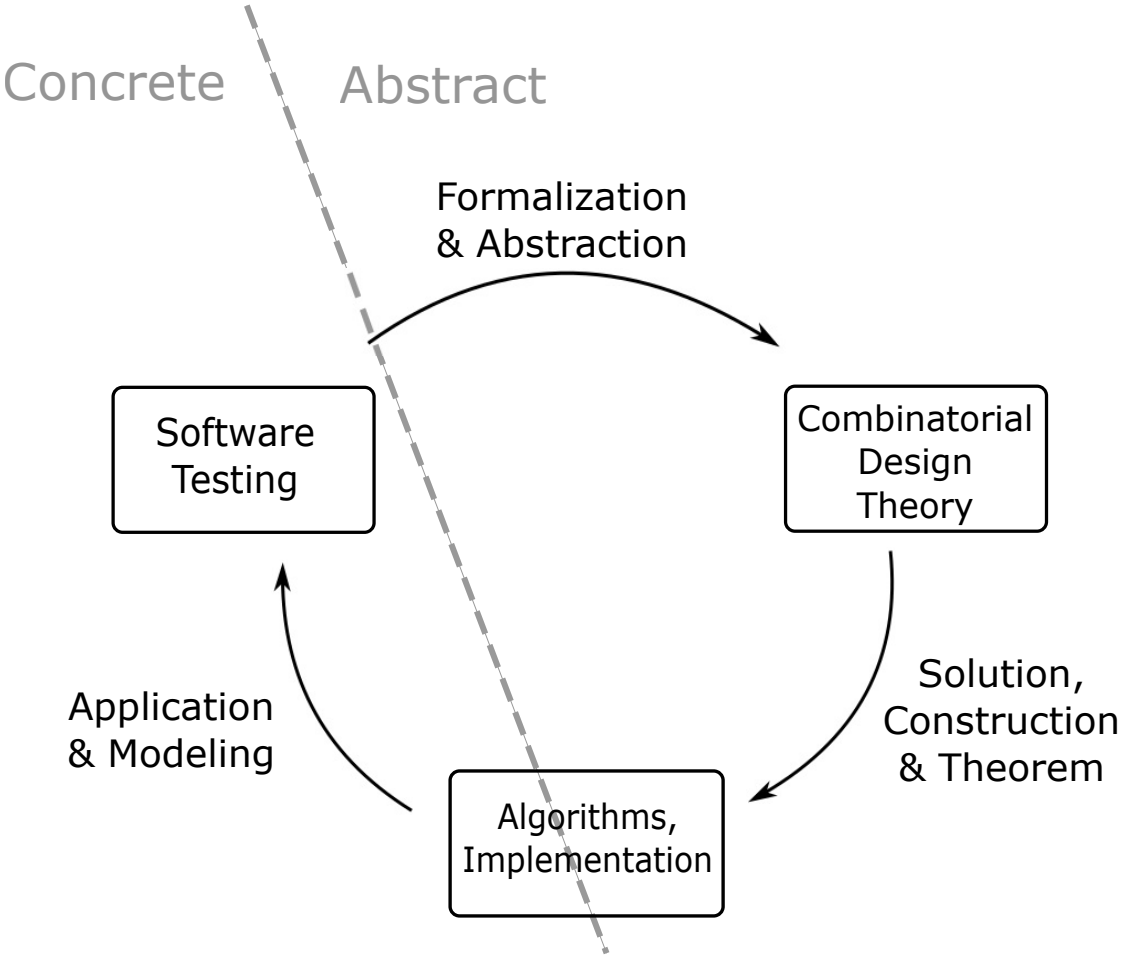


Figure 1.4: Overall methodology of Combinatorial Design Theory and Software Testing.

1.5.1 Structure of the Thesis

In the first part, we make mainly use of formal methods and methods of experimental algorithmics. Formal methods will be applied whenever feasible, but primarily in Chapter 2, and will include: the formulation of the subject problems as formal problems of computational complexity, the formulation of algorithms and their analysis, and the formulation of mathematical statements and their proof. Where applicable, but primarily in Chapter 3, methods of empirical algorithmics will be applied, i.e., algorithms will be implemented and their performance compared by means of benchmarking against similar approaches, including but not limited to algorithmic approaches of the same nature and the state of the art.

In the second part, we make use of empirical methods mainly, where experimental and empirical evaluations of the developed methods take place. Proofs of concept will be devised via applications to sample SUTs. Practical feasibility and applicability can be shown through a demonstration in a concrete case study. In some cases, where the

circumstances permit, the methods are deployed in industrial case studies.

As mentioned above, Part I of this thesis pertains to rather theoretical aspects, and Part II covers applied aspects of combinatorial design theory for software testing. Both parts are divided into two chapters, whereby each chapter starts with an introduction followed by a review of related literature. This review presents a detailed state of the art regarding the sections of the chapter. Each chapter ends with a brief conclusion. To conclude this thesis, we summarize the contributions (Section 6.1), put them into the context of the combinatorial testing cycle, and outline future developments and potential avenues (Section 6.2).

1.5.2 Publications Underlying the Thesis

This thesis is based on contributions that are part of the following publications.

Contributions to Combinatorial Design Theory

[66] L. Kampel and D. E. Simos, "A survey on the state of the art of complexity problems for covering arrays," *Theoretical Computer Science*, vol. 800, pp. 107–124, 2019

Contributions: developing the research question, developing and proving the complexity reductions, substantial contribution in writing the paper.

[67] L. Kampel, D. E. Simos, B. Garn, I. S. Kotsireas, and E. Zhereshchin, "Algebraic models for arbitrary strength covering arrays over v-ary alphabets," in *Algebraic Informatics*, (Cham), pp. 177–189, Springer International Publishing, 2019

Contributions: co-developing the research question, devising the modeling concept, developing the proofs, substantial contribution in writing the paper, designing the algorithms, co-implementation of research prototypes, co-guiding the experimental evaluation.

[138] L. Kampel, M. Leithner, and D. E. Simos, "Sliced AETG: a memory-efficient variant of the AETG covering array generation algorithm," *Optimization Letters*, vol. 14, no. 6, pp. 1543–1556, 2020

Contributions: developing the research question, developing the algorithmic concepts, developing the proofs, guiding the implementation, designing the experimental evaluation, substantial contribution in writing the paper.

[139] L. Kampel, I. Hiess, I. S. Kotsireas, and D. E. Simos, "Balanced covering arrays: A classification of covering arrays and packing arrays via exact methods," *Journal of Combinatorial Designs*, vol. 31, no. 4, pp. 205–261, 2023

Contributions: developing the research question, devising the algorithmic architecture, co-developing the algorithmic concepts, co-developing the proofs, guiding the implementation, designing the experimental evaluation, substantial contribution in writing the paper.

[140] L. Kampel, M. Koelbing, and D. E. Simos, "A reduced-shuffe product of finite automata arising from sequential combinatorial testing." Submitted for publication.

Contributions: developing the research question, guiding the proofs, co-developing the proofs, contribution in writing the paper

[141] L. Kampel, M. Wagner, I. S. Kotsireas, and D. E. Simos, "How to use Boltzmann machines and neural networks for covering array generation," in *Learning and Intelligent Optimization*, (Cham), pp. 53–68, Springer International Publishing, 2020

Contributions: co-developing the research question, devising the algorithmic architecture, co-developing the algorithmic concepts, guiding the implementation, designing the experimental evaluation, substantial contribution in writing the paper.

[142] L. Kampel, M. Wagner, I. S. Kotsireas, and D. E. Simos, "A primer on the application of neural networks to covering array generation," *Optimization Methods and Software*, vol. 37, no. 3, pp. 1165–1202, 2022

Contributions: co-developing the research question, devising the algorithmic architecture, co-developing the algorithmic concepts, guiding the implementation, designing the experimental evaluation, substantial contribution in writing the paper.

[143] M. Wagner, L. Kampel, and D. E. Simos, "IPO-Q: A quantum-inspired approach to the IPO strategy used in CA generation," in *Mathematical Aspects of Computer and Information Sciences*, (Cham), pp. 313–323, Springer International Publishing, 2020

Contributions: co-developing the research question, co-design of the algorithmic concepts, co-designing the experimental evaluation, co-guiding the experiments, supporting writing the paper.

Further, the changes that the following article has undergone during its revision, based on the *Theoretical Computer Science* referee reports received in November 2018, contribute to this thesis:

[144] L. Kampel, M. Leithner, B. Garn, and D. E. Simos, "Problems and algorithms for covering arrays via set covers," *Theoretical Computer Science*, vol. 800, pp. 90–106, 2019

Contributions: developing the algorithmic concepts, designing the experimental evaluation, guiding the experiments, substantial contribution in writing the paper.

Contributions to Combinatorial Software Testing

[145] D. E. Simos, L. Kampel, and M. Ozcan, "Combinatorial methods for testing communication protocols in smart cities," in *Learning and Intelligent Optimization*, (Cham), pp. 437–440, Springer International Publishing, 2019

Contributions: co-devising the modeling concept, test set generation, contribution in writing the paper.

[146] D. Jarman, R. Smith, G. Gosney, L. Kampel, M. Leithner, D. Simos, R. Kacker, and R. Kuhn, "Applying combinatorial testing to large-scale data processing at Adobe," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 190–193, 2019

Contributions: co-designing test set generation, supporting writing the paper.

[147] B. Garn, L. Kampel, M. Leithner, B. Celic, C. Çulha, I. Hiess, K. Kieseberg, M. Koelbing, D.-P. Schreiber, M. Wagner, C. Wech, J. Zivanovic, and D. E. Simos, "Applying pairwise combinatorial testing to large language model testing," in *Testing Software and Systems*, (Cham), pp. 247–256, Springer Nature Switzerland, 2023

Contributions: co-framing the research question, devising the modeling concept, guiding test set generation, co-designing the experimental evaluation, co-guiding the experiments, supporting writing the paper.

[68] L. Kampel, P. Kitsos, and D. E. Simos, "Locating hardware Trojans using combinatorial testing for cryptographic circuits," *IEEE Access*, vol. 10, pp. 18787–18806, 2022

Contributions: co-developing the research question, co-devising the modeling concept, designing the experimental evaluation, designing the algorithms, implementation of research prototypes, co-guiding the experiments, substantial contribution in writing the paper.

[148] L. Kampel, D. E. Simos, D. R. Kuhn, and R. N. Kacker, "An exploration of combinatorial testing-based approaches to fault localization for explainable AI," *Annals of Mathematics and Artificial Intelligence*, vol. 90, no. 7, pp. 951–964, 2022

Contributions: co-developing the research questions, substantial contribution in writing the paper.

[149] L. Kampel, M. Wagner, D. E. Simos, M. Nica, D. Dodig, D. Kaufmann, and F. Wotawa, "Applying CT-FLA for AEB function testing: A virtual driving case study," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 237–245, 2023

Contributions: co-developing the algorithmic architecture, co-developing the algorithmic concepts, co-guiding the experiments, substantial contribution in writing the paper.

Part I

Contributions to Combinatorial Design Theory

CHAPTER 2

Combinatorial Test Sets as Objects of Theoretical Computer Science and Discrete Mathematics

The second chapter of this thesis aims at getting a deeper understanding of combinatorial test sets. Therefore we study combinatorial test sets as abstract, discrete mathematical objects as which they appear in combinatorial design theory. Similarly, we can formally investigate the generation problems emerging with the respective combinatorial design-theoretic notions. Covering arrays have thereby a central role which corresponds to their crucial function in the applications of combinatorial testing, as mentioned in Chapter 1. However, this chapter also includes generalizations of CAs like mixed-level covering arrays and variable-strength covering arrays as well as notions used for sequential combinatorial testing.

With the intention of better understanding the combinatorial objects that provide the basis for combinatorial testing, this chapter is motivated by the following questions:

- How difficult or *hard* is the problem of generating optimal covering arrays and related objects?
- How can the problem of optimal combinatorial test set generation be captured or *modeled* in terms of discrete mathematics and theoretical computer science?
- Can we say anything about the *structure* of classes of covering arrays?

These questions are addressed in the following sections. In order to investigate the difficulty of optimal combinatorial test set generation, we phrase optimal CA generation and related problems as formal problems of theoretical computer science in Section 2.2.

This also allows us to survey and reason about the computational complexity of these problems. We then look at different *modelings* of the CA generation problem. First, in Section 2.3, we consider an (existing) mapping to a well-known problem of theoretical computer science, the minimal set cover problem. Second, in Section 2.4 we establish a method to model the CA generation problem by means of commutative algebra and symbolic computation – a CA appears then as the solution to a multivariate polynomial equation system. We then have a more detailed look at classes of CAs in Section 2.5 by dissecting them using a balance-notion which arises when considering the intersections of classes of CAs with classes of another combinatorial design (packing arrays). By means of a classification of all non-equivalent balanced CAs of a specific size, we can get a detailed picture of the respective class of CAs. Finally, in Section 2.6 we propose, with regards to sequential combinatorial testing, an automata-theoretic notion to model the problem of finding short test sequences that contain sub-sequences of a given length.

2.1 Related Literature

There exist several papers that pertain the computational complexity of problems that are related to optimal CA generation, or the determination of CAN, which we briefly mention here, and investigate them more detailed in Section 2.2. Maybe the first paper of this kind was presented in [30], where Seroussi and Bshouty studied vector sets for exhaustive testing of logic circuits. The (L, d)-*universal sets* of vectors considered therein can be understood as a special kind of VCAs. They present construction methods as well as upper and lower bounds on the size of (L, d)-universal vector sets. Finally, they also show that "*the design of an optimal exhaustive test set for an arbitrary logic circuit is an NP-complete problem*" [30]. Danziger et al. have investigated the computational complexity of problems arising with a notion of constrained covering arrays, called *covering arrays avoiding forbidden edges* in [60], and have proven several results for such structures. This line of research was extended later by Maltais and Moura in [150] and [119]. Nayeri, Colbourn and Konjevod study the post-optimization of CAs in [121]. A problem occurring thereby is, to identify so called *flexible positions*, that are entries of a CA that can be changed while the defining property of the CA remains intact. In [121] it was shown that it is NP-complete to decide for a given CA if there is a set of flexible positions, that could be changed simultaneously, of at least a given size. We will review these results more detailed and put them into context later in Section 2.2. The above works will be considered more detailed in the next section.

In this chapter we will further review a connection between CAs and set covers (SC), which has also been mentioned informally in the literature before, for example in [65, Sec. 2.4]. Relying on the close connection between SC problems and integer linear programming (IP) problems, i.e. the fact that any SC problem is equivalent to a 0-1 IP problem, see for example [151], we can consider also works that formulate problems related to CAs as IP problems as related works. As such, Sloane gives an IP formulation programming formulation that can be applied for finding an optimal CA [40]. Such an encoding was later also used by Williams and Probert [152] to compute some optimal

CAs. and by Bulutoglu and Margot [153] for the classification of CAs and related notions, in particular orthogonal arrays and packing arrays. With regards to algebraic encodings of CA problems, the related works consist of [154], presenting an algebraic encoding of the binary CA of strength two. This work represents the basis for the work presented later in Section 2.4. This line of research was later extended by Garn and Simos in [155], to capture also CAs with constraints. Additionally, we mention that a similar algebraic encoding is used for the computation of SCAs by Koelbing et al. in [156].

As already mentioned in the state of the art section 1.4.2 the pioneering works related to sequential CT are [75] and [63]. The work in [75] proposes the application of SCAs for testing software that takes sequences of events as input. In [63] a connection between SCAs, and a notion known in combinatorial design theory as *completely t-scrambling sets* is established. Further, lower bounds on the number of rows in optimal SCAs are derived, as well as a recursive product construction for SCAs of strength $t = 3$. In [157] Murray and Colbourn present a post-optimization approach to the generation of SCAs. Starting from a test set that covers the desired subsequences, local changes are performed with the aim of rendering a test sequence redundant for the coverage, so that it can be removed from the test set. Abandoning the notion of arrays, and turning to sequential combinatorial test sets of sequences, Yu et al. present [122] algorithms for sequential combinatorial test set generation for SUTs that are modeled by means of a (non-deterministic) labeled transition system. They propose an algorithm to generate all valid t-way target sequences and, based on this, four algorithms that generate a *t-way test sequence set*, i.e. a minimized set of transition sequences where each *valid t-way target sequence* appears as sub-sequence of the labels of at least one transition sequence in the set [122, Def. 8]. Duan et al. study the problem of sequential combinatorial test set generation in the presence of constraints in [158]. They use a formalism based on finite automata for expressing the sequencing constraints of a given SUT. Their approach was realized in an algorithm for sequential test set generation. Bombarda and Gargantini [123] also follow an automata theoretic approach to sequential combinatorial test set generation. They consider *t-wise automata*, which correspond to subsequences of length t, and generate test sequences through a repeated product of t-wise automata with an automaton modeling the SUT. We will review this work more detailed in Section 2.6.

In addition to the already mentioned work [153], further research pertaining the classification of CAs and closely related notions has been conducted. Colbourn et al. perform classification of *radius covering arrays* in [159]. Radius covering arrays can be considered a generalization of CAs, which then emerge as the special case of radius covering arrays of radius $r = 0$. In [160] Torres-Jimenez and Izquierdo-Marquez perform classification dedicated for CAs, based on a SAT encoding. Their results allowed them to determine for CANs. Most recently, Kokkala et al. conducted a study on the structure of small CAs of strength $t = 2$ in [161]. In their study they focused on CAs where the frequencies of symbols per column are as uniformly as possible, they relate to these CAs therefore as *uniform* CAs. Deploying additional pruning conditions, in their computational results allowed them to determine the CAN for ten CA instances.

2.2 Survey and Study of the Computational Complexity of Optimal Covering Array Problems and Related Problems

Considering the efforts of numerous researchers pertaining the study and generation of optimal CAs, it seems apparent, that it is a very challenging task. However, as also mentioned in Section 1.4, we do not know its actual computational complexity. Yet, in several publications the problem is termed as NP-hard, giving no reference at all or referring to the hardness of related, but certainly not identical problems (e.g. [119], [120], [121] and [30]). Such erroneous citations seem to be not sufficiently questioned and are frequently reproduced by other authors. A first step to resolve the confusions with other problems is to formally formulate the problem of optimal CA generation in terms of computational complexity. Second, relations to related problems for which computational hardness results are already existent, will be set out and possible conclusions for optimal CA generation will be drawn.

This Section is based on the publication [66] and documents an investigation of the hardness of complexity problems related to CAs. First we give some definitions to allow for the presentation of formal complexity problems. This provides the means to distinguish and review the available literature on the subject. For some of these problems, we prove basic complexity properties such as membership in NP and poly-time relations (between them). In addition, our aim is to clarify some results regarding the complexity of certain problems pertaining to CAs as we have identified a number of incorrect statements in the literature, despite some attempts that have been made already in [70] which, as we will show in this paper, are incomplete. For this reason, we additionally conduct a state of the art review of known complexity results for problems related to CAs, supplementing it with some new results.

We start by revisiting a result regarding the number of rows in optimal CAs, which is a basic, but essential result used in the sequel of this Section. Thereafter we give formal statements of complexity problems for CAs. We then continue in the same manner, presenting similar statements for MCAs and VCAs, respectively. Thereafter, we firstly review a result given in [30] and secondly discuss its impact on the problems introduced earlier. We complete the survey by reviewing and analyzing other complexity results, related to CAs. To finish this Subsection, we provide a comprehensive overview of the state of the art of the computational complexity of problems related to optimal CA generation, including also the results of this thesis.

A Logarithmic Guarantee for Classes of Covering Arrays

There exist several works concerning the logarithmic growth of the number of rows of classes of covering arrays in terms of the number of their columns. In [36] the logarithmic growth of the number of rows of CAs in terms of the number of columns is proven for strength $t = 2$. More general results for CAs of higher strength and VCAs can be found

in [71] and [73] respectively. In comparison to the work in [36] the latter two rely on a density method construction which provides not only a guarantee of existence but also an algorithmic construction. For the sake of making this work more self-contained we generalize the arguments of [36] to higher strengths $t \geq 2$, MCAs as well as VCAs in a straightforward manner, providing yet another proof of the *Logarithmic Guarantees* proven in [71] and [73].

Lemma 2.2.1 *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 $(v_1, \ldots, v_k)$ as well as n be the number of $\mathbf{v}$-ary t-way interactions 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, i.e. a fraction of at least $1/h$ of the $\mathbf{v}$-ary t-way interactions 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-way interactions covered by them, i.e.

$$W := \{(d, \tau) | d \in R \text{ and } d \text{ covers } \tau \in T_{\mathbf{v},k,t}\}.$$

Each $d \in R$ appears exactly $\binom{k}{t}$ as first component of elements of W. Furthermore, a $\mathbf{v}$-ary t-way interaction $\tau = \{(x_1, p_1), \ldots, (x_t, p_t)\}$ is covered by exactly $\prod_{i \in \{1,\ldots,k\} \setminus \{p_1,\ldots,p_t\}} v_i$ rows, and hence appears as second component of exactly that many elements of W. Therefore, $\mathbf{v}$-ary t-way interactions $\{(x_1, p_1), \ldots, (x_t, p_t)\}$ with $(p_1, \ldots, p_t) = (1, \ldots, t)$ 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

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

and prove the theorem by counting the cardinality of V in two different ways.

For any pair $(d, \tau) \in V$, since τ is a $\mathbf{v}$-ary t-way interaction not covered by the rows of A, all pairs (d', τ) where d' covers τ appear in V, and hence τ 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-way interactions that are not covered by the rows of A, we get

$$\ell \cdot n \quad \leq \quad |V|. \tag{2.1}$$

Conversely, let m_d denote the number of $\mathbf{v}$-ary t-way interactions 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

$$|V| = \sum_{d \in R} m_d \leq m \cdot |R| = m \cdot \prod_{i=1}^{k} v_i. \tag{2.2}$$

From (2.1) together with (2.2) 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-way interactions that are not covered by the rows of A. □

With the help of the previous lemma, we can now show the following.

Theorem 2.2.2 (Logarithmic Guarantee) *Let t, v be fixed and $v_i \leq v$ for all $i \in \{1, \ldots, k\}$, then $\mathsf{MCAN}(t, k, (v_1, \ldots, v_k))$ belongs to $O(\log k)$,*

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

Proof. We reuse the notation of Theorem 2.2.1 and its proof. Suppose we construct an array row by row, where in each step we add a row that covers at least a fraction of $\frac{1}{h}$ of the yet uncovered $\mathbf{v}$-ary t-way interactions. Theorem 2.2.1 ensures the existence of such a row in each step. The initial number of uncovered $\mathbf{v}$-ary t-way interactions is

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

We are therefore guaranteed having constructed an MCA after adding at most s rows to an initially empty array, where

$$n_0 \cdot \left(1 - \frac{1}{h}\right)^s < 1 \tag{2.3}$$

$$\Leftrightarrow \quad s \cdot \ln\left(1 - \frac{1}{h}\right) < \ln\left(\frac{1}{n_0}\right) \tag{2.4}$$

$$\Leftrightarrow \quad s > -\frac{\ln\ n_0}{\ln(1 - 1/h)} < h \cdot \ln(n_0) = h \cdot \ln\left(\binom{k}{t} h\right). \tag{2.5}$$

Thus, we need a maximum of $h \cdot \ln(h) + h \cdot \ln\binom{k}{t} + 1$ steps, which is in $O(h \log h + h \log k) = O(\log k)$. □

Using Theorem 2.2.2 it is possible to show a result regarding VCAs, first given in [73] but based on different arguments than ours.

Corollary 2.2.3 *Let $\bar{t}$, $v \in \mathbb{N}$ be fixed with $v_i \leq v$ for all $i \in \{1, \ldots, k\}$ as well as $|I| \leq \bar{t}$ for all $I \in \mathcal{I}$, then $\mathsf{VCAN}(\mathcal{I}, k, (v_1, \ldots, v_k))$ belongs to $O(\log k)$,*

$$\mathsf{VCAN}(\mathcal{I}, k, (v_1, \ldots, v_k)) \in O(\log k).$$

Proof. Let $t := \max\{|I| : I \in \mathcal{I}\} \leq \bar{t}$ denote the highest cardinality of any column selection defined by $\mathcal{I}$, and $v := \max\{v_i : i \in \{1, \ldots, k\}\}$ the maximal alphabet size. From Theorem 2.2.2 we know that $\mathsf{CAN}(t, k, v) \in O(log\ k)$. Now every $\mathsf{CA}(N; t, k, v)$ yields a $\mathsf{VCA}(N; \mathcal{I}, k, (v_1, \ldots, v_k))$ by replacing all the entries in column i that rise from the set $[v] \setminus [v_i]$ with arbitrary elements of $[v_i]$. Hence, we get an array over $(v_1, \ldots, v_k)$. Since $\forall I \in \mathcal{I}$ we have $|I| \leq t$ the such constructed array is a $\mathsf{VCA}(N; \mathcal{I}, k, \mathbf{v})$ with $N \in O(log\ k)$ number of rows. □

Required Complexity Concepts

A short introduction to the terms and notions related to computational complexity used in the following can be found for example in Chapter 11 of [162]. Let us briefly mention that we use the binary encoding for integers n representing them with $\lceil \log_2 n \rceil \in O(\log n)$ bits. Additionally, we also use the notation $A \leq_P^T B$ to denote that problem A is polynomial time Turing reducible to problem B. Informally, in that case we say there exists a poly-time reduction (from A to B). In case that also B is poly-time Turing reducible to A, i.e. A and B are poly-time Turing equivalent, we denote this by $A \equiv_P^T B$.

2.2.1 Complexity Problems of CAs and their Relations

Let us first consider the special case of CAs. We already mentioned in the introduction informally the problems that arise with these structures. In the following we want to concretize these notions, as well as others in the form of problem statements.

Problem 2.2.4 (optimal $\mathsf{CA}_{t,v}$: Generation ($genOCA_{t,v}$)) *For arbitrary but fixed values t and v, find an optimal $\mathsf{CA}(N; t, k, v)$, i.e. one with $N = \mathsf{CAN}(t, k, v)$, having as input k and $\binom{\{k\}}{t}$. In summary, we have:*
Input: k, $\binom{\{k\}}{t}$.
Question: Find an optimal $\mathsf{CA}(N; t, k, v)$, i.e. one with $N = \mathsf{CAN}(t, k, v)$.

Problem 2.2.5 (optimal $\mathsf{CA}_{t,v}$: Determine Size ($detSizeOCA_{t,v}$)) *For arbitrary but fixed values t and v, determine $\mathsf{CAN}(t, k, v)$, i.e. the smallest integer N for which a $\mathsf{CA}(N; t, k, v)$ exists, having as input k and $\binom{\{k\}}{t}$. In summary, we have:*
Input: k, $\binom{\{k\}}{t}$.
Question: Find $\mathsf{CAN}(t, k, v)$.

Problem 2.2.6 (optimal $\mathsf{CA}_{t,v}$: Decide Size ($decSizeOCA_{t,v}$)) *For arbitrary but fixed values t and v, decide whether $\mathsf{CAN}(t, k, v) \leq N$ does hold, having as input k, $\binom{\{k\}}{t}$ and N. In other words, decide if a $\mathsf{CA}(N; t, k, v)$ exists, or not. In summary, we have:*
Input: k, $\binom{\{k\}}{t}, N$.
Question: Decide whether $\mathsf{CAN}(t, k, v) \leq N$.

We will see in the sequel why the input $\binom{\{k\}}{t}$ is needed, at least for our argumentation, regarding the complexity results presented.

We refer to the entity of Problems 2.2.4 - 2.2.6 as CA-Problems if no further specification is needed.

We can find poly-time reductions of **Problem i+1** to **Problem i**, for $\mathbf{i} \in \{2.2.4, 2.2.5\}$. For the reduction of decision problems to optimization problems see for example also [163]. We summarize this in the following Theorem 2.2.7.

Theorem 2.2.7 *For arbitrary but fixed t and v, it holds that*

(i) $decSizeOCA_{t,v} \leq_P^T detSizeOCA_{t,v} \leq_P^T genOCA_{t,v}$.

(ii) $decSizeOCA_{t,v} \equiv_P^T detSizeOCA_{t,v}$.

Proof. *(i)* To give a Turing reduction from $decSizeOCA_{t,v}$ to $detSizeOCA_{t,v}$, we first have to show that there exists an algorithm that transforms an instance of the problem $decSizeOCA_{t,v}$, in time polynomial in the size of the input $k, \binom{\{k\}}{t}, N$, to an instance of the problem $detSizeOCA_{t,v}$. Second we show that a solution (or answer), attained from a one-step oracle for $detSizeOCA_{t,v}$, can be transformed to a solution for $decSizeOCA_{t,v}$, again in time polynomial in the size of the original input $k, \binom{\{k\}}{t}, N$. We may depict this by the left hand side of the diagram below, where the arrows $\xrightarrow{p}$ in the upper sequence represent poly-time algorithms that transform the starting problem together with its input to an instance of the target problem. The arrows $\xleftarrow{p}$ in the lower sequence represent poly-time algorithms that transform a solution $(Sol^{(i+1)})$ of one problem to a solution $(Sol^{(i)})$ of another problem.

$$\begin{array}{ccccc} decSizeOCA_{t,v} & \longrightarrow & detSizeOCA_{t,v} & \longrightarrow & genOCA_{t,v} \\ \downarrow & & \downarrow & & \downarrow \\ Sol & \longleftarrow & Sol' & \longleftarrow & Sol'' \end{array}$$

To show $decSizeOCA_{t,v} \leq_P^T detSizeOCA_{t,v}$ we only need to request $\mathsf{CAN}(t,k,v)$ from an (one-step) oracle for $detSizeOCA_{t,v}$, and compare the integer, that is the answer of the oracle to the input N of $decSizeOCA_{t,v}$. This decides whether $\mathsf{CAN}(t,k,v) \leq N$ holds, in $O(\log N)$ steps.

To prove $detSizeOCA_{t,v} \leq_P^T genOCA_{t,v}$ (see the right hand side of the diagram above) the input to $detSizeOCA_{t,v}$ is submitted as a query to an oracle for $genOCA_{t,v}$, which returns an optimal CA, say A, for the parameters (t,k,v). From Theorem 2.2.2 we know that the number of rows of A is in $O(\log\ k)$, counting the $\mathsf{CAN}(t,k,v)$ rows of A can hence be done in $O(\log\ k)$ steps. The integer $\mathsf{CAN}(t,k,v)$ can be returned as a solution for the $detSizeOCA_{t,v}$ problem. The previously described algorithm is in poly-time, as k, being part of the input, has a size of $\lceil \log k \rceil$.

Algorithm 1 DET2DEC

1: INPUT: $k, \binom{\{k\}}{t}$
2: $N = 1$
3: **while** $decSizeOCA_{t,v}\left(k, \binom{\{k\}}{t}, N\right)$ is false **do**
4: $\quad N \leftarrow N + 1$
5: **end while**
6: **return** N

(*ii*) To establish $detSizeOCA_{t,v} \equiv_P^T decSizeOCA_{t,v}$, we are left to show $detSizeOCA_{t,v} \leq_P^T decSizeOCA_{t,v}$. Consider the following Algorithm 1, in which an oracle for $decSizeOCA_{t,v}$ is called incrementally. Again from Theorem 2.2.2 we know that the while loop in line 4 of Algorithm 1 breaks after $O(\log k)$ steps. The return value N is an answer to $detSizeOCA_{t,v}$. □

Theorem 2.2.8 *$decSizeOCA_{t,v}$ is a member of* NP.

Proof. Suppose we have the input $k, \binom{\{k\}}{t}, N$ to the $decSizeOCA_{t,v}$ problem and $\mathsf{CAN}(t,k,v) \leq N$ holds. We will show that there exists a polynomially balanced witness (c.f. [162]), so that this statement can be checked in poly-time, i.e. time polynomial in the size of the input $k, \binom{\{k\}}{t}, N$. Encoding the appearing integers and sets of integers in the input, the size of the input is lower bounded by $\log k + \binom{k}{t} + \log N$.

As a witness we assume that an optimal $\mathsf{CA}(M;t,k,v)$, say A, with $M = \mathsf{CAN}(t,k,v)$ rows is given. An encoding of the size $O(kM \log v) = O(k \log k)$ of such an array can be done, which is polynomially bounded in the size of the input. We have to check in polynomial time that

(i) A is a $\mathsf{CA}(M;t,k,v)$

(ii) $M \leq N$

where ((ii)) can be done in a number of steps that is in $O(\log N)$. Focusing on ((i)) we know from Theorem 2.2.2 that $\mathsf{CAN}(t,k,v) \in O(log\ k)$. Therefore, iterating over the rows of A takes $O(log\ k)$ time. To check whether all the $v^t\binom{k}{t}$ t-way interactions in $\mathbb{T}_{v,k,t}$ are covered by the rows of A, we iterate once over the rows of A, and for each row we record which t-way interactions are covered. This takes $O\big(\log(k)\binom{k}{t}v^t\big) = O\big(\log(k)\binom{k}{t}\big)$ time (recall that v and t are constant). Finally, we can check that all t-way interactions are covered in $O\big(\binom{k}{t}v^t\big) = O\big(\binom{k}{t}\big)$ time, which is dominated by $O\big(\log(k)\binom{k}{t}\big)$. Hence, we can verify A being a $\mathsf{CA}(M;t,k,v)$ in polynomial time in the size of the input. □

Remark 2.2.9 *The proof of Theorem 2.2.8 shows why the set* $\binom{\{k\}}{t}$ *is considered as input to the problem* $decSizeOCA_{t,v}$*, since otherwise we would not be able to check the witness* A *in time polynomial in the input size, the way we did. In other words, if we consider only the integers* k *and* N *as input, we would need to find an algorithm capable of checking* $\mathsf{CAN}(t,k,v) \leq N$ *in time polynomial in* $\log k + \log N$*.*

To the best of my knowledge, until today there is no general method known for checking $\mathsf{CAN}(t,k,v) \leq N$ other than finding an array with at most N rows, that is also a CA for the parameters (t,k,v), and checking the latter for being valid by checking the occurrence of all t-way interactions, by iterating over them. Note that for specific parameters of CAs there are some theoretical constructions that can allow for a more efficient method, see [101], [106]. Moreover, notice that the verification of CAs is subject to current research (for example, [164] use parallel and grid computing methods).

Similarly this discussion shows why the parameters v and t are fixed in these problem formulations, as an optimal $\mathsf{CA}(N;t,k,v)$ has at least v^t rows, which is already exponential in $\log v$ and double exponential in $\log t$.

Remark 2.2.10 *It is also possible to formulate respective problems to CA-Problems that also have* t *and* v *as input, e.g.* $decSizeOCA(t,k,v,N)$ *as generalization of* $decSizeOCA_{t,v}$*. Generalizing the problems in such a way yields harder problems, as we can always find reductions by passing the input as well as the constants to an oracle for the target problem, establishing e.g.* $decSizeOCA_{t,v} \leq_P^T decSizeOCA(t,k,v,N)$*.*

Complexity Results for Binary CAs of Strength Two

For the case of binary CAs of strength $t = 2$ the covering array numbers $\mathsf{CAN}(2,k,2)$ are known for all values of k. We summarize this result, discovered by several authors (e.g. [100], [99]) independently in the beginning of the 1970s, in the following theorem.

Theorem 2.2.11 *For every* $k \geq 1$ *it holds that*

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

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* 0*-entry in the first position, and* $\lceil N/2 \rceil$ 1*-entries in total.*

Although these results are known for several years, they have never been used to establish membership in P of binary CA-Problems for strength $t = 2$. With the help of this result, we can establish the following Corollaries regarding binary CA-Problems for strength $t = 2$ and their membership in P.

Corollary 2.2.12 *$detSizeOCA_{2,2}$ is a member of* P.

Proof. From Theorem 2.2.2 we know that $\mathsf{CAN}(2,k,2) \in O(\log k)$. Hence, we can determine $\mathsf{CAN}(2,k,2)$ by iteratively increasing the value of $N=1$ by one, until $k \leq \binom{N-1}{\lceil \frac{N}{2} \rceil}$, in $O(\log k)$ steps. As the computation of $\binom{N-1}{\lceil \frac{N}{2} \rceil}$ can be done in poly-time in $\log k$, the assertion holds. □

An immediate consequence of Corollary 2.2.12 and Theorem 2.2.7 is that also $decSizeOCA_{2,2} \in \mathsf{P}$. Further, we can show the following.

Corollary 2.2.13 *$genOCA_{2,2}$ is a member of* P.

Proof. We describe an algorithm that constructs an optimal $\mathsf{CA}(N,2,k,2)$ in poly-time. First we compute $N = \mathsf{CAN}(2,k,2)$. Corollary 2.2.12 shows that this is possible in poly-time. We now construct an optimal $\mathsf{CA}(N;2,k,2)$ analogue to the optimal $\mathsf{CA}(N;2,\binom{N-1}{\lceil \frac{N}{2} \rceil},2)$ as described in Theorem 2.2.11, but only generate the first k columns. To show that this can actually be done in poly-time, we use a known algorithm for exhaustive generation of subsets of fixed cardinality of a finite set. More specifically, after we add the first row of zeroes, the remaining subarray can be built by listing column wise the characteristic vectors of the $\lceil N/2 \rceil$-subsets of an $(N-1)$-set in lexicographical order, until k columns have been listed. For the generation of these subsets, we first list the lexicographically smallest, namely $\{1,2,\ldots,\lceil N/2 \rceil\}$, yielding a first column of the subarray containing $\lceil N/2 \rceil$ entries of 1s followed by $(N-1-\lceil N/2 \rceil)$ entries of 0s. Each subsequent $\lceil N/2 \rceil$-subset of $\{1,2,\ldots,N-1\}$ can be generated from the previous by a lexicographical successor algorithm, for example the Algorithm 2.6 kSubsetLexSuccessor of [165], which runs in $O(N)$ time. By repeating this algorithm $k-1$ times we can generate the desired subsets in $O(kN)$ time. The translation of these subsets to their characteristic vectors can also be done in $O(kN)$ time. Since $N \in O(\log k)$, the whole construction takes time in $O(k \log k)$. □

2.2.2 Complexity Problems of MCAs and their Relations

Analogue to the CA-Problems presented in Subsection 2.2.1, we can define the following MCA-Problems for MCAs.

Problem 2.2.14 (optimal $\mathsf{MCA}_{t,v}$: Generation ($genOMCA_{t,v}$)) *For arbitrary but fixed values t and v, find an optimal $\mathsf{MCA}(N;t,k,(v_1,\ldots,v_k))$, i.e. one with $N = \mathsf{MCAN}(t,k,(v_1,\ldots,v_k))$, having as input k, $\mathbf{v}=(v_1,\ldots,v_k)$ with $v_i \leq v\ \forall i \in \{1,\ldots,k\}$ and $\binom{\{k\}}{t}$. In summary, we have:*
Input: k, $(v_1,\ldots,v_k)$, $\binom{\{k\}}{t}$.
Question: Find an optimal $\mathsf{MCA}(N;t,k,\mathbf{v})$, i.e. one with $N = \mathsf{MCAN}(t,k,\mathbf{v})$.

We want to highlight that v is fixed, hence it can be considered part of the *name* of the problem, and that the alphabet sizes v_i, which are part of the input, are upper bounded by v.

Problem 2.2.15 (optimal $\mathsf{MCA}_{t,v}$: Determine Size ($detSizeOMCA_{t,v}$)) *For arbitrary but fixed values t and v, determine $\mathsf{MCAN}(t,k,(v_1,\ldots,v_k))$, i.e. the smallest integer N for which an $\mathsf{MCA}(N;t,k,(v_1,\ldots,v_k))$ exists having as input k, $\mathbf{v}=(v_1,\ldots,v_k)$ with $v_i \leq v\ \forall i \in \{1,\ldots,k\}$ and $\binom{\{k\}}{t}$. In summary, we have:*
Input: k, $(v_1,\ldots,v_k)$, $\binom{\{k\}}{t}$.
Question: Find $\mathsf{MCAN}(t,k,\mathbf{v})$.

Problem 2.2.16 (optimal $\mathsf{MCA}_{t,v}$: Decide Size ($decSizeOMCA_{t,v}$)) *For arbitrary but fixed values t and v, decide whether $\mathsf{MCAN}(t,k,(v_1,\ldots,v_k)) \leq N$ holds, having as input k, $\binom{\{k\}}{t}$, $\mathbf{v}=(v_1,\ldots,v_k)$ with $v_i \leq v\ \forall i \in \{1,\ldots,k\}$ and N. In other words, decide if an $\mathsf{MCA}(N;t,k,(v_1,\ldots,v_k))$ exists, or not. In summary, we have:*
Input: k, $\binom{\{k\}}{t}$, $(v_1,\ldots,v_k)$, N.
Question: Decide whether $\mathsf{MCAN}(t,k,\mathbf{v}) \leq N$.

Similarly, as in Subsection 2.2.1 we can establish the following results for the MCA-Problems.

Theorem 2.2.17 *For arbitrary but fixed t and v, it holds that*

(i) $decSizeOMCA_{t,v} \leq_P^T detSizeOMCA_{t,v} \leq_P^T genOMCA_{t,v}$.

(ii) $decSizeOMCA_{t,v} \equiv_P^T detSizeOMCA_{t,v}$.

Sketch of proof. The proof of Theorem 2.2.7 can be generalized straightforward for MCAs, as the key observation used, the Logarithmic Guarantee (Theorem 2.2.2), also holds for MCAs. ■

Theorem 2.2.18 *$decSizeOMCA_{t,v}$ is a member of NP.*

Sketch of Proof. A similar argument as the one used in the proof of Theorem 2.2.8 can be used to show $decSizeOMCA_{t,v}$ being a member in NP, as an optimal $\mathsf{MCA}(M;t,k,(v_1,\ldots,v_k))$ has $M \in O(\log k)$ rows and can be verified to be an $\mathsf{MCA}(M;t,k,(v_1,\ldots,v_k))$ in

$$\Big(\sum_{\substack{I\subseteq\{1,\ldots,k\}\\|I|=t}}\prod_{j\in I} v_j\Big)O(\log\ k) = O\Big(\binom{k}{t}\log\ k\Big)$$

time, checking that all $\mathbf{v}$-ary t-way interactions are covered (where we use that $\prod_{j\in I} v_j \leq v^t$, is bounded by a constant, as $\bar{t}$ and v are fixed), which is polynomial in the size of the input k, $\binom{\{k\}}{t}$, N, $(v_1,\ldots,v_k)$, and lower bounded by $\log k + \binom{k}{t} + \log N$. ■

2.2.3 Complexity Problems of VCAs and their Relations

Analogue to the MCA-Problems presented in Subsection 2.2.2, we can define the following VCA-Problems for VCAs.

Problem 2.2.19 (optimal $\mathrm{VCA}_{\bar{t},v}$: Generation ($genOVCA_{\bar{t},v}$)) *For arbitrary but fixed values $\bar{t}$ and v, find an optimal $\mathsf{VCA}(N;\mathcal{I},k,(v_1,\ldots,v_k))$, i.e. one with $N = \mathsf{VCAN}(\mathcal{I},k,(v_1,\ldots,v_k))$, for arbitrary input k, $\mathcal{I} \subseteq \mathscr{P}(\{1,\ldots,k\})$ with $|I| \leq \bar{t}$, $\forall I \in \mathcal{I}$ and $\mathbf{v} = (v_1,\ldots v_k)$ with $v_i \leq v$ $\forall i \in \{1,\ldots,k\}$. In summary, we have:*
Input: k, $\mathcal{I}$, $(v_1,\ldots,v_k)$.
Question: Find an optimal $\mathsf{VCA}(N;\mathcal{I},k,\mathbf{v})$, i.e. one with $N = \mathsf{VCAN}(\mathcal{I},k,\mathbf{v})$.

Once more, we want to highlight that v and $\bar{t}$ are fixed, hence they can be considered part of the *name* of the problem, and that the alphabet sizes v_i, which are part of the input, are upper bounded by v, and also the cardinality of the column selections are upper bounded by $\bar{t}$.

Problem 2.2.20 (optimal $\mathrm{VCA}_{\bar{t},v}$: Determine Size ($detSizeOVCA_{\bar{t},v}$)) *For arbitrary but fixed values $\bar{t}$ and v determine $\mathsf{VCAN}(\mathcal{I},k,(v_1,\ldots,v_k))$, i.e. the smallest integer N for which a $\mathsf{VCA}(N;\mathcal{I},k,(v_1,\ldots,v_k))$ exists, for an arbitrary input k, $\mathcal{I} \subseteq \mathscr{P}(\{1,\ldots,k\})$ with $|I| \leq \bar{t}$, $\forall I \in \mathcal{I}$ and $\mathbf{v} = (v_1,\ldots v_k)$, with $v_i \leq v$ $\forall i \in \{1,\ldots,k\}$. In summary, we have:*
Input: k, $\mathcal{I}$, $(v_1,\ldots,v_k)$.
Question: Find $\mathsf{VCAN}(\mathcal{I},k,\mathbf{v})$.

Problem 2.2.21 (optimal $\mathrm{VCA}_{\bar{t},v}$: Decide Size ($decSizeOVCA_{\bar{t},v}$)) *For arbitrary but fixed values $\bar{t}$ and v, decide whether $\mathsf{VCAN}(\mathcal{I},k,(v_1,\ldots,v_k)) \leq N$ holds, having as input k, $\mathcal{I} \subseteq \mathscr{P}(\{1,\ldots,k\})$ with $|I| \leq \bar{t}$, $\forall I \in \mathcal{I}$, $\mathbf{v} = (v_1,\ldots v_k)$ with $v_i \leq v$ $\forall i \in \{1,\ldots,k\}$ and N. In other words, decide if a $\mathsf{VCA}(N;\mathcal{I},k,(v_1,\ldots,v_k))$ exists, or not. In summary we have:*
Input: $k,\mathcal{I},(v_1,\ldots v_k),N$.
Question: Decide whether $\mathsf{VCAN}(\mathcal{I},k,\mathbf{v}) \leq N$ holds.

Analogously to the results for CAs and MCAs, we can show the following.

Theorem 2.2.22 *For arbitrary but fixed t and v, it holds that*

(i) $decSizeOVCA_{\bar{t},v} \leq_P^T detSizeOVCA_{\bar{t},v} \leq_P^T genOVCA_{\bar{t},v}$.

(ii) $decSizeOVCA_{\bar{t},v} \equiv_P^T detSizeOVCA_{\bar{t},v}$.

Sketch of Proof. The proof of the statements is similar to the proof of Theorem 2.2.17, as it is likewise a generalization of the arguments in the proof of Theorem 2.2.7, as Corollary 2.2.3 establishes the logarithmic guarantee for VCAs. ■

Theorem 2.2.23 *$decSizeOVCA_{\bar{t},v}$ is member of* NP.

Proof. Suppose $\mathsf{VCAN}(\mathcal{I}, k, (v_1, \ldots, v_k)) \leq N$. We show that there exists a polynomially balanced witness such that this statement can be checked in poly-time, i.e. time polynomial in the size of the input $k, \mathcal{I}, (v_1, \ldots, v_k), N$. Encoding the integers occurring in the input as binary numbers, the size of the input is lower bounded by $\log k + |\mathcal{I}| + k + \log N$.

Let A be an optimal $\mathsf{VCA}(M; \mathcal{I}, k, (v_1, \ldots, v_k))$. We show that we can check in poly-time that A is in fact a $\mathsf{VCA}(M; \mathcal{I}, k, \mathbf{v})$, which by checking $M \leq N$ ensures $\mathsf{VCAN}(\mathcal{I}, k, \mathbf{v}) \leq N$.

From Corollary 2.2.3 we know that $\mathsf{VCAN}(\mathcal{I}, k, (v_1, \ldots, v_k)) \in O(\log k)$. Iterating over the rows of A takes $O(\log k)$ time. To check whether all $\mathbf{v}$-ary $|I|$-way interactions, for all $I \in \mathcal{I}$, are covered by the rows of A, we iterate once over the rows of A, and for each row we record which are covered. This takes $O(|\mathcal{I}|v^{\bar{t}} \log k) = O(|\mathcal{I}| \log k)$ time (recall that v and $\bar{t}$ are fixed). Finally, checking whether all $\mathbf{v}$-ary $|I|$-way interactions, for all $I \in \mathcal{I}$, are covered takes $O(|\mathcal{I}|v^{\bar{t}}) = O(|\mathcal{I}|)$ (as v and $\bar{t}$ are constants) time which is dominated by $O(|\mathcal{I}| \log k)$. Hence, we can verify A being a $\mathsf{VCA}(M; \mathcal{I}, k, (v_1, \ldots, v_k))$ in time polynomial in size of the input, thus establishing membership in NP. □

Remark 2.2.24 *Compared to the CA- and MCA-Problems for the VCA-Problems, defined in this subsection, the specification of $\mathcal{I}$ in the input seems to be more natural than the specification of $\binom{\{k\}}{t}$ as input to the respective CA- and MCA-Problems. Yet, as the CA- and MCA-Problems are special cases of VCA-Problems, this choice seems to be justified. In specifying the column selections of interest for these problems as part of the input, we may reduce the complexity of the problems by lengthening the input. This technique is referred to as* padding *in complexity theory (see for example, [166]). In retrospect, looking at Remark 2.2.9 this is what enabled us to establish membership in* NP *for these problems.*

2.2.4 Complexity Problems of Bshouty-Seroussi (BS) Arrays: Their Relations and Effects

Translating the discrete structures Bshouty and Seroussi consider in their work [30] to our terminology, they consider arrays with L columns over the binary alphabet $\{0, 1\}$. For $2 \leq d \leq L$ they consider (L, d)-sets, which are subsets of $\{1, \ldots, L\}$ of cardinality d. (L, d)-sets serve the same purpose as the column selections $I \in \mathcal{I}$ of VCAs, i.e. the specification of, in this case d, columns. The arrays considered in [30] (we denote them by $\mathsf{BS}(L, d, Y)$, since there is no notion introduced therein) are binary arrays with L columns, with the property that given a set Y of (L, d)-sets, each binary d-way interaction $\{(p_1, x_1), \ldots, (p_d, x_d)\} |\ (p_1, \ldots, p_d) \in Y, (x_1, \ldots, x_d) \in \{0, 1\}^d\}$, is covered by at least one row of the array. We can thus define BS arrays using the already introduced notation as follows.

Definition 2.2.25 *The arrays* $\mathsf{BS}(L, d, Y)$ *are special VCAs, namely* $\mathsf{VCA}(N; \mathcal{I}, k, (2)_{i=1}^{k})$ *with* $\mathcal{I} = Y$, $k = L$ *and* $|I| = d$, $\forall I \in \mathcal{I}$.

Next we formalize some problems mentioned in [30], which we will collectively refer to as BS-Problems when no further specification is needed.

Problem 2.2.26 (optimal BS_d: Generation ($genBS_d(L,Y)$)) *For an arbitrary but fixed value d, find an optimal $\mathsf{BS}(L,d,Y)$, i.e. one with the smallest number of rows possible, having as input L and Y. In summary, we have:*
Input: L, Y.
Question: Find an optimal $\mathsf{BS}(L,d,Y)$, i.e. one with the smallest number of rows.

Problem 2.2.27 (optimal BS_d: Determine Size ($detSizeBS_d(L,Y)$)) *For an arbitrary but fixed value d, find the smallest number of rows for which a $\mathsf{BS}(L,d,Y)$ exists, having as input L and Y. In summary, we have:*
Input: L, Y.
Question: Find the smallest number of rows for which a $\mathsf{BS}(L,d,Y)$ exists.

Problem 2.2.28 (optimal BS_d: Decide Size ($decSizeBS_d(L,Y,N)$)) *For an arbitrary but fixed value d, decide whether a $\mathsf{BS}(L,d,Y)$ with N rows exists, having as input L,Y and N. In summary, we have:*
Input: L, Y, N.
Question: Decide whether there exists a $\mathsf{BS}(L,d,Y)$ with N rows.

Problem 2.2.29 (optimal $BS_{d,n}$: Decide fixed Size ($decFixedSizeBS_{d,n}(L,Y)$)) *For arbitrary but fixed values d and n, decide whether a $\mathsf{BS}(L,d,Y)$ with n rows exists, having as input L and Y. In summary, we have:*
Input: L,Y
Question: Decide whether there exists a $\mathsf{BS}(L,d,Y)$ with n rows.

As was the case for CAs, MCAs and VCAs, we can establish the following connections for BS-Problems, some of which are also mentioned in [30].

Theorem 2.2.30 *For arbitrary but fixed d and n, we have $decFixedSizeBS_{d,n}(L,Y) \leq_P^T decSizeBS_d(L,Y,N) \equiv_P^T detSizeBS_d(L,Y) \leq_P^T genBS_d(L,Y)$.*

Sketch of Proof. To establish $decFixedSizeBS_{d,n}(L,Y) \leq_P^T decSizeBS_d(L,Y,N)$ consider an algorithm that takes n together with the inputs L and Y to make a query to an oracle for $decSizeBS_d$. The answer to $decSizeBS_d(L,Y,n)$ is an answer to $decFixedSizeBS_{d,n}(L,Y)$.

The proofs for the other statements are similar to the ones in the proof of Theorem 2.2.7, where it is worthwhile to mention that a Logarithmic Guarantee, as the one in Theorem 2.2.2, also holds for BS arrays, since any $\mathsf{CA}(N;d,L,2)$ is a $\mathsf{BS}(L,d,Y)$ for any Y. ■

The previously mentioned Logarithmic Guarantee for BS arrays establishes also the following membership property.

Theorem 2.2.31 *$decSizeBS_d$ is member in* NP.

Sketch of Proof. Similar to the proof of Theorem 2.2.8, replacing $\binom{\{k\}}{t}$ with Y. ■

The first result on hardness of complexity problems related to classes of covering arrays we want to review is one of Seroussi and Bshouty, as in [30] they show the following theorem.

Theorem 2.2.32 (Theorem 8, [30]) *For an arbitrary but fixed d it is an* NP*-complete problem to decide whether there exists a $BS(L, d, Y)$ with 2^d rows. In particular, we have:*

$$decSizeBS_{d,2^d}(L, Y) \text{ is NP-complete.}$$

The consequences of the results of [30] together with Theorem 2.2.30 above can be summarized in the following corollary.

Corollary 2.2.33 *For an arbitrary but fixed d, we have:*

(i) $decSizeBS_{d,2^d}(L, Y)$ *is* NP*-complete*

(ii) $decSizeBS_d(L, Y, N)$ *is* NP*-complete*

(iii) $detSizeBS_d(L, Y)$ *is* NP*-hard*

(iv) $genBS_d(L, Y)$ *is* NP*-hard*

Effects of the BS-Problems Hardness Results to VCA-Problems

By now it becomes apparent that $BS(L, d, Y)$ arrays are special VCAs over the binary alphabet. This immediately yields *lower bounds* on the complexity of the corresponding VCA-Problems, since each one of the BS-Problems can be reduced in polynomial time to the corresponding complexity problem regarding VCAs.

Theorem 2.2.34 *For arbitrary but fixed d and $v = 2$, it holds that*

$$decSizeBS_d \leq_P^T decSizeOVCA_{d,2}.$$

Proof. As a $\mathsf{BS}(L, d, Y)$ is a $\mathsf{VCA}(N; Y, k, (2)_{i=1}^k)$, to solve $decSizeBS_d(L, Y, N)$, we simply call an oracle for $decSizeOVCA_{d,2}(L, Y, N)$ and return its answer. □

Now from Theorem 2.2.34 together with the results from Subsection 2.2.3 we can establish the following complexity results for the hardness of binary VCA-Problems.

Corollary 2.2.35 *For arbitrary but fixed $\bar{t} \geq 2$ and $v = 2$ it holds that*

(i) $decSizeOVCA_{\bar{t},2}$ *is NP-complete.*

(ii) $detSizeOVCA_{\bar{t},2}$ *is NP-hard.*

(iii) $genOVCA_{\bar{t},2}$ *is NP-hard.*

Remark 2.2.36 *Note that from Corollary 2.2.35 we cannot directly, at least not in an obvious way, deduce any result for the complexity of non-binary VCA-Problems, as there is no known reduction from e.g. $decSizeOVCA_{\bar{t},2}$ to $decSizeOVCA_{\bar{t},v}$ for $v > 2$.*

Yet, as already mentioned in Remark 2.2.10, it is possible to consider more general problems, where the alphabet size v is part of the input. Moreover, we apply some padding, having 2^v as part of the input, which increases the size of the input by v (recall that we use a binary encoding for integers).

Problem 2.2.37 (optimal $VCA_{\bar{t}}$: Decide Size ($decSizeOVCA_{\bar{t}}$)) *For an arbitrary but fixed value $\bar{t}$, decide whether $\mathsf{VCAN}(\mathcal{I}, k, (v_1, \ldots, v_k)) \leq N$ holds, having as input k, v, 2^v, $\mathcal{I} \subseteq \mathscr{P}(\{1, \ldots, k\})$ with $|I| \leq \bar{t}$, $\forall I \in \mathcal{I}$, $\mathbf{v} = (v_1, \ldots v_k)$ with $v_i \leq v$ $\forall i \in \{1, \ldots, k\}$ and N. In other words, decide if a $\mathsf{VCA}(N; \mathcal{I}, k, (v_1, \ldots, v_k))$ exists, or not. In summary we have:*
Input: $k, v, 2^v, \mathcal{I}, (v_1, \ldots v_k), N$.
Question: Decide whether $\mathsf{VCAN}(\mathcal{I}, k, \mathbf{v}) \leq N$ holds.

Since v is part of the input to Problem 2.2.37, we can establish the following relation between $decSizeOVCA_{\bar{t},2}$ and $decSizeOVCA_{\bar{t}}$.

Theorem 2.2.38 *For arbitrary but fixed $\bar{t}$, it holds that*

$$decSizeOVCA_{\bar{t},2} \leq_P^T decSizeOVCA_{\bar{t}}.$$

Proof. Any input to $decSizeOVCA_{\bar{t},2}$ can simply be passed to an oracle for $decSizeOVCA_{\bar{t}}$ additionally specifying $v = 2$ for the input of the latter. □

Only now we can establish results on the NP-hardness of a (general) VCA problem. Analogue to Problem 2.2.37 we can further introduce the corresponding **Determine Size** ($detSizeOVCA_{\bar{t}}$) and **Generation** ($genOVCA_{\bar{t}}$) Problem. For these problems we get the same relations as for the VCA problems where v was fixed as part of the name of the problem. This, is summarized in the following theorem.

Theorem 2.2.39 *For arbitrary but fixed $\bar{t}$ we have*

(i) $decSizeOVCA_{\bar{t}} \leq_P^T detSizeOVCA_{\bar{t}} \leq_P^T genOVCA_{\bar{t}}$.

(ii) $decSizeOVCA_{\bar{t}} \equiv_P^T detSizeOVCA_{\bar{t}}$.

Further, we have

(iii) $decSizeOVCA_{\bar{t}}$ *is NP-complete.*

(iv) $detSizeOVCA_{\bar{t}}$ *is NP-hard.*

(v) $genOVCA_{\bar{t}}$ *is NP-hard.*

Sketch of proof. Showing $decSizeOVCA_{\bar{t}} \leq_P^T detSizeOVCA_{\bar{t}}$ is similar to the proof of Theorem 2.2.22, as it is likewise a generalization of the arguments in the proof of Theorem 2.2.7. The case is similar for $detSizeOVCA_{\bar{t}} \leq_P^T genOVCA_{\bar{t}}$, where we have to pay additional attention: Since now v and 2^v are part of the input, factors of v cannot be disregarded any longer when iterating over the rows of an optimal VCA. However, from the inequality in (2.5) of the proof of Theorem 2.2.2 we get that $\mathsf{VCAN} \in O(v^{\bar{t}}(\log v^{\bar{t}} + \log k))$, since $h \leq v^{\bar{t}}$, which is polynomial in the size of the input (notice that $\bar{t}$ is still considered as fixed). With the same argument we can also establish $decSizeOVCA_{\bar{t}} \equiv_P^T detSizeOVCA_{\bar{t}}$ in the same way as in Theorem 2.2.7.

For (*iii*) the NP-hardness follows directly from Theorem 2.2.38 together with Corollary 2.2.35. Regarding membership in NP the argument is analogue to the one given in the proof of Theorem 2.2.23. In particular, assuming given an optimal witness VCA with a number of rows in $O(v^{\bar{t}}(\log v^{\bar{t}} + \log k))$, we can count the rows in poly-time in the size of the input and compare this integer to the input N.
The statements in (*iv*) and (*v*) follow directly from (*i*) and (*iii*). ■

The arguments in the proof of the previous theorem make apparent why we needed to consider 2^v as part of the input. To elaborate further, assume 2^v was *not* given as part of the input of these problems. The size of VCAs is lower bounded by v, which is exponential in $\log v$. For this reason the arguments in the proof of Theorem 2.2.39 which were used to establish membership of $decSizeOVCA_{\bar{t}}$ in NP, those used to reduce $detSizeVCA_{\bar{t}}$ to $genOVCA_{\bar{t}}$ and those used to reduce $detSizeVCA_{\bar{t}}$ to $decSizeOVCA_{\bar{t}}$, cannot be applied as they are exponential in $\log v$. As a consequence, as long as there is no other witness for $\mathsf{VCAN}(\mathcal{I}, k, \mathbf{v}) \leq N$, other than a witness VCA (which in this case is not even polynomially balanced), which additionally has to be verified by iterating over the alphabet (or powers of that), then there is no way for showing that such a problem is member of NP.

The Complexities of the CA-Problems Are Still Unknown

Clearly, the arrays $\mathsf{BS}(L, d, Y)$ considered in [30] can be considered a generalization of binary CAs. In particular, CAs over the binary alphabet, $\mathsf{CA}(N; d, L, 2)$, can be attained by choosing $Y = \binom{\{k\}}{t}$ to be the set of all subsets of $\{1, \ldots, L\}$ of cardinality d. In other words, we have the following equality: $\mathsf{CA}(N; t, k, 2) = \mathsf{BS}(k, t, \binom{\{k\}}{t})$.

Remark 2.2.40 *Sadly, the complexity result for BS arrays proven in [30] can not be applied directly to CAs. One way to draw a conclusion about the hardness of the CA-Problems or MCA-Problems using the results from [30], reviewed in Corollary 2.2.33, would be to provide a poly-time reduction from the respective complexity problem concerning BS arrays to one of the CA-Problems or MCA-Problems.*

Unfortunately, no such poly-time reductions have appeared in the literature thus far, and neither is provided in this thesis. It is important to note that the result of Seroussi and Bshouty (see Theorem 2.2.32) is misinterpreted or miscited all too often, as some authors use it as a reference to provide evidence for the hardness of some of the CA-Problems, MCA-Problems or VCA-Problems.

We list a number of examples of such and similar misinterpretations below. Our intention is solely to clarify the situation of the hardness of complexity problems regarding CAs. As this documents the first attempt to provide a systematic overview for complexity results regarding CAs, we will adhere to the notation introduced above to point out the mistakes that have been made by others, frequently using a different notation or terminology. Before reviewing these statements, we would like to point out that NP-completeness is defined for decision problems, not for search or ge. This well-known complexity property is also misinterpreted in several of the following works.

For example, in [167] the optimal CA computation is referred to as an NP-complete problem. In the terminology of this thesis, this implies that the $genOCA_{t,v}$ is NP-complete which unfortunately is not shown (to be more precise, the complexity of this problem is *yet* unknown). We continue with [168] where it is claimed that the problem of construction of a minimal CA is NP-complete. Again, in our terms, this would mean that $genOCA_{t,v}$ is NP-complete which as we stated earlier is not known to be true.

Further, in [169] it is stated that determining the covering array number for a certain input, in our terms $detSizeOCA_{t,v}$, is NP-hard. Finally, I am also not immune to such misunderstanding, as in [170] I stated that the computation of optimal binary covering arrays is an NP-hard problem, claiming that $genOCA_{t,2}$ is NP-hard.

In addition, in [171] the authors claim that the problem of generating a minimum test set for pair-wise testing is NP-complete, which in our terminology would mean that $genOMCA_{2,v}$ is NP-complete, which is not known. Another example of erroneous citation is given in [172], where it is claimed that calculating the minimal test suite is an NP-hard problem. Translating their notion to ours, this would mean that $genOMCA_{t,v}$ is NP-hard, for which we do not have any evidence whether it holds or not.

In [168] it is claimed that selecting a row covering the greatest number of t-tuples is NP-complete, providing [30] as evidence. However, this problem is not subject of the work in [30]. It rather relates to Problem 2.2.42, which will be discussed in Subsection 2.2.5.

On the contrary, the works in which the result of [30] is interpreted correctly are very limited in number. One example is [173], where the result of Seroussi and Bshouty is cited to provide evidence that, in our terms, the generation of optimal VCAs is an NP-hard problem. However, the statement given in [173] does not clarify if, e.g., the (maximal) alphabet size is considered part of the input or fixed (which, as shown in the previous Subsection 2.2.4, can be crucial for the complexity of VCA-Problems). Another work that interprets the result of [30] correctly is given in [70], which also corrects a result of [174] regarding the complexity of optimal MCA generation problem for $t = 2$ (the latter paper is used often a source to claim hardness of the corresponding $genOMCA_{2,v}$ problem). Unfortunately, however, the same survey ([70]) also fell victim to misinterpretations of complexity results, in particular of Problem 2.2.41, as we will see below.

The list of wrong statements about the complexity of generating (optimal) covering arrays, can grow even longer as in several works the hardness of the corresponding problems is treated as a well-known result without giving any references, see e.g. [175] or even a well-cited survey [176].

CAs on Graphs

In [177] the authors consider *covering arrays on graphs* (CAs on graphs), which can be considered a special case of VCAs, namely those $\mathsf{VCA}(N;\mathcal{I},k,(v,\ldots,v))$ with $|I|=2$ for all $I\in\mathcal{I}$. The name CAs on *graphs* is due to the representation of $\mathcal{I}$ as the set of edges of a graph on k vertices. It follows that binary CAs on graphs are exactly the same as $\mathsf{BS}(L,2,Y)$ arrays, and hence the complexity results for BS-Problems for $d=2$ apply to CAs on graphs. In general, CAs on graphs are generalizations of $BS(L,2,Y)$, allowing also for higher alphabet sizes.

Unfortunately, the complexity result in [177] is imprecise, as it states that Seroussi and Bshouty prove that *determining the existence* of an optimal binary covering array on a graph is an NP-complete problem. To clarify this even further, *deciding the existence* of an optimal binary CA on a graph, i.e. one with the smallest possible number of rows, is trivial. In particular, the set of binary CAs on a graph is a non-empty set, as the array consisting of all binary (row-)vectors of length k is a CA on any graph. Considering the set of all binary CAs on a given graph G, having at most 2^k rows, we can *always* find an array with the smallest number of rows in this finite non-empty set.

One exemplary conclusion from the results of Seroussi and Bshouty [30] is that since $genBS_2(L,Y)$ is NP-hard, the search problem of finding an optimal binary CA on a graph with input k and G is also NP-hard. The other hardness results for binary CAs on graphs can be attained by considering the respective results for BS-Problems (Theorem 2.2.33) for $d=2$.

We would like to summarize the relation of the various complexity problems and classes of covering arrays considered so far in the following diagram (c.f. Figure 2.1), which gives an overview of the current state of the art. When viewing this diagram, one needs to keep in mind that it does not reflect strengths nor alphabet sizes. For example, BS arrays are only defined over binary alphabets, for arbitrary strength; whereas CAs on graphs (denoted as CA(G)) are only defined for strength 2, but for arbitrary alphabet sizes.

2.2.5 Other Complexity Problems Related to Covering Arrays

In this subsection, our aim is to review and discuss other related complexity problems regarding covering arrays, i.e. such that do not necessarily have tight relations with the CA-, MCA-, and VCA-Problems introduced in this section. Nevertheless, we include them here in order to provide a rigorous treatment of complexity aspects of covering arrays and improve or correct the understanding of the state of the art when applicable.

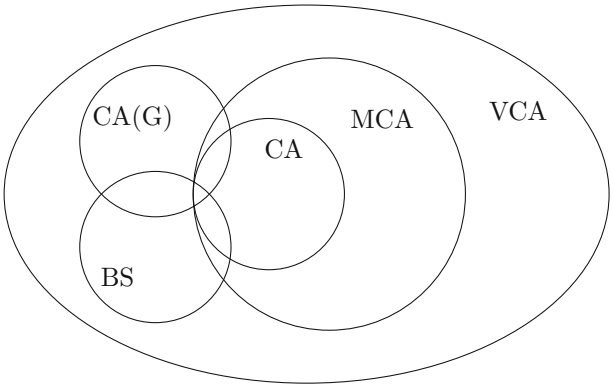


Figure 2.1: Venn diagram of structures related to CAs.

The Complexity of the Maximal Array Extension Problem Is Still Unknown

There exists a vast amount of literature on greedy and intelligent heuristic algorithms for the optimal CA generation problem. Some of these algorithms (like AETG [36] and DDA [71]) for CA generation are so called *one row at a time* algorithms, which aim to produce CAs with a *small* number of rows, by covering a *large* number t-way interactions in every row that is added to an initially empty array, growing the array one row at a time. Other algorithms rely on a similar technique, such as the algorithms belonging to the IPO family [27], which grow CAs in two dimensions, first adding a column to a CA at hand, followed by zero or more row extensions, where again in each row the strategy is to cover as many uncovered t-way interactions as possible.

Strategies for finding such rows are essential to both the run time and the quality of generated output of these algorithms. This is reason enough to elaborate on the *hardness of finding a row that covers a maximal number of tuples.* To begin the discussion related to such problems, we recall Definition 1.3.9 and Definition 1.3.10 in order to introduce following formal decisions problems.

Problem 2.2.41 (v-ary t-way interaction cover problem) *Given parameters t, k, $(v_1, \ldots, v_k)$, a set $\mathcal{T}$ of $(v_1, \ldots, v_k)$-ary t-way interactions and an integer m, decide if there is a vector $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$, such that w covers at least m of the $(v_1, \ldots, v_k)$-ary t-way interactions in $\mathcal{T}$. In summary, we have:*
Input: $t, k, (v_1, \ldots, v_k), \mathcal{T}, m$.
Question: Decide whether there is $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$ that covers at least m of the $(v_1, \ldots, v_k)$-ary t-way interactions in $\mathcal{T}$.

Problem 2.2.42 (Maximal array extension) *Given parameters $t, k, \mathbf{v} = (v_1, \ldots, v_k)$, an integer m and an array A over $(v_1, \ldots, v_k)$ as well as the set $\mathcal{U}$ of all $\mathbf{v}$-ary t-way interactions not covered by the rows of A. Decide if there is a vector $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$, such that w covers at least m elements of $\mathcal{U}$. In summary, we have:*

Input: $t, k, (v_1, \ldots, v_k), A, \mathcal{U}, m$.
Question: Decide whether there is $w \in \prod_{i=1}^{k} \{0, \ldots, v_i - 1\}$ *that covers at least* m *of the* $(v_1, \ldots, v_k)$*-ary* t*-way interactions not covered by any of the rows of* A.

Every instance given as input to Problem 2.2.42 can be translated to an instance of Problem 2.2.41, via mapping $\mathcal{U}$ to $\mathcal{T}$ and omitting A, in poly-time. It is obvious that any solution to such an instance of the $\mathbf{v}$-ary t-way interaction cover problem, is also a solution to the original instance of the array extension problem. This provides a poly-time reduction of Problem 2.2.42 to Problem 2.2.41.

Remark 2.2.43 *On the contrary, there exist instances to Problem 2.2.41 that can not be* directly *translated to instances of Problem 2.2.42, as the following example shows. The array extension Problem 2.2.42 seems to have more structure than Problem 2.2.41.*

Example 2.2.44 *Consider the following set of* $(2, 2, 2)$*-ary* 2*-way interactions*[1] $\mathcal{T} = \{(0, 0, -), (0, 1, -)\}$*, which can be considered as part of an input to Problem 2.2.41. However, there is no binary array* A *that leaves exactly the set* $\mathcal{T} = \{(0, 0, -), (0, 1, -)\}$*, of* $(2, 2, 2)$*-ary* 2*-way interactions uncovered, since every binary array not covering* $\mathcal{T}$ *does not have any row with a* 0*-entry in the first column, i.e., the array also does not cover the binary* 1*-way interaction* $(0, -, -)$*, and therefore it also does not cover the* $(2, 2, 2)$*-ary* 2*-way interactions* $(0, -, 0)$ *and* $(0, -, 1)$*. Hence* $\mathcal{T}$ *can not appear as part of an input to Problem 2.2.42.*

The authors of [178] argued that the subclass of instances to Problem 2.2.41, having binary alphabets $v_1 = \ldots = v_k = 2$ and strength $t = 2$, is NP-complete by giving a reduction from (the decision version of) MAX2SAT to it. However, their argument is formulated incorrectly. They map a formula ψ over a set of k logical variables, given as input to the MAX2SAT problem, to a set $\mathcal{T}$ of $(2)_{i=1}^{k}$-ary 2-tuples. To this end, a truth assignment of the k logical variables is encoded as a binary vector of length k, where the i-th position of the vector encodes the truth assignment of the i-th logical variable. Furthermore, a partial truth assignment of two logical variables is encoded as the corresponding $(2)_{i=1}^{k}$-ary 2-tuple. The set $\mathcal{T}$ is then defined as the set of all $(2)_{i=1}^{k}$-ary 2-tuples that falsify none of the clauses of ψ.

The claim that each $(2)_{i=1}^{k}$-ary 2-tuple in $\mathcal{T}$ satisfies a clause of ψ, and hence any binary vector of length k covering m elements of $\mathcal{T}$ corresponds to a truth assignment of the k logical variables that satisfies m clauses of ψ, does not hold. The following counter example illustrates this.

[1] We use the informal notation for $\mathbf{v}$-ary t-way interactions as introduced in the paragraph after Definition 1.3.9 and as used in Example 2.3.2.

Example 2.2.45 *Consider the formula* $\psi = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4)$*. The set of all* $(2)_{i=1}^4$*-ary 2-tuples corresponding to partial truth assignments of two variables that falsify none of the clauses of* ψ *is given by*

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

The truth assignment corresponding to the vector $(0,0,0,0)$*, i.e. the one with* $x_i \mapsto 0$ *for all* $i = 1,\ldots,4$*, covers three* $(2)_{i=1}^4$*-ary 2-tuples of* $\mathcal{T}$*, namely* $(0,-,0,-),(0,-,-,0),(-,0,-,0)$*. However, the assignment* $(x_1,x_2,x_3,x_4) \mapsto (0,0,0,0)$ *satisfies none of the clauses of* ψ*.*

Let us briefly use the argument of [178] in order to establish the following result.

Theorem 2.2.46 *The* v*-ary* t*-way interaction cover problem is* NP*-complete.*

Proof. Membership in NP holds, since for a vector w covering at least m of the $\mathbf{v}$-ary t-way interactions in $\mathcal{T}$, we can iterate once over $\mathcal{T}$, checking which $\mathbf{v}$-ary t-way interactions are covered by w, and count them. This can be done in $O(|\mathcal{T}|)$ time. Since the size of the input is lower bounded by $|\mathcal{T}|$ and the size of $\mathbf{v} = (v_1,\ldots,v_k)$, w is a polynomially balanced witness for m.

For establishing NP-hardness we give a reduction from MAX2SAT, which is an NP-complete problem [179]. To that end, consider a formula ψ in 2-CNF over a set of k logical variables. We have to decide whether there is a truth assignment that satisfies at least m clauses of ψ. To perform this task in poly-time in $|\psi|$ using an oracle for the $(2)_{i=1}^k$-ary 2-way interaction cover problem, we identify a truth assignment with a binary (row) vector of length k, via encoding the truth assignment of the i-th logical variable in position i of the vector.

From the set of all $2^2\binom{k}{2}$ binary t-way interactions $\mathbb{T}_{2,k,t}$, we define the set $\mathcal{T}_\psi$ to be the set consisting of those $(2)_{i=1}^k$-ary 2-tuples that correspond to partial truth assignments of two logical variables appearing in the same clause of ψ, so that the clause evaluates to true. For each clause of ψ there are exactly three partial thruth assignments of the appearing variables so that the clause evaluates to true, e.g. for the clause $(\neg x_1 \vee x_2)$, these are $(x_1,x_2) \mapsto (0,0)$, $(x_1,x_2) \mapsto (1,1)$ and $(x_1,x_2) \mapsto (0,1)$, which correspond to the $(2)_{i=1}^k$-ary 2-way interaction $\{(1,0),(2,0)\}$, $\{(1,1),(2,1)\}$ and $\{(1,0),(2,1)\}$. Hence the size $|\mathcal{T}_\psi|$ of $\mathcal{T}_\psi$ is in $O(|\psi|)$, and we can compute $\mathcal{T}_\psi$ in poly-time in the size of the input ψ. Now each $(2)_{i=1}^k$-ary 2-way interaction in $\mathcal{T}_\psi$ satisfies a specific clause in ψ. Further, when we consider a binary vector of length k, it can cover at most one of the three $(2)_{i=1}^k$-ary 2-way interactions that correspond to a single clause of ψ, since any two $(2)_{i=1}^k$-ary 2-way interctions contradict in their values in at least one position. So a binary vector of length k covering at least m elements of $\mathcal{T}_\psi$ corresponds to a truth assignment of the k logical variables satisfying at least m clauses of ψ. □

Remark 2.2.47 *At this point, we would like to clarify that Remark 2.2.43 does not refute the* NP*-completeness of Problem 2.2.42. Note that in order to be able to argue for the* NP*-completeness of Problem 2.2.42, one way would be to establish a poly-time reduction from Problem 2.2.41. However, as Example 2.2.44 shows, such a reduction can not be devised in a straightforward way.*

This subtle but essential difference between the two problems previously discussed has been also misinterpreted in the literature, as some authors (e.g., [70]) refer to [178] as a means to provide evidence for the NP-completeness of Problem 2.2.42.

Interestingly enough, in the case of $t = 2$, and appropriate alphabet sizes, it is NP-complete to decide if certain sets of 2-tuples can appear as part $\mathcal{U}$ of an input to an array extension problem or not, as we will see below.

Complexity Results of Covering Arrays Avoiding Forbidden Edges

Covering arrays avoiding forbidden edges (CAFEs) were introduced in [60], to which we refer the reader for their precise definition. Informally speaking, CAFEs are covering arrays of strength $t = 2$ where a set G of certain $(v_1, \ldots, v_k)$-ary 2-tuples are *not allowed* to be covered by any row of the array, i.e., their appearance is *forbidden*. All other tuples not specified by G need to be covered. Such arrays are denoted by $\mathsf{CAFE}(N; G, k, (v_1, \ldots, v_k))$. The term *avoiding edges* is used due to the representation using a k-partite graph where the forbidden $(v_1, \ldots, v_k)$-ary 2-tuples appear as edges. We include here the complexity results related to CAFEs, for the sake of providing a complete report regarding complexity results of CAs in the literature. For that purpose we restate Theorem 9 of [60], supplementing it with results from another important complexity result on CAFEs given in [119].

Theorem 2.2.48 (Theorem 9, [60] and Corollary 9, [119]) *For covering arrays avoiding forbidden edges, it holds that:*

(i) Deciding the existence of a binary k-tuple that does not cover any $(2)_{i=1}^{k}$-ary 2-way interactions of a given set G, is in P.

(ii) For arbitrary but fixed $v \geq 3$, deciding the existence of a v-ary k-tuple that does not cover any $(v)_{i=1}^{k}$-ary 2-way interactions of a given set G, is NP*-complete.*

(iii) For a given set G of $(v)_{i=1}^{k}$-ary 2-way interactions, for arbitrary but fixed $v \geq 5$, it is NP-complete to decide the existence of a $\mathsf{CAFE}(n; G, k, (v, \ldots, v))$ for any n.

(iv) For a given set G of $(v)_{i=1}^{k}$-ary 2-way interactions, for arbitrary but fixed $v \geq 2$ and an integer N, it is NP*-complete to decide the existence of a $\mathsf{CAFE}(N; G, k, (v, \ldots, v))$, i.e. whether $\mathsf{CAFEN}(G, k, (v, \ldots, v)) \leq N$.*

Point (iv) of Theorem 2.2.48 can be considered a hardness result for the decision problem of the size for CAFEs, having as input k and the graph G. Analogous to the CA-Problems we can define problems for CAFEs, like determining the smallest number of rows for which a CAFE for given k and G for an arbitrary but fixed v, exists, as well as the problem of generating such a CAFE having the smallest possible number of rows.

Like for CAs, we can reduce the decision problem of the size for CAFEs to the determination of the smallest number of rows for which a CAFE exists, in poly-time. The latter problem can be again reduced in poly-time to the *optimal* CAFE generation problem. As long as there exists a CAFE on G at all, the argument here is again analogue to the one used for CAs, using the important property from Corollary 11 in [60], that $\mathsf{CAFEN}(G_{k,v}) \in O(|E(G)|^2 + \log k)$, which implies that we can count the number of rows of an optimal CAFE in poly-time in the size of the input, which is lower bounded by $|E(G)| + \log k$. In the case that a CAFE does not exist the involved one-step oracle would return fail, in this case an answer for the smallest number of rows could be ∞ by convention.

CAs and MCAs of strength $t = 2$ can be considered special cases of CAFEs where the set of forbidden tuples G is empty. We would like to point out that complexity results for CAFEs might not be directly applicable to CAs. For example, point ((iii)) of Theorem 2.2.48 shows that even deciding the existence of CAFEs is an NP-complete problem, whereas the existence of CAs and MCAs is always ensured, e.g. by taking the set of all k-tuples over the respective alphabet. This minor point highlights even more the different nature of these problems and hence, in some cases, their different complexities.

Remark 2.2.49 *Interestingly enough, in retrospect of Problems 2.2.41 and 2.2.42, from Theorem 2.2.48, point* ((iii)) *it follows that it is* NP*-complete to distinguish, whether to a given set $\mathcal{U}$ of $(v)_{i=1}^{k}$-ary 2-tuples (for $v \geq 5$), there exists an array A, such that $\mathcal{U}$ is exactly the set of $(v)_{i=1}^{k}$-ary 2-tuples not covered by any row of A. This is due to the reason, that deciding this problem is equivalent to deciding the existence of a CAFE with a set of forbidden edges defined by $\mathcal{U}$. In other words, it is* NP*-complete to decide whether such a set $\mathcal{T}$ can appear as part of an input to Problem 2.2.42.*

The Complexity of the Covering Array Flexibility Problem

Finally, again for the sake of completeness, we want to include a result from [121]. This paper treats the problem of *post-optimization* of CAs, i.e. the starting point is a $\mathsf{CA}(N; t, k, v)$ and the goal is to reduce the number of rows whilst preserving the coverage of all $(v)_{i=1}^{k}$-ary t-tuples, thus obtaining a CA with a number of rows closer to $\mathsf{CAN}(t, k, v)$. In detail, in [121], this problem is approached by identifying *flexible positions* (also known as *don't care positions*, see e.g. [180]) in the CA.

A position in a CA is called flexible if all $\binom{k-1}{t-1}$ $(v)_{i=1}^{k}$-ary t-tuples that include this position are covered at least twice by the rows of the CA. Thus, the entry in this position can be changed without cancelling the defining property (see Definition 1.3.2) of the

CA, i.e., the value is flexible. Furthermore, a *flexible set* of positions in a CA is a set of flexible positions that can be changed simultaneously to arbitrary entries while the given array remains a CA (for further details, see [121]). In particular, whenever all positions of a row of a CA are a subset of a flexible set, i.e., the row consists entirely of flexible positions, we are free to remove this row from the array, yielding a CA with less rows.

The authors of [121] define the following problem and show that it is NP-complete, by giving a reduction from partial Latin square completion, which is known to be NP-complete [22].

Problem 2.2.50 (Covering Array Flexibility, [121]) *Given a covering array A over an arbitrary alphabet and an integer n, decide if there is a flexible set of positions in A that has a size of at least n.*
Input: A, n
Question: Decide whether A has a set of flexible positions of size at least n.

2.2.6 An Overview of the Current State of the Art

In this subsection, we aim to give an overview of the current state of the art of complexity problems for covering arrays, and at the same time enrich it with the results and statements given in this section. We deem it necessary before we embark on this task to update Figure 2.1 with the additional structures that have been defined in Subsection 2.2.5 so that we make it easier for the reader to understand the relations of the structures and their (complexity) problems.

In particular, the diagram in Figure 2.2 can be viewed in different ways: for a fixed strength $t = 2$, which makes BS collapse to a subset of CA(G); or for arbitrary $t \geq 2$, for which it must be considered that CA(G) and CAFE only contain VCAs for $t = 2$. Note that some of the visible intersections are empty (e.g. all MCAs that are BSs are also CAs).

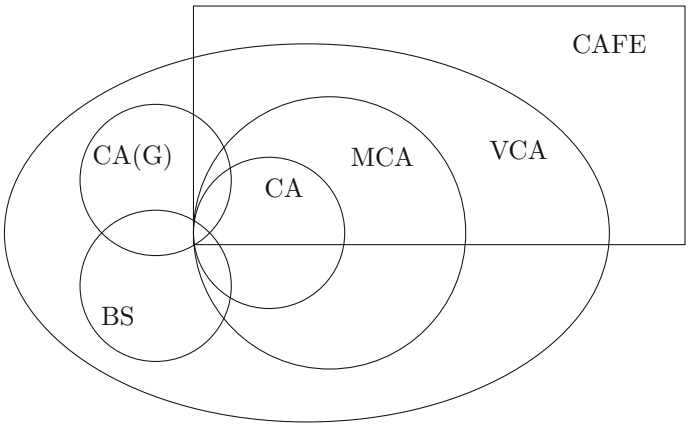


Figure 2.2: Updated Venn diagram of structures related to CAs.

Note that the parameters of any structure in the diagram in Figure 2.2 can be regarded both as input to a certain problem for this class of covering arrays and as an input to the corresponding problem of any super-class, the object lies in.

In this regard, Table 2.1 gives an overview of complexity results related to covering arrays. The complexity of determining the existence of an (optimal) object is polynomial in all cases, except for CAFEs (see [60]). The existence of the considered objects is guaranteed, which can be argued analogously to the case of CAs on graphs (see Subsection 2.2.4).

In particular, in Table 2.1 the first column refers to the respective class of covering arrays for which we surveyed, improved or corrected, where possible, the complexity of its related problems. Last but not least, the last column of this table lists the key references used to create this overview.

Classes of Covering Arrays	Decide Existence	Decide Size	Determine Size	Generation	Key References
optimal $CA_{2,2}$	P	P	P	P	[100], [99], Subsection 2.2.1
optimal $CA_{t,v}$	P	NP	?	?	[70], Subsection 2.2.1, Subsection 2.2.4
optimal $MCA_{t,v}$	P	NP	?	?	Subsection 2.2.2, Subsection 2.2.4
optimal BS_d	P	NP-complete	NP-hard	NP-hard	[30], Subsection 2.2.4
optimal $VCA_{\bar{t},2}$	P	NP-complete	NP-hard	NP-hard	[30], Subsections 2.2.3, Subsection 2.2.4
optimal $VCA_{\bar{t},v}$	P	NP	?	?	Subsection 2.2.3, Subsection 2.2.4
optimal $VCA_{\bar{t}}$	P	NP-hard	NP-hard	NP-hard	[30], Subsection 2.2.4
optimal CA(G)	P	NP-complete	NP-hard	NP-hard	[30], [177], Subsection 2.2.4
CAFE	NP-complete	NP-complete	NP-hard	NP-hard	[60], [119], Subsection 2.2.5

Table 2.1: Overview of complexity results regarding main **classes** of covering arrays. Notice that the results for $CA(G)$ are obtained only for $v = 2$, while for CAFEs the NP-completeness results apply when $v \geq 5$. Results for optimal $CA_{t,v}$ are for $t > 2$ or $v > 2$ and for optimal $VCA_{\bar{t},v}$ for $v > 2$.

2.3 Review and Extension of Problems and Algorithms for Covering Arrays via Set Covers

In this section we first review how the problem of generating CAs can be interpreted as a set cover problem, a connection that has been previously explained in an extensive way in [144]. Large parts of [144] are part of the authors Master thesis and can thus not be considered a contribution of this dissertation. However, the changes that the manuscript [144] has undergone until it reached its finally published form, especially the added content, fall within the PhD-studies of the author and can thus be considered as part of this dissertation. We therefore content ourselves with repeating the key idea of encoding the optimal CA generation problem as a minimal set cover problem , guided by means of an example, as it is also relevant for Section 3.2 and was also used in [142]. The interested reader is kindly referred to [144] or [181] for more details. Thereafter we present those parts of [144] that were later added and that are relevant for this thesis. In particular, we briefly review the notion of *density* as introduced in [71], which is the basis for the notion of *weighted density*. Further, the greedy heuristic algorithm for CA generation that results from the connection to minimal set covers is compared against a

CA generation algorithm that uses an exact method, in particular SAT solving. Finally, the greedy heuristics are also used to re-compute some optimal VCAs.

2.3.1 Encoding Covering Array Problems as Set Cover Problems

The notion of t-way interactions provides the means to formulate problems related to CAs as set cover problems. The latter being very well studied problems in computer science, see e.g. [182].

Definition 2.3.1 *A* set cover *(SC) of a finite set U is a set $\mathcal{S}$ of non-empty subsets of U whose union is U. In this context, U is called the* universe *and the elements of $\mathcal{S}$ the* blocks*.*

A typical optimization problem for set covers is the *minimal set cover problem*. That is, for given $(U, \mathcal{S})$, to find a subset $\mathcal{C} \subseteq \mathcal{S}$ of minimal cardinality, such that $\bigcup \mathcal{C} = U$. We call $(U, \mathcal{S})$ also an *SC instance*.

The problem of constructing a CA for given strength t, number of columns k and alphabet size v is interpreted as an SC instance $(U, \mathcal{S})$, by defining the universe U as the set of all t-way interactions. The blocks in $\mathcal{S}$ correspond to the rows that can appear in a CA, where a block is defined as the set of t-way interactions covered by the corresponding row. We call such a set cover instance *corresponding* to the CA instance given t, k and v. To make this connection more explicit, consider the following Example:

Example 2.3.2 *Assume we want to construct a* $\mathsf{CA}(N; 2, 4, 2)$ *with minimal N. We translate this problem into a minimal set cover problem. Each 2-way interaction needs to be covered, thus*

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

Each vector of $\{0,1\}^4$ which can appear as a row in a $\mathsf{CA}(N; 2, 4, 2)$ *is identified with the set of 2-way interactions it covers, e.g. the row $(0,0,1,0)$ is mapped to the block $\{(0,0,-,-),(0,-,1,-),(0,-,-,0),(-,0,1,-),(-,0,-,0),(-,-,1,0)\}$. Thus we get the SC instance $(U, \mathcal{S})$ corresponding to the CA instance with parameters $t = 2$ and $k = 4$, where*

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

$$\{(0,1,-,-),(0,-,1,-),(0,-,-,0),(-,1,1,-),(-,1,-,0),(-,-,1,0)\},$$
$$\{(0,1,-,-),(0,-,1,-),(0,-,-,0),(-,1,1,-),(-,1,-,0),(-,-,1,1)\},$$
$$\{(1,0,-,-),(1,-,0,-),(1,-,-,0),(-,0,0,-),(-,0,-,0),(-,-,0,0)\},$$
$$\{(1,0,-,-),(1,-,0,-),(1,-,-,1),(-,0,0,-),(-,0,-,1),(-,-,0,1)\},$$
$$\{(1,0,-,-),(1,-,1,-),(1,-,-,0),(-,0,1,-),(-,0,-,0),(-,-,1,0)\},$$
$$\{(1,0,-,-),(1,-,1,-),(1,-,-,1),(-,0,1,-),(-,0,-,1),(-,-,1,1)\},$$
$$\{(1,1,-,-),(1,-,0,-),(1,-,-,0),(-,1,0,-),(-,1,-,0),(-,-,0,0)\},$$
$$\{(1,1,-,-),(1,-,0,-),(1,-,-,1),(-,1,0,-),(-,1,-,1),(-,-,0,1)\},$$
$$\{(1,1,-,-),(1,-,1,-),(1,-,-,0),(-,1,1,-),(-,1,-,0),(-,-,1,0)\},$$
$$\{(1,1,-,-),(1,-,1,-),(1,-,-,1),(-,1,1,-),(-,1,-,1),(-,-,1,1)\}\}.$$

This set cover instance $(U,\mathcal{S})$ *contains the minimal set cover*

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

Provided the correspondence between rows and blocks, it is possible to map the minimal set cover $\mathcal{C}$ *to the optimal* $\mathsf{CA}(5;2,4,2)$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0\\ 0 & 1 & 1 & 0\\ 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 1\\ 0 & 0 & 1 & 1 \end{pmatrix}. \tag{2.6}$$

2.3.2 A Weighted Density Notion for Covering Arrays

The greedy heuristic algorithms for CA generation resulting from the connection to minimal set covers that have been formulated in [144] are compared against more sophisticated heuristic algorithms, algorithmic variants of the *deterministic density algorithm* (DDA) which was developed dedicated for CA generation, see [71]. This algorithm is based on the notion of density, also introduced in [71], which we review briefly next for the sake of completeness.

Using the same notation as in [71]: For a given s-way interaction S and a set W with $\phi(S) \subseteq W \subseteq \{p_1,\ldots,p_k\}$ and $|W| = t$, let $E(S,W) := \{S \cup \{(w,v_w)\} | w \in W \setminus \phi(S), v_w \in V_w\}$ denote the set of t-way interactions, that extend the s-way interaction S to the t positions defined by W. The cardinality of this set is denoted by $\pi(S,W) = \prod_{f\in W\setminus\phi(S)} |V_f|$. Further, for a given array A let γ^A be the function that

maps a t-way interaction T to $\gamma^A(T) = 0$, if T is covered by some row of A and to $\gamma^A(T) = 1$ otherwise. Then, the *density of S with respect to W (and A)* is defined as $\delta^A(S,W) := \pi(S,W)^{-1} \sum_{T \in E(S,W)} \gamma^A(T)$; the *density of S* is defined as $\delta^A(S) := \sum_{\substack{\phi(S) \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ |W|=t}} \delta^A(S,W)$, and finally the *factor density* $\delta_f^A(S)$ for a factor (or column) f and the s-tuple S, as $\delta_f^A(S) := \sum_{\substack{\phi(S) \cup \{f\} \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ |W|=t}} = 1/|V_f| \cdot \sum_{v \in V_f} \delta(S \cup \{(f,v)\})$.

Based on the above reviewed notion, we introduce the following version of a *weighted density*, used in the wbDDAAlgorithm, a weighted and budgeted variant of DDA, please see Algorithm 4 in [144] for more details. For given weights t_{i,u_j} reflecting the importance of the assignment of value u_j to the i-th factor, let $\omega(V_i) := \sum_{u \in V_i} \omega(u)$ denote the sum of the weights of the values in V_i for the i-th factor. The weight of a t-way interaction T is defined as $\omega(T) := \prod_{(f,v_f) \in T} t_{f,v_f}$, and for a given s-way interaction S and a t-way interaction T we define $\omega(S,T) := \prod_{(f,v_f) \in T \setminus S} t_{f,v_f}$ for $S \subsetneq T$ and $\omega(S,T) := 0$ otherwise, i.e. $\omega(S,T)$ is the weight of the $(t-s)$-way interaction that extends S to T. The notations $E(S,W)$ and γ^A, for a given array A, are as in the unweighted case discussed above. Instead of the number of elements as in $\pi(S,W)$, we consider the sum of $\omega(S,T)$ for each tuple T in $E(S,W)$, to define: ${}^{\omega}\pi(S,W) := \sum_{T \in E(S,T)} \omega(S,T) = \prod_{f \in W \setminus \phi(S)} \omega(V_f)$. The *weighted density of S with respect to W (and A)* is defined as ${}^{\omega}\delta^A(S,W) := {}^{\omega}\pi(S,W)^{-1} \sum_{T \in E(S,W)} \omega(S,T)\gamma^A(T)$. Then the *weighted density of S* is defined as ${}^{\omega}\delta^A(S) := \sum_{\substack{\phi(S) \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ |W|=t}} {}^{\omega}\delta^A(S,W)$, i.e. the relation of the sum of weights of tuples extending S to *uncovered* t-way interactions in the columns specified by W to the sum of all such t-way interactions. For the derivation of the *weighted factor density* ${}^{\omega}\delta_f^A(S)$ for factor (or column) f and the s-tuple S, we proceed analogue to [169, 71], splitting the sum of ${}^{\omega}\delta^A(S)$ in two parts:

$$ {}^{\omega}\delta^A(S) = \sum_{\substack{\phi(S) \cup f \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ |W|=t}} {}^{\omega}\delta^A(S,W) + \sum_{\substack{\phi(S) \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ f \notin W, |W|=t}} {}^{\omega}\delta^A(S,W), $$

and define ${}^{\omega}\delta_f^A(S)$ as the first of these two sums. With the introduced notions this yields:

$$ \begin{aligned} {}^{\omega}\delta_f^A(S) &= \sum_{\substack{\phi(S) \cup f \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ |W|=t}} {}^{\omega}\delta^A(S,W) \\ &= \sum_{\substack{\phi(S) \cup f \subseteq W \\ |W|=t}} \sum_{v \in V_f} \frac{1}{\omega(V_f)} t_{f,v} \sum_{T \in E(S \cup \{(f,v)\},W)} {}^{\omega}\pi(S \cup \{(f,v)\},W)^{-1} \omega(S \cup \{(f,v)\},T)\gamma^A(T) \\ &= \frac{1}{\omega(V_f)} \sum_{v \in V_f} t_{f,v} \sum_{\substack{\phi(S) \cup f \subseteq W \subseteq \{p_1,\ldots,p_k\} \\ |W|=t}} {}^{\omega}\delta^A(S \cup \{(f,v)\},W) \\ &= \frac{1}{\omega(V_f)} \sum_{v \in V_f} t_{f,v} \cdot {}^{\omega}\delta^A(S \cup \{(f,v)\}). \end{aligned} $$

Hence, in step 7 of Algorithm 3 in [144], a value u with ${}^{\omega}\delta^{A}(R\cup\{(i,u)\}) \geq {}^{\omega}\delta_i^{A}(R)$, can be found by choosing u such that $t_{i,u}\cdot{}^{\omega}\delta^{A}(R\cup\{(i,u)\}) \geq 1/|V_i|\sum_{v\in V_i} t_{i,v}\cdot{}^{\omega}\delta^{A}(R\cup\{(i,v)\})$.

For the experimental evaluation, where the greedy heuristic algorithm for weighted, budgeted CA generation is compared against WBDDA, using the weighted density notion above, the interested reader is referred to [144]. Here, we show a comparison of the greedy heuristics discussed in Section 4 of [144] to the exact approach using SAT solving and backtracking, presented in [69].

Table 2.2 shows the results of our experiments, where we compared against the documented instances for the exact algorithms in [69], where it is important to note that experiments in [69] were conducted on a Pentium M 1.7GHz, compared to our Intel Core i7-4770 3.40GHz and that we compared against the fastest version documented in [69]. As expected the exact algorithms produce smaller arrays, in particular they achieve CAN in cases where they terminated, consuming much more time than the heuristic algorithms, which produce larger arrays in many cases but are capable of returning output within fractions of a second, when the exact algorithms fail to do so. Further consider that another advantage of the greedy heuristics is that they do not depend on input of the number of rows N, whereas exact methods often rely on this input. The input of N can be circumvent by adding an additional loop around the backtrack algorithm, increasing or decreasing N, but this is even more time consuming.

(t, k, v)	Hnich et al.		GAETG		DDA	
	N	*time (s)*	*N*	*time (s)*	*N*	*time (s)*
(3, 4, 2)	**8**	0.01	**8**	0.000093769	**8**	0.000081963
(3, 5, 2)	**10**	0.01	12	0.000162323	12	0.00014341
(3, 6, 2)	**12**	0.02	**12**	0.000451739	14	0.000233559
(3, 7, 2)	**12**	0.05	14	0.001420999	15	0.000358077
(3, 8, 2)	**12**	0.06	15	0.004380311	16	0.000528182
(3, 9, 2)	**12**	0.25	16	0.011517703	16	0.000766397
(3, 10, 2)	**12**	0.35	16	0.030609118	16	0.001059486
(3, 11, 2)	**12**	0.71	18	0.098491521	20	0.001716452
(3, 12, 2)	-	269.58	**19**	0.266934654	21	0.00239722
(4, 5, 2)	**16**	0.01	**16**	0.0001733	**16**	0.000148796
(4, 6, 2)	**21**	0.11	26	0.000759202	31	0.000415568
(4, 7, 2)	-	>300	**24**	0.003204877	34	0.000833821

Table 2.2: Comparison of exact algorithms to heuristic algorithms. The values in column "Hnich et al." correspond to the fastest algorithm as documented in [69], Tables 1 and 2. Bold entries indicate the smallest number of rows in this comparison, when no CA was found we denote the size as "-".

Finally, to evaluate our implementations of greedy algorithms in the realm of VCAs, we compare by means of *(cyclic) consecutive CAs*, denoted as $\mathsf{cCCA}(N;t,k,v)$, for which several values of VCAN are known; see [59] for details. For the tested binary cases $cCCA(N;2,k,2), k \in \{2,\ldots,22\}$, $cCCA(N;3,k,2), k \in \{6,\ldots,22\}$ and $cCCA(N;4,k,2), k \in \{12,\ldots,22\}$, as well as for the ternary cases $cCCA(N;2,k,3), k \in \{2,\ldots,12\}$ and $cCCA(N;3,k,3), k \in \{6,\ldots,12\}$ GAETG (see also Algorithm 9 and

Method \ Param.	(2,7,3)	(2,12,3)	(3,6,3)	(3,9,3)	(3,12,3)	(3,5,3)	(4,6,3)	(4,7,3)	(5,7,3)	(5,8,3)
VCAN	**9**	**9**	**27**	**27**	**27**	**27**	**81**	**81**	**243**	**243**
GAETG	**9**	**9**	**27**	**27**	**27**	**27**	88	86	283	273
DDA	12	19	35	38	38	34	110	117	352	367

Table 2.3: Results for non-binary VCAs. The values for VCAN are taken from [59].

Section 3.3 for more details) and DDA achieved VCAN. For other known values of VCAN, as given in [59], Section 3.2, GAETG achieved VCAN in some cases, whereas DDA stays above this bound for all cases. See Table 2.3 for the results.

In this section we briefly reviewed how the optimal CA generation problem can be mapped to a minimal set cover problem, and have thus presented a way how the CA generation problem can be translated to a well known problem in theoretical computer science. In the next section we will see how the optimal CA generation problem can be mapped to a multi-variate polynomial equation system, and thus a way how the CA generation problem can be translated to a problem of discrete mathematics.

2.4 Modeling Optimal Covering Array Generation via Computational Commutative Algebra

The previous Section 2.2 pointed already out that, aside from the problem of determining CANs and generating optimal CAs, a variety of different problems arise with the notion of covering arrays. An extensive list of such problems can be found in [154]. In this section we consider the following two problems, which correspond to Problems 2 and 9 of [154], and propose algorithms algebraic approach of modeling binary CAs of strength two.

Problem 2.4.1 (Computational Existence) *For given CA parameters and given $N \in \mathbb{N}$, construct one/all covering array(s) with exactly N rows, or terminate indicating there exists no such CA.*

Problem 2.4.2 (Computational Factor Extension) *Given a covering array and an alphabet size v, construct one/all new additional column(s) such that the extended matrix constitutes a covering array with the additional column (not adding any additional rows), or terminate indicating there exists no such a column.*

The work presented in this section builds upon the work presented in [67] which is an generalization and extension of [154]. In particular, in Subsection 2.4.1 the algebraic approach of modeling binary CAs of strength two in [154] is extended for CAs over arbitrary alphabets and for higher strengths. In Subsection 2.4.1, based on our algebraic model, we devise an algorithm for the construction of such CAs. Further, in Subsection 2.4.2 we discuss how the proposed algebraic modeling of CAs can be used when trying to extend a given CA with an additional column and show how the degrees of the appearing polynomials can be reduced. In Subsection 2.4.3 we list some experiments where we compute CAs arising as solutions of equation systems using different solving techniques.

Notations. In addition to the notations introduced in Section 1.3, in this section we make use of the following notations. By $\mathbb{Z}_v := \mathbb{Z}/(v\mathbb{Z})$ we denote the ring of integers modulo v, where we may identify the elements of $\mathbb{Z}_v$ with the elements of $[v] = \{0, 1, \ldots, v-1\}$. Further, with $\mathbf{1}$ we denote the vector having all entries equal to 1, where the length of the vector is clear from the context. Additionally, for vectors $\mathbf{h} = (h_1, \ldots, h_N)^T$ we use the notation $\mathbf{prod}(\mathbf{h}) := \prod_{i=1}^{N} h_i$.

2.4.1 Algebraic Models for CAs of Arbitrary Strength

Definition 2.4.3 (v-ary t-way interaction distinguish property) *Let $(R, +, \cdot, 0, 1)$ be an integral domain with 1, and $a_1, \ldots, a_t \in R$. We say that $(R, a_1, \ldots, a_t)$ has the* v-ary t-way interaction distinguish property*, if and only if $\forall u_i \in [v],\ \forall i = 1, \ldots, t$ the elements $u_1 a_1 + \ldots + u_t a_t \in R$, are pairwise different, where we interpret the natural numbers $u \in [v]$ embedded in R as $\underbrace{(1 + \ldots + 1)}_{u}$.*

Remark. *Notice that from the v-ary t-way interaction distinguish property, for $t \geq 1$, it immediately follows that $char(R) \geq v$ or $char(R) = 0$. Hence the set $[v] = \{0, 1, \ldots, v-1\}$ of natural numbers is mapped injectively into R.*

When the underlying alphabet $[v]$ is clear from the context, we also speak of the *t-way interaction distinguish property* for short. We can interpret the notion of the t-way interaction distinguish property as a special kind of linear independence, considering the appropriate algebraic structures. For that purpose, using the same notations as in Definition 2.4.3, we regard the integral domain R as a unitary $\mathbb{Z}_v$-module. Then $(R, a_1, \ldots, a_t)$ having the v-ary t-way interaction distinguish property is equivalent to the linear independence of $a_1, \ldots, a_t$. We give some examples of rings R and elements $a_1, \ldots, a_t$, such that $(R, a_1, \ldots, a_t)$ has the v-ary t-way interaction distinguish property:

1. For a ring S of characteristic $char(S) \geq v$ or $char(S) = 0$, let $R = S[x_1, \ldots, x_n]$ be the ring of all polynomials in the indeterminates x_i, and $a_i = x_i\ \forall i = 1, \ldots, n$.
2. Let $K \geq L$ be fields of characteristic $char(L) \geq v$ or $char(L) = 0$, $R = L$ and let $a_1, \ldots, a_t \in K$ be algebraically independent over L.

Lemma 2.4.4 *Let $(R, a_1, \ldots, a_t)$ have the v-ary t-way interaction distinguish property, then for any $(u_1, \ldots, u_t) \in [v]^t$ and $(x_1, \ldots, x_t) \in [v]^t$ we have:*

$$(x_1, \ldots, x_t) = (u_1, \ldots, u_t) \Leftrightarrow (x_1, \ldots, x_t) \cdot (a_1, \ldots, a_t)^T - \sum_{i=1}^{t} u_i a_i = 0.$$

Proof: The claim follows directly from the definition of the v-ary t-way interaction distinguish property. □

In the following we will use elements $a_1, \ldots, a_t$ to select t different columns of a given matrix. For that purpose we need the following additional definition.

Definition 2.4.5 *Let $t \leq k \in \mathbb{N}$ and $C = \{c_1, \ldots, c_t\} \subseteq [k]$ with $|C| = t$. Further let $(R, +, \cdot, 0)$ be a ring and $a_1, \ldots, a_t \in R \setminus \{0\}$, then we define $\iota^C_{t,k}(a_1, \ldots, a_t)$ as the (column) vector of length k having entry a_i in position c_i, $\forall i = 1, \ldots, t$ and the entry 0 in all other positions.*

For example, for $C = \{2, 3, 6\}$ then $\iota^C_{3,6}(a_1, a_2, a_3) = (0, a_1, a_2, 0, 0, a_3)^T$.

Lemma 2.4.6 *Let R be an integral domain (with 1) and $(R, a_1, \ldots, a_t)$ have the v-ary t-way interaction distinguish property, M be a given $N \times k$ matrix $M = (\mathbf{m}_1, \ldots, \mathbf{m}_k)$ defined over R, $C = \{c_1, \ldots, c_t\}$ with $1 \leq c_1 < \ldots < c_t \leq k$ and $M_C = (\mathbf{m}_{c_1}, \ldots, \mathbf{m}_{c_t})$ be the matrix comprised of the t columns defined by C. Further consider a t-tuple $(u_1, \ldots, u_t) \in [v]^t$, then the following statements are equivalent:*

1. *The tuple $(u_1, \ldots, u_t)$ appears at least once as a row in the matrix M_C.*
2. *The vector $\mathbf{h} := (h_1, \ldots, h_N)^T := M \cdot \iota^C_{t,k}(a_1, \ldots, a_t) - \mathbf{1}(\sum_{i=1}^t u_i a_i)$ contains at least one component equal to zero.*
3. ***prod***$(\mathbf{h}) = \prod_{i=1}^N h_i = 0$.

Proof: The equivalence of 1 and 2 follows from Lemma 2.4.4. The equivalence of 2 and 3 holds since R is an integral domain. □

Equations as in item 3 of Lemma 2.4.6 are formulated in such a way that they are semantically equivalent to the appearance of a t-way interaction in an array. Considering these equations for all t-way interactions and all selections of t columns, Lemma 2.4.6 leads to the main result of this section.

Theorem 2.4.7 *Let R be a ring and $(R, a_1, \ldots, a_t)$ have the v-ary t-way interaction distinguish property. Then for a matrix $M \in \mathbb{Z}_v^{N \times k}$ the following statements are equivalent:*

1. *M is a $\mathsf{CA}(N; t, k, v)$*
2. *$\forall C \in \binom{\{k\}}{t}$, $\forall (u_1, \ldots, u_t) \in [v]^t$:*

$$\textbf{\textit{prod}}(M \cdot \iota^C_{t,k}(a_1, \ldots, a_t) - \mathbf{1} \cdot (u_1, \ldots, u_t) \cdot (a_1, \ldots, a_t)^T) = 0. \tag{2.7}$$

Proof: The assertion of the theorem follows immediately considering that the equivalence of 1 and 3 of Lemma 2.4.6 holds for all $C \subseteq \{1, \ldots, k\}$ with $|C| = t$ and all $(u_1, \ldots, u_t) \in [v]^t$. □

Based on this algebraic characterization of CAs, we can now describe the previously mentioned computational or decisional problems for CAs as related problems found in multivariate polynomial algebra. In particular, through our algebraic modeling the problem(s) of constructing and computing covering arrays can be formulated as instances of algebraic equation systems, where each solution of the system, provided existence, corresponds to a covering array.

Corollary 2.4.8 *Let R be a ring and $(R, a_1, \ldots, a_t)$ have the v-ary t-way interaction distinguish property, and $X := (x_{i,j})$ be an $N \times k$ array of variables. Then any solution to the following system of equations in the unknowns $x_{i,j}$ yields a $\mathsf{CA}(N; t, k, v)$:*

1. $\forall i \in \{1, \ldots, N\}, \forall j \in \{1, \ldots, k\}$

$$\prod_{r=0}^{v-1} (x_{i,j} - r) = 0. \tag{2.8}$$

2. $\forall C \in \binom{\{k\}}{t}, \ \forall (u_1, \ldots, u_t) \in [v]^t$:

$$\boldsymbol{prod}(X \cdot \iota_{t,k}^{C}(a_1, \ldots, a_t) - \mathbf{1} \cdot (u_1, \ldots, u_t) \cdot (a_1, \ldots, a_t)^T) = 0. \tag{2.9}$$

Following the terminology of [154], we call the equations (2.8) of Corollary 2.4.8 the *domain equations* and the equations (2.9) the *coverage equations*.

An Algebraic Algorithm for Searching CAs

Provided the derived algebraic characterization (Theorem 2.4.7, Corollary 2.4.8) of CAs, it is possible to interpret these combinatorial structures as elements in varieties corresponding to ideals in polynomial rings over fields. In [154] an algorithm was presented that addresses Problem 2.4.1 in the binary case for strength $t = 2$, interpreting the appearing polynomials as elements of $R = \mathbb{Q}[x_1, \ldots, x_\gamma, a_1, a_2]$, depending on the binary 2-way interaction distinguish property of (R, a_1, a_2).

With the results presented in this paper, a natural way to generalize this algorithm to the case of $\mathsf{CA}(N; t, k, v)$ for arbitrary t and v is possible by interpreting appearing polynomials as elements of $R = \mathbb{Q}[x_1, \ldots, x_\gamma, a_1, \ldots, a_t]$ and relying on the t-way interaction distinguish property of $(R, a_1, \ldots, a_t)$. Before we describe such an algorithm, we address how the replacement of the indeterminates a_i can reduce the number of symbolic variables. For example, in Algorithm 3 of [154] the indeterminates a_i were replaced by random elements of $\mathbb{Q}$. In the following we show how we can choose values for the a_i while still ensuring the t-way interaction distinguish property.

Lemma 2.4.9 *Let $R = \mathbb{Q}[x_1, \ldots, x_\gamma]$ and let $v \in \mathbb{N}$ with $v \geq 2$, then $(R, 1, v^1, \ldots, v^{t-1})$ has the v-ary t-way interaction distinguish property.*

Proof: We have to show that all elements $u_t v^{t-1} + \ldots u_2 v^1 + u_1 v^0$, for $u_i \in [v]$ $\forall i = 1, \ldots, t$ are pairwise different. Certainly this holds, as the elements of this set are exactly the natural numbers in $\{0, \ldots, v^t - 1\}$ and $(u_t, \ldots, u_1)$ corresponds to their base v representation. □

Summarizing briefly, Algorithm 2 initializes an $N \times k$ array of symbolic variables, and generates all *coverage equations* (lines 4-9) and *domain equations* (lines 10-13) for this

matrix according to equations (2.8) and (2.9). Provided the previous Lemma 2.4.9 we can interpret the appearing polynomials as elements of $\mathbb{Q}[x_1, \ldots, x_{Nk}]$ using the elements $a_i = v^{i-1}$ for $i = 1, \ldots, t$ providing the t-way interaction distinguish property. This system of multivariate polynomial equations then is fed to an external Solve procedure. In Subsection 2.4.3 we will describe how such a procedure can be instantiated. Depending on the instantiation of this Solve procedure, the respective version of Problem 2.4.1 for searching *all* or *one* CA(s) is targeted.

Algorithm 2 AlgebraicSearchCAs

1: INPUT: N, t, k, v
Require: $t \leq k$
2: Create a symbolic $N \times k$ array X containing variables $x_1, \ldots, x_{Nk}$
3: $EQall := \emptyset$
4: **for** $C \in \binom{\{k\}}{t}$ **do** ▷ Add coverage equations
5: **for** $\mathbf{u} \in [v]^t$ **do**
6: $EQ := \mathbf{prod}(X \cdot \iota^C_{t,k}(v^0, \ldots, v^{t-1}) - \mathbf{1} \cdot (u_1, \ldots, u_t) \cdot (v^0, \ldots, v^{t-1})^T) = 0$
7: add EQ to $EQall$
8: **end for**
9: **end for**
10: **for** $i = 1, \ldots, Nk$ **do** ▷ Add domain equations
11: $EQ := \prod_{j=0}^{v-1}(x_i - j) = 0$
12: add EQ to $EQall$
13: **end for**
14: Interpret $EQall$ as subset of $\mathbb{Q}[x_1, ..., x_{Nk}]$
15: $V = \text{Solve}(EQall)$ ▷ Call external solver
16: **if** $V \neq \emptyset$ **then**
17: **return** V;
18: **else** print "No CA exists";
19: **end if**

2.4.2 An Algebraic Model for Column Extensions of CAs

Similar to Subsection 2.4.1, in this subsection we devise a model such that Problem 2.4.2 can be treated as a problem of computational algebra. When extending a CA, with one column it is sufficient to ensure that in all subarrays comprised by t columns, involving the newly added column, all t-way interactions are covered, to guarantee that the defining properties of Definition 1.3.2 hold. Note that this technique of iteratively extending an existing CA with a column, followed by possible row extensions, is applied in the widely used IPO strategy for CA construction (see [174]). We illustrate this by the following example.

Example 2.4.10 *Consider the following* $\mathsf{CA}(9; 2, 3, 3)$*:*

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

Next, we will interpret the problem of extending M by one column as a problem of finding solutions to the unknowns $x_1, \ldots, x_9$, such that

$$\overline{M} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{pmatrix}^T \tag{2.10}$$

is again a CA. One such solution is e.g. $(x_1, \ldots, x_9) = (0, 1, 2, 2, 0, 1, 1, 2, 0)$.

Theorem 2.4.11 *Consider the ring $R = \mathbb{Q}[x_1, \ldots, x_N]$ and elements $a_1, \ldots, a_t$ such that $(R, a_1, \ldots, a_t)$ has the v-ary t-way interaction distinguish property. Given an $N \times k$ matrix M that is a $\mathsf{CA}(N; t, k, v)$, the following assertions are equivalent:*

1. *There exists a vector $\mathbf{c} \in [v]^{N\times 1}$ such that the horizontal extension $(M|\mathbf{c})$ of M by the column $\mathbf{c}$ is a $\mathsf{CA}(N; t, k+1, v)$.*
2. *The system of equations in the unknowns $x_1, \ldots, x_N$ consisting of the equations in (2.11) and (2.12) has a non trivial solution.*

$$\forall i \in \{1, \ldots, N\} : \prod_{j=0}^{v-1} (x_i - j) = 0. \tag{2.11}$$

$$\forall C \in \binom{\{k\}}{t-1} \; \forall (u_1, \ldots, u_t) \in \prod_{i\in C} [v_i] \times [v_{k+1}] :$$

$$\boldsymbol{prod}((M|\mathbf{x}) \cdot \iota_{t,k+1}^{C\cup\{k+1\}}(a_1, \ldots, a_t) - \mathbf{1} \sum_{i=1}^{t} u_i a_i) = 0. \tag{2.12}$$

Note that in (2.12) we only consider those subsets of $\{1, \ldots, k, k+1\}$ having cardinality t that contain the element $k+1$.

Proof: If there exists a vector $\mathbf{c} = (c_1, \ldots, c_N) \in [v]^{N\times 1}$ such that $(M|\mathbf{c})$ is a $\mathsf{CA}(N; t, k+1, v)$, then $\mathbf{x} := \mathbf{c}$ obviously satisfies the equations in (2.11). From Theorem 2.4.7 we also get that x satisfies all equations in (2.12) when substituting k with $k+1$ in Theorem 2.4.7. Conversely assume $\mathbf{x} = (x_1, \ldots, x_N) \in [v]^{N\times 1}$ is a solution to the system of equations given by (2.11) and (2.12). Since M is an $\mathsf{CA}(N; t, k, v)$, from Theorem 2.4.7 we get that $\mathbf{prod}(M \cdot \iota_{t,k}^{C}(a_1, \ldots, a_t) - \mathbf{1} \sum_{i=1}^{t} u_i a_i) = 0$ holds for all $C \in \binom{\{k\}}{t}$ and $(u_1, \ldots, u_t) \in [v]^t$. Toghether with (2.12) we have that $\mathbf{prod}((M|\mathbf{x}) \cdot \iota_{t,k+1}^{C}(a_1, \ldots, a_t) - \mathbf{1} \sum_{i=1}^{t} u_i a_i) = 0$ holds for the remaining $C \in \binom{\{k+1\}}{t}$ and $(u_1, \ldots, u_t) \in [v]^t$. Corollary 2.4.8 then ensures that $(M|\mathbf{x})$ is a CA. □

Similar to Algorithm 2, based on Theorem 2.4.11 one can formulate an algorithm that treats Problem 2.4.2, i.e. an algorithm that finds all possible column extensions to a given CA when they exist, see the pseudo-code in Algorithm 3.

Algorithm 3 AllColumnExtensions

INPUT: M, N, t, k, v
Require: M is a $\mathsf{CA}(N; t, k, v)$
Create a symbolic $N \times 1$ vector $\mathbf{x} = (x_1, \dots x_N)$
$EQall := \emptyset$
for $C \in \binom{\{k\}}{t-1}$ **do** ▷ Add coverage equations
  for $\mathbf{u} \in [v]^t$ **do**
    $EQ := \mathbf{prod}((M|\mathbf{x}) \cdot \iota_{t,k}^{C \cup \{k+1\}}(v^0, \dots, v^{t-1})^T - (u_1, \dots, u_t) \cdot (v^0, \dots, v^{t-1})^T) = 0$
    add EQ to $EQall$
  end for
end for
for $i = 1, \dots, N$ **do**
  $EQ := \prod_{j=0}^{v-1}(x_i - j) = 0$ ▷ Add domain equations
  add EQ to $EQall$
end for
Interpret $EQall$ as subset of $\mathbb{Q}[x_1, ..., x_N]$
$V = \text{Solve}(EQall)$ ▷ Call external solver
if $V \neq \emptyset$ **then**
  return V;
else print "No column extension exists";
end if

Remark 2.4.12 (Reduction of appearing Degree) *Taking a closer look at the linear factors, e.g. the r-th factor* $(m_{r,1}, \dots, m_{r,k}, x_r) \cdot \iota_{t,k+1}^{C \cup \{k+1\}}(a_1, \dots, a_t) - \sum_{i=1}^{t} u_i a_i = \sum_{i=1}^{t-1}(m_{r,c_i} - u_i)a_i + (x_r - u_t)a_t$*, of the polynomials appearing in* (2.12)*, we can see that some of them can never evaluate to zero, independent of the choice of* x_r*. This is due to the t-way interaction distinguish property, which ensures that* $\sum_{i=1}^{t-1}(m_{r,c_i} - u_i)a_i + (x_r - u_t)a_t = 0$ *if and only if* $(m_{r,c_1}, \dots, m_{r,c_{t-1}}, x_r) = (u_1, \dots, u_t)$*. Thus,* $(m_{r,c_1}, \dots, m_{r,c_{t-1}}) = (u_1, \dots, u_{t-1})$ *is a necessary condition so that there exists a value for* x_r*, such that the r-th factor evaluates to zero. Therefore we can significantly reduce the degrees of the polynomials appearing in the coverage equations, as in* (2.12)*, when using the result of Theorem 2.4.11 for the computation of CAs. We make this explicit by providing a small example.*

Example 2.4.13 *Continuing Example 2.4.10, we consider the matrix* $\overline{M}$*, as given in* (2.10)*, as a matrix over* $GF(3)[a_1, a_2]$*, the ring of polynomials in the indeterminates* a_1, a_2 *over the finite field with three elements, and consider the coverage equation for* $C = \{1, 4\}$ *and* $(u_1, u_2) = (1, 0)$*.*

$$\begin{aligned}
&\mathbf{prod}((M|\mathbf{x}) \cdot (a_1, 0, 0, a_2)^T - (a_1, a_1)) = \\
&(0 \cdot a_1 + x_1 \cdot a_2 - a_1)(0 \cdot a_1 + x_2 \cdot a_2 - a_1)(0 \cdot a_1 + x_3 \cdot a_2 - a_1) \cdot \\
&(1 \cdot a_1 + x_4 \cdot a_2 - a_1)(1 \cdot a_1 + x_5 \cdot a_2 - a_1)(1 \cdot a_1 + x_6 \cdot a_2 - a_1) \cdot \\
&(2 \cdot a_1 + x_7 \cdot a_2 - a_1)(2 \cdot a_1 + x_8 \cdot a_2 - a_1)(2 \cdot a_1 + x_9 \cdot a_2 - a_1) = 0. \qquad (2.13)
\end{aligned}$$

Due to the 3-ary 2-way interaction distinguish property of $GF(3)[a_1, a_2]$ *we have e.g.* $(0a_1 + x_1a_2 - a_1) \neq 0$ *for any value of* $x_1 \in GF(3)$ *or* $(2a_1 + x_9a_2 - a_1) \neq 0$ *for any value*

of $x_9 \in GF(3)$. *Hence equation* (2.13) *is equivalent to*

$$(a_1 + x_4a_2 - a_1)(a_1 + x_5a_2 - a_1)(a_1 + x_6a_2 - a_1) = 0, \tag{2.14}$$

reducing the degree of the polynomial in this coverage equation from 9 *to* 3. *A similar reduction of the degrees can be done for the other coverage equations. The combinatorial interpretation, or reason, for this reduction of the degrees, is that, again considering the above example, the* 2*-way interaction* $\{(1,1),(4,0)\}$ *can only be covered by the* 4*-th,* 5*-th or* 6*-th row when extending* M *with one column.*

2.4.3 Experiments Using Gröbner Bases and Supercomputing

We employed two methodologies to solve the systems of polynomial equations arising from our algebraic modeling for CAs. Firstly, Gröbner bases (GB) computations in Maple and Magma, and secondly, exhaustive search using C and parallel programming using C/MPI (Message Passing Interface). Each of these different solving implementations has been used as a means to instantiate the SOLVE procedure in Algorithm 2.

Solving Using Gröbner Bases

For the Gröbner bases computations, the equation system can be encoded in Maple and Magma format and lexicographical and total degree Gröbner bases can be computed. If the result of the (reduced) Gröbner bases computation is equal to $\{1\}$, then we know that the system does not have any solutions [183]. If the result of the (reduced) Gröbner bases computation is not equal to $\{1\}$, then we use the actual basis to recover some solution of the system. We observed that in general, Maple and Magma are able to successfully compute Gröbner bases for systems of polynomial equations arising from CA constructions, for up to 20 binary and 10 ternary variables. We give a related example below:

Example 2.4.14 (Column extension of a covering array) *Continuing Example 2.4.10, recall that*

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \end{pmatrix}^T \quad \textit{and} \quad \overline{M} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{pmatrix}^T.$$

We generate the system of domain and coverage equations, where the 9 *polynomials in the domain equations have degree* 3*, and the degree of the polynomials in the* 27 *coverage equations can be reduced from* 9 *to* 3*, when applying the reduction of degrees as described in Remark 2.4.12. This system has* 6 *solutions, namely* $\{(0,1,2,2,0,1,1,2,0), (0,2,1,1,0,2,2,1,0), (1,0,2,2,1,0,0,2,1), (1,2,0,0,1,2,2,0,1), (2,0,1,1,2,0,0,1,2), (2,1,0,0,2,1,1,0,2)\}$. *Note the nice linear equalities:* $x_1 = x_5 = x_9$ *and* $x_3 = x_4 = x_8$ *revealed by the GB and reflected in the corresponding positions above, for the 6 solutions.*

Exhaustive Search Using Supercomputing

For systems with more than 20 binary or 10 ternary variables we designed a serial C program to perform exhaustive search. The program uses the ranking and unranking functions described in [165] to efficiently enumerate all combinations of values for the binary or the ternary variables. Obviously this approach readily generalizes to quaternary variables and beyond. For each combination of the variables generated, we solve the equations incrementally, i.e. we look at the equations as constraints that must be satisfied simultaneously and proceed by first checking whether the first equation is satisfied then secondly, by checking whether the second equation is satisfied and so forth. If a particular generated combination of values satisfies all equations then it is a solution and we collect all solutions found in a result file for post-processing. We found it beneficial to use a meta-programming approach, i.e. a bash script that parses the systems of polynomial equations arising from CA constructions automatically and generates the corresponding serial C program, without any intervention by hand. Using meta-programming allows us to produce massive amounts of bug-free and reliable C code with minimal effort.

For systems where our serial C program approach is insufficient, to either produce solutions or verify that no solutions exist, we parallelize our automatically generated C program using MPI. The parallelization is achieved with meta-programming again, in order to make efficient use of the ranking and unranking functions to distribute the workload among the parallel processors. We run our generated C/MPI code on the heterogeneous cluster known as "graham"[2], operated by Compute Canada at the University of Waterloo. Given that the system of polynomial equations arising from CA constructions exhibit a very precise structure and symmetries, we use this structure to distribute the computation not only at the level of variables, but at the level of equations as well. More precisely, by construction, the equations are divided in groups of r equations, where the number of variables featured in each group is a function of r and is significantly smaller than the total number of variables γ. This clearly suggests a two-phase approach to solve the original system:

1. Solve each group of equations independently and in parallel.

2. Look for common solutions among the solutions of all the groups.

The first phase of the above two-phase approach is reminiscent of the selection of subsets of clauses when applying resolution to large CNFs. Subsequently, we revised the first phase, by amalgamating one or more groups together, which has the advantage that fewer solutions are generated, and at the same time may prove insolvability of the system, if one or more groups of equations do not possess any solutions. Using our meta-programming bash script, we are able to run multiple experiments, to determine optimal cut-off points, as far as the number of groups of equations that can be solved independently, with the aim to keep the sizes of the generated solutions files small enough for the second-phase processing. We give below a related example using this approach:

[2]https://docs.computecanada.ca/wiki/Graham, accessed on 01.08.2023

Example 2.4.15 (Computation of optimal CAs) *In this example we want to show how an optimal* $\mathsf{CA}(9;2,3,3)$ *can be computed based on Corollary 2.4.8 using exhaustive search techniques. Therefore, we initialize a* 9×3 *array* X *of symbolic variables*

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} & x_{1,7} & x_{1,8} & x_{1,9} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} & x_{2,7} & x_{2,8} & x_{2,9} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} & x_{3,7} & x_{3,8} & x_{3,9} \end{pmatrix}^T,$$

generate all 27 *domain equations of degree* 3 *and the* 27 *coverage equations of degree* 9*, according to equations* (2.8) *and* (2.9)*, where we use* $(a_1, a_2) = (3, 1)$ *due to Lemma 2.4.9:*

1. $\forall i \in \{1, \ldots, 9\}, \forall j \in \{1, 2, 3\}$

$$x_{i,j}(x_{i,j} - 1)(x_{i,j} - 2) \quad = \quad 0.$$

2. $\forall C \in \binom{\{3\}}{2}, \ \forall (u_1, u_2) \in \{0, 1, 2\}^2$*:*

$$\boldsymbol{prod}(X \cdot \iota^C_{2,3}(3,1) - \mathbf{1} \cdot (u_1, u_2) \cdot (3,1)^T) \quad = \quad 0.$$

Any solution of this system yields an optimal $\mathsf{CA}(9;2,3,3)$*, one of which is e.g.*

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} & x_{1,7} & x_{1,8} & x_{1,9} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} & x_{2,7} & x_{2,8} & x_{2,9} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} & x_{3,7} & x_{3,8} & x_{3,9} \end{pmatrix}^T = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \end{pmatrix}^T.$$

Initial Experiments for Computation of Optimal CAs

Last, in Table 2.4 we list initial experiments, that aim for reconstruction of $\mathsf{CAN}(3,k,2)$ and $\mathsf{CAN}(2,k,3)$ values. Based on the algebraic model for CAs presented in Subsection 2.4.1, we used an implementation of Algorithm 2. For the computation of the precise value of $\mathsf{CAN}(t,k,v)$, the input values t, k, v to Algorithm 2 are specified according to the respective CA instance (given in the first column of Table 2.4). Whereas the input value N, determining the number of rows of the target CA to be constructed, is set to either the exact value of $\mathsf{CAN}(t,k,v)$, or $(\mathsf{CAN}(t,k,v) - 1)$. As expected, in case of $N < \mathsf{CAN}(t,k,v)$ the implementation returned "No CA exists", and for $N = \mathsf{CAN}(t,k,v)$ it was possible to find solutions for all cases documented above. For example, using our algorithms based on algebraic models, it was also possible to check that there do not exist $\mathsf{CA}(7;3,4,2)$ and $\mathsf{CA}(8;2,3,3)$. Further, we computed all optimal $\mathsf{CA}(8;3,4,2)$ and 3022997 optimal $\mathsf{CA}(9;2,3,3)$.

Note that these results report the number of all solutions, i.e. we do distinguish between equivalent CAs. This however, will be done differently in the next section, where amongst other things, we are interested particularly in the number of non-equivalent CAs of a certain size and strength.

CA instance	Solver	# Vars	# Sols	CAN	Reference
$\mathsf{CA}(4;2,3,2)$	GB	12	48	4	[100]
$\mathsf{CA}(5;2,3,2)$	GB	15	1440	4	[100]
$\mathsf{CA}(5;2,4,2)$	GB	20	1920	5	[100]
$\mathsf{CA}(7;3,4,2)$	GB	28	0	8	[101], $(7 < 2^3)$
$\mathsf{CA}(8;3,4,2)$	GB	32	80640	8	[101]
$\mathsf{CA}(8;2,3,3)$	GB	24	0	8	[101], $(8 < 3^2)$
$\mathsf{CA}(9;2,3,3)$	C/MPI	27	$\geq 3 \cdot 10^6$	5	[101]

Table 2.4: The column # **Vars** lists the number of unknowns in the respective equation systems; # **Sols** lists the exact number of solutions, i.e. CAs for the respective instance, except for entries $\geq x$, which indicate that at least x solutions were found. Column **CAN** lists the exact values of CAN for given parameters (t, k, v). Column **Reference** gives a reference where the respective CAN value can be found in the literature, and, in case, a reason why there exists no such CA.

2.5 Balanced Covering Arrays: Definition and Classification

In this section we study subclasses of CAs that fulfill certain balance constraints. In particular, we consider the intersections of classes of CAs and packing arrays (PAs). Arrays appearing in these intersections obey to upper and lower bounds regarding the appearance of t-way interactions. We call these arrays *balanced covering arrays* (balanced CAs) and formulate first observations for which upper and lower bounds on the appearance of t-way interactions it is of interest to consider these structures. A classification of balanced CAs brings to light a dissection of classes of CAs that is partially nested due to the nature of the considered intersections. These dissections can be trivial, containing only a single type of balanced CAs, but can also appear as highly structured containing multiple, nested types of balanced CAs. The results indicate that balanced CAs are a class of designs that is rich of structure.

We begin with the definition of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs in Subsection 2.5.1, and the review of related work in Subsection 2.5.2. Thereafter, in Subsection 2.5.3, we discuss some properties of CAs and packing arrays with regards to their symmetries. In Subsection 2.5.4 we discuss some theoretical results for $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs and provide bounds on $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ that tell us which intersections of $\mathsf{CA}_{\boldsymbol{\lambda}}(N;t,k,v)$ and $\mathsf{PA}_{\boldsymbol{y}}(N;t,k,v)$ are certainly *not* of interest. Subsequently, we exemplify how to adapt a combinatorial construction for CAs to $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs. Finally, Subsection 2.5.5 presents classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs obtained through computational search. The computational search, together with the discussed bounds allows us to determine $\mathsf{CAN}(3, 20, 2) = 18$.

2.5.1 Definition of Balanced Covering Arrays

We begin this section by generalizing the notion of the index λ of a CAs. The balance properties of CAs that we are interested in have previously been partly considered in

[170].

Definition 2.5.1 *We say that a* $\mathsf{CA}(N;t,k,v)$ *is (lower)* balanced *with respect to the vector* $\boldsymbol{\lambda}=(\lambda_1,\ldots,\lambda_t)$*, or* $\boldsymbol{\lambda}$*-balanced for short, if for all* $i\in\{1,\ldots,t\}$ *in each* $N\times i$ *subarray, each* i*-tuple over the alphabet appears at least* λ_i *times as a row.*

In other words a CA is balanced with respect to $(\lambda_1,\ldots,\lambda_t)$, if and only if for all $i\in\{1,\ldots,t\}$ it is a CA of index λ_i and strength i, i.e. a $\mathsf{CA}_{\lambda_i}(N;i,k,v)$. Thus, we denote a $\boldsymbol{\lambda}$-balanced CA also as $\mathsf{CA}_{\boldsymbol{\lambda}}(N;t,k,v)$. For the set of all $\mathsf{CA}_{\boldsymbol{\lambda}}(N;t,k,v)$ we use the notation $\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v)$ and we have the identity: $\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v)=\cap_{i=1}^{t}\mathbb{CA}_{\lambda_i}(N;t,k,v)$.

Definition 2.5.2 *We say that a* $\mathsf{CA}(N;t,k,v)$ *is (upper)* balanced *with respect the vector* $\boldsymbol{y}=(y_1,\ldots,y_t)$*,* $\boldsymbol{y}$*-balanced for short, if for all* $i\in\{1,\ldots,t\}$ *in each* $N\times i$ *subarray, each* i*-tuple over the alphabet appears at most* y_i *times as a row.*

Arrays with upper bounds on the occurrence of t-way interactions have been studied previously. Stevens and Mendelsohn investigated *packing arrays* (PAs) for strength $t=2$ in [184] and [185]. A generalized notion for arbitrary t was later considered in [186], which we adopt below:

Definition 2.5.3 *Let* $N,k,s,t,\boldsymbol{y}$ *be positive integers. A packing array* $\mathsf{PA}_{\boldsymbol{y}}(N;t,k,v)$ *is an* $N\times k$ *array with entries from an alphabet of size* v*, with the property that in every set of* t *columns, any* t*-tuple of symbols from the alphabet occurs in at most* $\boldsymbol{y}$ *rows. If the index* $\boldsymbol{y}=1$*, then it is usually omitted and we simply write* $\mathsf{PA}(N;t,k,v)$*.*

In analogy with the case of CAs, if a given array is a $\mathsf{PA}_{y_i}(N;i,k,v)$ for given y_i and $i\in\{1,\ldots,t\}$, we denote it as $\mathsf{PA}_{\boldsymbol{y}}(N;t,k,v)$ where $\boldsymbol{y}=(y_1,\ldots,y_t)$; and the set of all packing arrays $\mathsf{PA}_{\boldsymbol{y}}(N;t,k,v)$ by $\mathbb{PA}_{\boldsymbol{y}}(N;t,k,v)$. The maximal number N for which a $\mathsf{PA}(N;t,k,v)$ exists is called the *packing array number* and is denoted by $\mathsf{PAN}(t,k,v)$.

Having recalled these notions, we can describe the (covering) arrays of interest as lower $\boldsymbol{\lambda}$-balanced and upper $\boldsymbol{y}$-balanced arrays. These arrays appear as the intersection of *covering arrays* and *packing arrays*:

Definition 2.5.4 *An* $N\times k$ *array* A *over a* v*-ary alphabet is called (lower)* $(\lambda_1,\ldots,\lambda_t)$*-balanced and (upper)* $(y_1,\ldots,y_t)$*-balanced if and only if:*

$$A\in\bigcap_{i\in\{1,\ldots,t\}}\mathbb{CA}_{\lambda_i}(N;i,k,v)\cap\mathbb{PA}_{y_i}(N;i,k,v)=\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v)\cap\mathbb{PA}_{\boldsymbol{y}}(N;t,k,v). \quad (2.15)$$

Since in this section we are primarily interested in lower $(\lambda_1,\ldots,\lambda_t)$*-balanced arrays with* $\lambda_t\geq 1$*, we call the arrays appearing in the intersection* $\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v)\cap\mathbb{PA}_{\boldsymbol{y}}(N;t,k,v)$

also $(\boldsymbol{\lambda}, \boldsymbol{y})$*-balanced CAs and we denote them by* $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,v)$*. Similarly, we may abbreviate* $\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v) \cap \mathbb{PA}_{\boldsymbol{y}}(N;t,k,v)$ *with* $\mathbb{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,v)$ *when a more compact notation is beneficial.*

We further define the maximum number of columns for which a balanced CA for given N,t,v *and* $\boldsymbol{\lambda}$*,* $\boldsymbol{y}$ *exists:*

$$\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v) := max\{k : \exists \mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,v)\}.$$

Informally we will refer collectively to objects appearing in intersections as in (2.15) as *balanced CAs*, especially when $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ are clear from the context or do not need to be specified more precisely. The reason we extend the notion of lower $\boldsymbol{\lambda}$-balance from [170] by the notion of upper $\boldsymbol{y}$-balance is that it allows a finer classification of balanced CAs, as is elucidated in the following example.

Example 2.5.5 *For* $N = 7, t = 2$ *and* $v = 2$ *consider the following vector* $\boldsymbol{\lambda} = (3,1)$*. Both arrays* A_1 *and* A_2 *given below are lower* $\boldsymbol{\lambda}$*-balanced: each symbol appears at least three times per column and each pair appears at least once in the two columns. However, only* A_2 *is also upper* $\boldsymbol{y}$*-balanced for* $\boldsymbol{y} = (4,2)$*. In* A_1 *the pair* $(1,0)$ *appears three times, violating the upper bound* $\boldsymbol{y}_2 = 2$*. At the same time other pairs,* $(0,0)$ *and* $(1,1)$ *occur only once. In that regard, we will generally recognize* A_2 *as more balanced than* A_1*.*

$$A_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

We know already that the intersection in (2.15) is not always empty, as for vectors $\boldsymbol{\lambda} = \boldsymbol{y} = (v^{t-1}\lambda, v^{t-2}\lambda, \ldots, v\lambda, \lambda)$ and suitable parameters N,t,k,v the elements in the resulting set are Orthogonal Arrays of index λ, i.e. $\mathsf{OA}_\lambda(N,k,v,t)$, using the notation from [19] and [24]. For other vectors $\boldsymbol{\lambda}, \boldsymbol{y}$ and parameters N,t,k,v the intersection in relation (2.15) may be empty, which can be more or less trivial. It is trivial, when $\lambda_i > \boldsymbol{y}_i$ for some i, but it is less so, when we consider the case where $N = 96$, $t = 6$, $k = 9$, $v = 2$, $\boldsymbol{\lambda} = (48,24,12,6,2,1)$ and $\boldsymbol{y} = (48,24,12,6,3,1)$. In the latter case any array in the intersection in relation (2.15), when considered for strength $t = 4$, would constitute an $\mathsf{OA}(96,9,2,4)$, which does not exist according to [187, Theorem 18]. To give further examples, we deduce from the experiments in Subsection 2.5.5, that there does not exist any $\mathsf{CA}^{\boldsymbol{y}}_{(26,13,6,2,1)}(52;5,8,2)$, for arbitrary $\boldsymbol{y}$, while our computational search shows that there certainly exists a $\mathsf{CA}^{(26,14,8,5,4)}_{(26,12,6,2,1)}(52;5,8,2)$. More precisely, as we will see, there are exactly 5 non-equivalent (the notion of equivalence will be discussed in Subsection 2.5.3) of such balanced CAs, as will be shown in the Table under [188].

The primary purpose of the study below is to investigate the existence of *balanced* CAs and their properties, i.e. we want to answer the following questions: *For which parameters are the intersections in relation* (2.15) *non-empty? - And how many (non-equivalent) arrays appear in these intersections? Are there any assertions or connections that can be established related to these intersections?* A secondary goal of our study is to investigate potential *properties of (optimal) CAs with regards to balance.*

2.5.2 Related Work

We review the related literature in three parts. First, related work on classification of CAs, i.e. identification of all non-equivalent CAs for given parameters N, t, k, v, and related literature on PAs; second, related work on formal methods for CA generation, with a focus on SAT solving for CA generation; and third, related works that consider a notion of balance for CAs in the widest sense.

Related Work on Covering Array Classification and Packing Arrays

In [153] the problem of orthogonal array, CA and PA generation is formulated as an integer linear program and the problem of classifying all isomorphism classes of orthogonal arrays is reduced to finding all isomorphism classes of non-negative integer solutions to systems of linear equations under its symmetry group. A branch-and-cut algorithm is used for solving the integer programming problems that arise. The classification of some orthogonal arrays, some optimal CAs and PAs with a maximal number of rows is given. However, the authors of [153] do not consider intersections of CAs and PAs.

The authors of [159] study CAs and generalizations (radius-covering arrays) and discuss theoretical results on the size of CAs. These theoretical bounds are supplemented with computational techniques in order to derive new upper bounds on the covering array number which leads to exact computations in some cases. Another focus of [159] lies on the classification of CAs and radius-covering arrays, as the work presents also some exact numbers of non-equivalent optimal CAs.

In [189] an algorithm for classification of CAs is presented, which uses column extension with a minimality check to generate all non-equivalent CAs. The values in a column are assigned one-by-one starting from the top. Several rules and data structures are used to decide on the allowed values for each cell. If no new value is available for a cell, the algorithm backtracks to the previous cell. Whenever a value is assigned to the last cell of the current column, a minimality check is performed to decide whether the partial CA is discarded or further extended.

In a more recent work [161] a classification of CAs of strength 2 with $v > 2$ is presented. A special focus lies on so called *uniform* CAs, which are introduced as $\mathsf{CA}(N; t, k, v)$ where every symbol occurs $\lfloor N/v \rfloor$ or $\lceil N/v \rceil$ times in each column. A column extension algorithm is used to find and count all non-equivalent CAs. The CAs are represented as colored graphs and a solver (*Cliquer* or *libexact*) for detecting equivalent graphs is used

to count only non-equivalent CAs. The paper also presents theoretical results on the size of *uniform* CAs.

For Packing Arrays, to the best of my knowledge the first dedicated studies (at least under this name) have been conducted by Stevens and Mendelsohn in [184] and [185]. Previously, these structures have been partially investigated also under the names *transversal packings* ([190]) and *mutually orthogonal partial latin squares* ([191]). The authors of [184] and [185] introduce packing arrays as a natural generalization of orthogonal arrays (of strength two and index one), where in any subarray comprised of two columns, each pair of symbols (over the given alphabet) appears *at most once*, instead of *exactly once* as for OAs. Further, pairs of rows of a $\mathsf{PA}(N; t, k, v)$ have Hamming distance $k - t + 1$ and hence its rows constitute the codewords of an error-correcting code with minimum distance $d = k - t + 1$, as also mentioned in [184] and [185]. The work in [185] introduces the notion of packing arrays with disjoint rows, i.e. PAs with a set of at least n rows that pairwise differ in each position. Thereafter, upper bounds on the number of columns in such PAs are established. Notably is the modification of the Plotkin bound (see also [19, Theorem 1.158]) to apply for PAs with disjoint rows. Disjoint rows are essential for recursive constructions of PAs, which are also described in [185], and lead to new lower bounds on the maximum number of columns in PAs. Finally, the discussed bounds are instantiated for specific parameters and tables with upper and lower bounds for the largest number of rows possible in PAs are given. In [184] connections between PAs and other classes of combinatorial designs, such as resolvable transversal designs and especially packing designs are discussed. Further, lower bounds on the number of rows of PAs are given, based on constructions of PAs from packing designs. A bound for PAs with $v - 1$ disjoint rows over v-ary alphabets, which is derived from known values of packing designs is given. This bound is at least as strong as the modified Plotkin bound from [185]. Finally, in [186] the notion of packing arrays, as considered in this paper, is given and discussed in the context of generalized packing designs.

Related Work on SAT Solving for Covering Array Generation

To the best of my knowledge, the first SAT formulation for covering array generation was presented in [69]. The authors describe a formula encoding the entire CA and use local search to find a solution. Thereby, the authors describe different matrix models in order to derive formulas encoding the entire CA to be generated. The first matrix model encodes the value of every entry in the CA to be constructed. The second matrix model encodes for every row and column combination which t-tuple is covered. Finally, a third matrix model, combining the first two matrix models through channelling constraints is described. All three matrix models are solved with an exact CSP approach but only the last one is solved with a local search SAT solver. For SAT, the variables of the matrix model are mapped to binary variables via a one-hot encoding. In [192] complete SAT solvers were applied to the formula from [69] and two of its modified versions. For these versions, the same sets of variables were used, but an order encoding was used instead of a one-hot encoding. Both approaches [69] and [192] have as input the number of

rows N, the number of columns k, the strength t and the alphabet size v. For a given parameter combination $(N; t, k, v)$ the existence of a covering array is determined from the generated SAT formula. The goal is typically to find an optimal CA for given (t, k, v). The previously mentioned approaches require trying different values for N. Optimality of an existing $\mathsf{CA}(N; t, k, v)$ is shown when the formula for $\mathsf{CA}(N-1; t, k, v)$ is proven unsatisfiable.

A different approach is followed in [193], where additionally the number of used rows in the matrix model is encoded and minimized using a partial MaxSAT approach. A variable r_i is set to true if a variable in row i got assigned a value. With MaxSAT the number of satisfied $\neg r_i$ (the number of unused rows) is maximized. This method requires setting an upper bound for N so that one MaxSAT call is enough to find the best value for N.

Finally, we mention that SAT solving has also been used for generating CAs with constraints. We may refer the interested reader to [194] for exact algorithms and to [195] or [196] for incomplete algorithms.

Existing Balance Notions for Covering Arrays

The notion of lower $\boldsymbol{\lambda}$-balance was previously used already as a guiding heuristic in a column extension algorithm for CA generation in [170]. As this work can be considered a starting point of the work presented in this section, we give a more detailed review in the beginning of Subsection 4.

In [197] an exact search algorithm for CA generation using backtracking is described. The algorithm individually assigns values to the variables of a matrix model and uses constraint propagation when possible. If constraints are violated, i.e. no CA exists with the current (partial) assignment, backtracking is performed to remove (individual) conflicting values. Aside from coverage and domain constraints, various pruning techniques are used. One of them is called Sub-Combination Equalization Heuristic (SCEH), which is closely related to the proposed $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance, as it also quantifies the appearance of tuples of length up to a specified bound t'. We restate this condition briefly with our own notation (recall that $\rho_A(\tau)$ denotes the number of rows of A that cover τ): Let A be a $\mathsf{CA}(N; t, k, v)$, then A fulfills the SCEH-condition for $t' \leq t$, if and only if:

$$\text{For all } s \in \{1, \ldots, t'\} \text{ and all } s\text{-way interactions } \tau_1, \tau_2\text{: } |\rho_A(\tau_1) - \rho_A(\tau_2)| \leq 1. \quad (2.16)$$

Clearly, we can express the SCEH-condition using the notion of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance, as it is equivalent to requiring $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance with $\lambda_i = \lfloor N/v^i \rfloor$ and $\boldsymbol{y}_i = \lfloor N/v^i \rfloor + 1$, for all $i \in \{1, \ldots, t'\}$.

In [198] a greedy column extension algorithm that makes use of a pseudo-Boolean constraint solver is described. Balance constraints are imposed on the appearance of symbols in individual columns, by requiring that each symbol appears $\lfloor \frac{N}{v} \rfloor$ or $\lceil \frac{N}{v} \rceil$ times per column.

Finally, we want to mention the work in [199], where the In-Parameter-Order algorithm, a widely used strategy for CA generation for reasonably fast and small CA generation, is combined with a *balance* heuristic that attempts to make each value appear as equally often as possible in each column of the CA to be constructed. The authors do mention that this balance heuristic can reduce the size of generated CAs in some cases, and together with a reordering strategy can drastically improve the fault detection rate of combinatorial test suites derived from these CAs.

2.5.3 Preliminaries

In this subsection we revisit and summarize some previously known results and notions in order to make this paper more self contained.

Symmetries of Covering Arrays

As for many other matrix models, the defining properties of CAs, PAs and thus balanced CAs, stay invariant under the following actions. We denote by S_n the symmetric group on n elements.

1. **Column permutations:** If the $N \times k$ array $A = (c_1, \ldots, c_k)$ with columns c_i is a CA (PA), then $(c_{\pi(1)}, \ldots, c_{\pi(k)})$ is also a CA (PA) for any permutation $\pi \in \mathcal{S}_k$ of the columns. Since a CA cannot contain duplicate columns (only t-way interactions with equal entries in these two positions would be covered), there are $k!$ pairwise *different* column permutations of a CA.

2. **Row permutations:** If the $N \times k$ array $A = (r_1, \ldots, r_N)^T$ with rows r_i is a CA (PA), then $(r_{\rho(1)}, \ldots, r_{\rho(N)})^T$ is also a CA (PA) for any permutation $\rho \in \mathcal{S}_N$ of the rows. When an array does not contain any duplicate rows, then there are $N!$ pairwise non-identical row permutations of it.

3. **Symbol permutation per column:** If the $N \times k$ array $A = (c_1, \ldots, c_i, \ldots, c_k)$ with columns c_i is a CA (PA), then $(c_1, \ldots, \sigma(c_i), \ldots, c_k)$ is also a CA (PA), where $\sigma \in \mathcal{S}_v$ is a permutation of the symbols appearing in the i-th column. Because each symbol appears at least once in each column of a CA, each permutation that acts on the symbols of a column generates a different CA. As they can be applied to all columns of a CA independently from each other, there are $v!^k$ pairwise different CAs that can be obtained through symbol permutations.

Two CAs (PAs) A and B are called *equivalent*[3] if A can be converted to B via any of the aforementioned actions, and *non-equivalent* otherwise. We adopt this notion of equivalence also for balanced CAs. The following proposition records some basic observations.

[3] In the literature also the term *isomorph* is used, as in [189] or [161]. However, we use the same terminology as in [159] or [69].

Proposition 2.5.6 *For the equivalence actions on arrays we have:*

(1) *Column, row and symbol permutations (per column) can be commuted when applied to v-ary $N \times k$ arrays. Specifically, any sequence s of such equivalence actions can be represented by applying a suitable column permutation $\pi \in \mathcal{S}_k$, a suitable row permutation $\rho \in \mathcal{S}_N$ and suitable symbol permutations $\sigma_i \in \mathcal{S}_v$, $1 \leq i \leq k$ in the k in the columns: $s = \pi \circ \rho \circ \sigma_1 \circ \ldots \circ \sigma_k$.*

(2) *For a given $\mathsf{CA}(N; t, k, v)$, the number of* distinct CAs *that are equivalent up to column and symbol permutations to the original CA is $k! \cdot (v!)^k$.*

(3) *For the arrays in $\mathbb{CA}(N; t, k, v)$ there are $k! \cdot N! \cdot (v!)^k$* different actions *that can be composed of column permutations, row permutations and symbol permutations per column.*

Proof: It can be easily shown that the property in (1) does hold for general matrix models.

To show (2), assume we are given $A \in \mathbb{CA}(N; t, k, v)$, say $A = (c_1, \ldots, c_k) \in \{0, \ldots, v-1\}^{N \times k}$. Assume there exist two different equivalence actions composed of column permutations and permutations of the symbols in the k columns that produce identical arrays. Then there must be two different columns c_{i_1}, c_{i_2} of A and symbol permutations $\sigma, \bar{\sigma}$, such that $\sigma(c_{i_1}) = \bar{\sigma}(c_{i_2})$, or equivalently $c_{i_2} = \bar{\sigma}^{-1}(\sigma(c_{i_1}))$. But then in the original array A, in the columns c_{i_1} and c_{i_2} only pairs $(u, \bar{\sigma}^{-1}(\sigma(u)))$, for $u \in \{0, \ldots, v-1\}$, appear as rows. Thus, any subarray comprised of t columns that involves columns c_{i_1} and c_{i_2} only covers t-way interactions that contain the v pairs $\{(i_1, u), (i_2, \bar{\sigma}^{-1}(\sigma(u))) : u = 0, \ldots, v-1\}$. A contradiction to A being a $\mathsf{CA}(N; t, k, v)$. Thus two different equivalence actions composed of column permutations and symbol permutations in the k columns, yield two distinct arrays. As we can compose $k! \cdot (v!)^k$ such equivalence actions, we have that many CAs equivalent to A, which are pairwise different.

To show the assertion in (3), assume given an action $s = \pi \circ \rho \circ \sigma_1 \circ \ldots \circ \sigma_k$, with $\pi \in \mathcal{S}_k$, $\rho \in \mathcal{S}_N$, and $\sigma_i \in \mathcal{S}_v$, for all $i \in \{1, \ldots, k\}$. We first show in (a) and (b) below that only the action id that consists only of identity permutations, i.e. $\pi = id_{\mathcal{S}_k}$, $\rho = id_{\mathcal{S}_N}$, and $\sigma_i = id_{\mathcal{S}_v}$ for all $i \in \{1, \ldots, k\}$, acts as identity on $\mathbb{CA}(N; t, k, v)$.

(a) For (3) we first show that for an action containing a non-identity row permutation, in each equivalence class there is at least one CA where this action acts as non-identity. So let $\rho \neq id_{\mathcal{S}_N}$, and write $s = \rho \circ \zeta$ with $\zeta = \pi \circ \sigma_1 \circ \ldots \circ \sigma_k$ for brevity. Given a CA $A = (r_1, \ldots, r_N)^T \in \mathbb{CA}(N; t, k, v)$ with rows r_i, if $s(A) \neq A$ we are done. If $s(A) = A$, then there are two different rows of A: r_{i_1}, r_{i_2} with $i_1 \neq i_2$, $\rho(i_1) = i_2$ and $\zeta(r_{i_1}) = r_{i_2}$. We can find another row r_{i_3} that satisfies $\zeta(r_{i_3}) \neq r_{i_2}$, e.g. by considering the preimage under s of any row of A distinct from r_{i_1} and r_{i_2}. We can find such a row since a CA of strength $t \geq 2$ must contain at least four different rows. When we exchange the rows

r_{i_1} and r_{i_3} of A by transposition τ_{i_1,i_3}, we obtain a CA that is equivalent to A. The i_2-th row of $\tau_{i_1,i_3}(A)$ is r_{i_2}, but when we apply s to $\tau_{i_1,i_3}(A)$, we find in the i_2-th row of $s(\tau_{i_1,i_3}(A)) = \zeta(\rho \circ \tau_{i_1,i_3}(A))$ the row $\zeta(r_{i_3}) \neq r_{i_2}$, and hence $s(\tau_{i_1,i_3}(A)) \neq \tau_{i_1,i_3}(A)$.

(b) Second, in case an action does not permute the rows (i.e. $\rho = id_{\mathcal{S}_N}$ in the representation obtained from (1)), then by (2) we get that this action acts as non-identity on any CA in $\mathbb{CA}(N;t,k,v)$.

Finally, assume $s_1 = \pi \circ \rho \circ \sigma_1 \circ \ldots \circ \sigma_k$ and $s_2 = \pi' \circ \rho' \circ \sigma'_1 \circ \ldots \circ \sigma'_k$ are equivalence actions on $\mathbb{CA}(N;t,k,v)$ composed by column, row and symbol permutations, that satisfy $s_1(A) = s_2(A)$ for all $A \in \mathbb{CA}(N;t,k,v)$. Then we have $s_2^{-1} \circ s_1(A) = A$ for all $A \in \mathbb{CA}(N;t,k,v)$. From (1) we know that $s_2^{-1} \circ s_1$ can be written as $s_2^{-1} \circ s_1 = \pi'^{-1} \circ \pi \circ \rho'^{-1} \circ \rho \circ \sigma_1'^{-1} \circ \sigma_1 \circ \ldots \circ \sigma_k'^{-1} \circ \sigma_k$. From (a) above we get that the row permutation in $s_2^{-1} \circ s_1$, must be the identity $id_{\mathcal{S}_N}$, and hence $\rho = \rho'$. Then from (2) it also follows that $\pi = \pi'$ and $\sigma_i = \sigma'_i$ for all $1 \leq i \leq k$. Assume otherwise $\pi \neq \pi'$ or $\sigma_i \neq \sigma'_i$ for any $1 \leq i \leq k$, then by (2) we had $s_2^{-1} \circ s_1(A) \neq A$, even for all $A \in \mathbb{CA}(N;t,k,v)$ – a contradiction to $s_1(A) = s_2(A)$ for all $A \in \mathbb{CA}(N;t,k,v)$.

We have shown that any combination of a column, a row and symbol permutations in the k columns yields a different action, therefore there are $k! \cdot N! \cdot (v!)^k$ different actions. □

These results may appear trivial, however the following small example illustrates, that we cannot assume (in general) that all $k! \cdot N! \cdot (v!)^k$ equivalence actions on a $\mathsf{CA}(N;t,k,v)$ do yield *distinct* arrays, i.e. some arrays stay fixed under some actions. Thus not all equivalence classes of $\mathbb{CA}(N;t,k,v)$ are of maximal size.

Example 2.5.7 *Consider the following* $\mathsf{CA}(5;2,4,2)$ *in equation* (2.17) *and apply as column permutation* π *the transposition of the first and second column* $\pi = \tau_{1,2} \in \mathcal{S}_4$*, and as row permutation* ρ *the transposition of the third and fourth row* $\rho = \tau_{3,4} \in \mathcal{S}_5$ *to* A*, which yields again the matrix* A_{min}*:*

$$A_{min} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \rho \circ \pi(A_{min}). \tag{2.17}$$

To the best of my knowledge, the question how large the equivalence class of a given $\mathsf{CA}(N;t,k,v)$ is, has not been addressed in the literature in this general form, and we will also not solve this problem in the following. However, the answers for some specific instances can be extracted from computational results on CA classifications from the literature, and the results presented in this work. For example we know that there are exactly 1920 distinct CAs in $\mathbb{CA}(5;2,4,2)$ (see [67]), which are all equivalent as we will also see later in Subsection 2.5.5. Thus, the $4! \cdot 5! \cdot 2^4 = 46080$ different equivalence actions on the array given in equation (2.17) produce "only" 1920 distinct (yet equivalent)

arrays. However, the size of equivalence classes grows dramatically for larger parameters N, k, v, which is partly explained by Proposition 2.5.6 item (2). To give another example, there exist $2.61 \cdot 10^7$ distinct $\mathsf{CA}(9; 2, 4, 3)$'s that are all equivalent; as we have witnessed in our computational experiments and can also be found in the literature, combining the information of [200, Table 4] and [159, Table 7]. Even though not all equivalence classes are of maximal size, traversing or producing all equivalent solutions consumes a lot of resources, which makes apparent the importance of symmetry breaking in order to remove equivalent solutions.

Symmetry Breaking

The importance of symmetry breaking for CA generation was already recognized in [69]. Symmetry breaking for general matrix models with row and column symmetries has also been subject to investigation, for example in [200], [201] and [202]. Symmetries in such matrix models can be reduced by imposing ordering constraints on the rows and the columns, e.g. by lexicographic (lex) ordering constraints: $(x_1, \ldots, x_n) \leq_{lex} (x'_1, \ldots, x'_n)$, where

$(x_1, \ldots, x_n) \leq_{lex} (x'_1, \ldots, x'_n)$ if and only if $x_i \leq x'_i$ at every position i with $x_j = x'_j$ for all $j < i$.

This means the two vectors have to be equal or, if they differ, there has to be $x_i < x'_i$ at the first position i where $x_i \neq x'_i$. Imposing such constraints on all consecutive rows (columns), removes all row (column) symmetries, as any row (column) permutation of an array that satisfies all these constraints would lead to a violation of some constraint.

When using different symmetry breaking techniques it is important to ensure that they do not interfere with each other. For example, when the rows of a matrix are already lexicographically ordered we can not freely permute the columns without undoing the row ordering. Nevertheless, the fact that row- and column-lex ordering can be used simultaneously (called *double-lex*) was shown in [201, Theorem 1], i.e. for matrix models with row and column symmetries in each equivalence class there exists a matrix that is both, row- and column-lex ordered. However, as shown in [200, Theorem 2], imposing row- and column-lex constraints does not remove *all* symmetries, as there exist matrix models where double-lex leaves a significant number of symmetries. The following example demonstrates that CAs are a class of arrays, where not all symmetries are removed by double-lex constraints.

Example 2.5.7 (continuing from p. 86) *We consider again a* $\mathsf{CA}(5; 2, 4, 2)$

$$A_{RowColVarLex} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \tag{2.18}$$

which is lexicographically row-wise ordered, as well as lexicographically column-wise ordered and thus satisfies the double-lex constraints. However, when we permute columns by $\pi = \tau_{1,4} \in \mathcal{S}_4$, i.e. the transposition of the first and fourth column, and permute the rows by $\rho = (14532)$ (denoted in one-line notation), we obtain the equivalent array A_{min} given in (2.17)*, which also satisfies the double-lex constraints.*

To remove all equivalent solutions under arbitrary symmetries, the *lex-leader* technique can be used (see e.g. [203]), which can be applied to matrices as follows: Instead of considering a two dimensional array the matrix is linearized. This means it is represented as a vector, which can be done for example by reading the matrix' entries *column-wise*, which is also called *column-major order*. We can thus sort two matrices lexicographically by sorting their linearizations respectively $\leq_{lex}$. For each equivalence action in the symmetry group of the matrix model, a constraint is formulated that enforces that the linearized representation of the solution matrix is lexicographically smaller than the linearized representation of the matrix under this equivalence action. A solution matrix that satisfies all these constraints is then the unique lexicographically smallest element in its equivalence class and is called the *lex-leader*. This technique has the disadvantage that it requires one constraint per element in the symmetry group. Example 2.5.8 visualizes how such constraints are generated for selected actions.

Example 2.5.8 *One way to order the matrix for lex-leader symmetry breaking is a column-wise linearization:*

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \\ x_{10} & x_{11} & x_{12} \end{pmatrix} \mapsto (x_1, x_4, x_7, x_{10}, x_2, x_5, x_8, x_{11}, x_3, x_6, x_9, x_{12}). \tag{2.19}$$

For fixed N and k this assigns a unique vector to each $N \times k$ matrix. These can be linearly ordered by the lexicographical order $\leq_{lex}$. Thus each equivalence class has a unique minimal element with regard to this linearization and order. The lex-leader method imposes one constraint per equivalence action to find the minimal element.

For example, the array resulting when interchanging the first two columns has the linearization $(x_2, x_5, x_8, x_{11}, x_1, x_4, x_7, x_{10}, x_3, x_6, x_9, x_{12})$. To rule out that this column permutation produces a lexicographically smaller solution, the following constraint is enforced:

$$(x_1, x_4, x_7, x_{10}, x_2, x_5, x_8, x_{11}, x_3, x_6, x_9, x_{12}) \leq_{lex} (x_2, x_5, x_8, x_{11}, x_1, x_4, x_7, x_{10}, x_3, x_6, x_9, x_{12}).$$

Imposing such constraints for all equivalence actions of the considered matrix model removes all symmetries.

As mentioned in [200] double-lex is consistent with and entailed by lex-leader constraints with a column-wise (or row-wise) linearization of the matrix.

Example 2.5.7 (continuing from p. 86) *The* $\mathsf{CA}(5;2,4,2)$ A_{min} *given in* (2.17) *is column-wise lexicographically smaller than* $A_{RowColVarLex}$ *given in* (2.18)*. In fact, it is a lex-leader with regard to column-wise linearization considering all equivalence actions of CAs.*

Further, in [200, Theorem 1] it was shown that row-wise lex leader constraints can be checked in $O(n!nm \log m)$ time for an $n \times m$ matrix with row and column symmetries. In order to check if a matrix A is a lex-leader of its equivalence class, instead of iterating over all matrices equivalent to A, we can select one type of equivalence action and iterate over the matrices derivable from A by any of the other equivalence actions. For every such derived matrix we then find the lex-leader of the matrices that are reachable via the equivalence actions of the selected type. For this task, semantics of the selected type of equivalence action can be exploited, e.g. with a sorting algorithm. See [202] for more details. If any resulting matrix is lexicographically smaller than the original matrix A, then A cannot be a lex-leader. For example, for CAs we can iterate over all the equivalent matrices obtained through column and value permutations of a matrix and sort its rows, in order to check it for lex-leadership. Further, in [202] lex-leader checks of partial solutions are considered, which can speed up the search process in these exhaustive searches. These techniques were also used in [139], to derive the results discussed later in Subsection 2.5.5.

2.5.4 Properties for $\boldsymbol{\lambda}$-balance and $\boldsymbol{y}$-balance

In this subsection we investigate assertions and connections that can be established for $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs. For the purpose of classification, we are interested in bounds on $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors that can tell us for which vectors it is relevant to perform a computational search and classification. When we impose too stringent balance constraints by means of $\boldsymbol{\lambda}$ and $\boldsymbol{y}$, there may not exist any $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs. On the other hand when the entries of $\boldsymbol{\lambda}$ are very small and those of $\boldsymbol{y}$ are very large, all CAs will be balanced with respect to these vectors and there is no need for computational search with further weakened balance constraints. Some bounds on $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors can be obtained from basic observations, which we formulate as a series of propositions in the next subsection.

As mentioned in the introduction, our secondary motivation is to investigate potential connections between $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance and optimal CA generation. One question arising is whether $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance can be used as a pruning heuristic in computational searches for (optimal) CAs. For that purpose one would be interested in finding the most restrictive $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors that maximally reduce the search space, while ensuring that not all solutions of a certain size, e.g. CAs with $\mathsf{CAK}(N;t,v)$ number of columns are removed from it. For this however, we can not provide similar propositions that provide sufficient conditions for search space reduction and the existence of balanced CAs of a certain size. Instead we review previously formulated hypotheses regarding balance constraints and optimal CAs to provide a context, where we extend and comment on these hypotheses where possible (Subsection 2.5.4).

Basic Properties and Bounds for $\boldsymbol{\lambda}$ and $\boldsymbol{y}$

First and foremost we want to state explicitly that if $\boldsymbol{\lambda}' \leq \boldsymbol{\lambda}$, i.e. $\lambda'_i \leq \lambda_i, \forall i\{1,\ldots,t\}$, each $\boldsymbol{\lambda}$-balanced CA is also $\boldsymbol{\lambda}'$-balanced, i.e. $\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v) \subseteq \mathbb{CA}_{\boldsymbol{\lambda}'}(N;t,k,v)$. In an analogous manner, we have the dual result for PAs: if $\boldsymbol{y} \leq \boldsymbol{y}'$ then $\mathbb{PA}_{\boldsymbol{y}}(N;t,k,v) \subseteq \mathbb{PA}_{\boldsymbol{y}'}(N;t,k,v)$. Thus when $\boldsymbol{\lambda}' \leq \boldsymbol{\lambda}$ and $\boldsymbol{y} \leq \boldsymbol{y}'$ we speak for $(\boldsymbol{\lambda}', \boldsymbol{y}')$ of *weaker* or *less restrictive* and for $(\boldsymbol{\lambda}, \boldsymbol{y})$ of *stronger* or *more restrictive* balance vectors.

Proposition 2.5.9 *Assume there exists an* $A \in \mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v) \neq \emptyset$ *with* $\boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_t)$, *then*

$$\lambda_i \quad \leq \quad \left\lfloor \frac{N}{v^i} \right\rfloor, \text{ for all } 1 \leq i \leq t. \tag{2.20}$$

If there exists an $B \in \mathbb{PA}^{\boldsymbol{y}}(N;t,k,v) \neq \emptyset$ *with* $\boldsymbol{y} = (y_1,\ldots,y_t)$, *then*

$$y_i \quad \geq \quad \left\lceil \frac{N}{v^i} \right\rceil, \text{ for all } 1 \leq i \leq t. \tag{2.21}$$

Proof: The inequality in (2.20) is clear: since in any i columns of A each of the v^i i-tuples is covered at least λ_i times we have $N \geq \lambda_i v^i$. In an analogous manner, we get the inequality in (2.21) since in any i columns of B each of the v^i i-tuples is covered at most y_i times we have $y_i v^i \geq N$. □

The bound in the propositions below, was already used in [170] for $v = 2$.

Proposition 2.5.10 *Let* A *be a* $\mathsf{CA}_{\boldsymbol{\lambda}}(N;t,k,v)$ *with* $\boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_t)$, *and for all* $1 \leq i < t$ *let* τ_i *denote an arbitrary* i*-way interaction, then it holds that*

$$\rho_A(\tau_i) \quad \geq \quad v \cdot \lambda_{i+1}. \tag{2.22}$$

Recall that $\rho_A(\tau)$ *denotes the number of rows of* A *that cover* τ.

Proof: We have $A \in \mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v)$. For $1 \leq i < t$ let $\tau_i = \{(p_1,u_1),\ldots,(p_i,u_i)\}$ be an arbitrary i-way interaction, and $1 \leq p_{i+1} \leq k$ with $p_{i+1} \notin \{p_1,\ldots,p_i\}$. Then $\rho_A(\tau_i \cup \{(p_{i+1},u)\}) \geq \lambda_{i+1}$ for each $u \in \{1,\ldots,v-1\}$. For $u \neq u'$ we have pairwise different $(i+1)$-way interactions $\tau_i \cup \{(p_{i+1},u)\}$ and $\tau_i \cup \{(p_{i+1},u')\}$ which must be covered in different rows. Thus we get $\rho_A(\tau_i) = \sum_{u=0}^{v-1} \rho_A(\tau_i \cup \{(p_{i+1},u)\}) \geq v\lambda_{i+1}$. □

When we restrict the precondition in the above proposition and assume that we only know that $\lambda_t = \lambda$, then the assertion simply says that every $\mathsf{CA}_{\lambda}(N;t,k,v)$ is $(\lambda_1,\ldots,\lambda_t)$-balanced for $\lambda_t = \lambda$ and $\lambda_i \leq \lambda \cdot v^{t-i}$, for all $1 \leq i < t$.

Proposition 2.5.11 *Let* A *be a* $\mathsf{PA}_{\boldsymbol{y}}(N;t,k,v)$ *with* $\boldsymbol{y} = (y_1,\ldots,y_t)$, *and for all* $1 \leq i < t$ *let* τ_i *denote an arbitrary* i*-way interaction, then it holds that*

$$\rho_A(\tau_i) \quad \leq \quad v \cdot y_{i+1}. \tag{2.23}$$

Proof: We have $A \in \mathbb{PA}_{\boldsymbol{y}}(N;t,k,v)$. For $1 \leq i < t$ let $\tau_i = \{(p_1,u_1),\ldots,(p_i,u_i)\}$ be an arbitrary i-way interaction, and $1 \leq p_{i+1} \leq k$ with $p_{i+1} \notin \{p_1,\ldots,p_i\}$. Then $\rho_A(\tau_i \cup \{(p_{i+1},u)\}) \leq \boldsymbol{y}_{i+1}$ for each $u \in \{1,\ldots,v-1\}$, and thus $\rho_A(\tau_i) = \sum_{u=0}^{v-1} \rho_A(\tau_i \cup \{(p_{i+1},u)\}) \leq v \cdot \boldsymbol{y}_{i+1}$. □

Proposition 2.5.12 *Assume $N \in \mathbb{N}, \boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_t) \in \mathbb{N}^t$ and $\boldsymbol{y} = (\boldsymbol{y}_1,\ldots,\boldsymbol{y}_t) \in \mathbb{N}^t$ such that*

$$A \in \bigcap_{i\in\{1,\ldots,t\}} \mathbb{CA}_{\lambda_i}(N;i,k,v) \cap \mathbb{PA}_{\boldsymbol{y}_i}(N;i,k,v) \neq \emptyset,$$

and let τ_i denote an arbitrary i-way interaction, then the following relations hold:

$$\rho_A(\tau_i) \geq N - (v^i-1)\boldsymbol{y}_i, \text{ for } 1 \leq i \leq t, \tag{2.24}$$
$$\rho_A(\tau_i) \leq N - (v^i-1)\lambda_i, \text{ for } 1 \leq i \leq t. \tag{2.25}$$

Proof: For a given i-way interaction τ_i, each of the (v^i-1) other i-way interactions in the same columns of A appears in at most $\boldsymbol{y}_i$ rows, thus τ_i appears in the remaining rows: $\rho_A(\tau_i) \geq N - (v^i-1)\boldsymbol{y}_i$. Analogously we get $\rho_A(\tau_i) \leq N - (v^i-1)\lambda_i$. □

In the following proposition we show two *Pascal-like* inequalities for balanced CAs, which are reminiscent of the Pascal-identity of binomial or q-binomial coefficients (see e.g. [204]), as well as of the Pascal-property of (block) intersection numbers (see [20, Chapter 2, Theorem 12], or [19, Construction 5.19]).

Proposition 2.5.13 *If $A \in \mathbb{CA}_{\lambda_{i+1}}(N;i+1,k,v) \cap \mathbb{PA}_{\boldsymbol{y}_i}(N;i,k,v) \neq \emptyset$, then also $A \in \mathbb{PA}_{\boldsymbol{y}_{i+1}}(N;i+1,k,v)$, for all*

$$\boldsymbol{y}_{i+1} \geq \boldsymbol{y}_i - (v-1)\lambda_{i+1}. \tag{2.26}$$

If $A \in \mathbb{CA}_{\lambda_i}(N;i,k,v) \cap \mathbb{PA}_{\boldsymbol{y}_{i+1}}(N;i+1,k,v) \neq \emptyset$, then also $A \in \mathbb{CA}_{\lambda_{i+1}}(N;i+1,k,v)$, for all

$$\lambda_{i+1} \leq \lambda_i - (v-1)\boldsymbol{y}_{i+1}. \tag{2.27}$$

Proof:

Let $\tau = \{(p_1,u_1),\ldots,(p_i,u_i),(p_{i+1},u)\}$ be an arbitrary $(i+1)$-way interaction, and let $\tau_i = \{(p_1,u_1),\ldots,(p_i,u_i)\}$. Then we have $\rho_A(\tau_i) = \sum_{u'=0}^{v-1} \rho_A(\tau_i \cup \{(p_{i+1},u')\})$ and thus

$$\rho_A(\tau) = \rho_A(\tau_i) - \sum_{\substack{u'=0 \\ u'\neq u}}^{v-1} \rho_A(\tau_i \cup \{(p_{i+1},u')\}) \tag{2.28}$$

If $A \in \mathbb{CA}_{\lambda_{i+1}}(N; i+1, k, v) \cap \mathbb{PA}^{\mathsf{y}_i}(N; i, k, v)$ we get from (2.28)

$$\rho_A(\tau) = \rho_A(\tau_i) - \sum_{\substack{u'=0 \\ u' \neq u}}^{v-1} \underbrace{\rho_A(\tau_i \cup \{(p_{i+1}, u')\})}_{\geq \lambda_{i+1}} \leq \mathsf{y}_i - (v-1)\lambda_{i+1}, \tag{2.29}$$

which shows that $A \in \mathbb{PA}^{(\mathsf{y}_i - (v-1)\lambda_{i+1})}(N; i+1, k, v)$.

If $A \in \mathbb{CA}_{\lambda_i}(N; i, k, v) \cap \mathbb{PA}^{\mathsf{y}_{i+1}}(N; i+1, k, v)$ we get from (2.28)

$$\rho_A(\tau) = \rho_A(\tau_i) - \sum_{\substack{u'=0 \\ u' \neq u}}^{v-1} \rho_A(\tau_i \cup \{(p_{i+1}, u')\}) \geq \lambda_i - (v-1)\mathsf{y}_{i+1},$$

which shows that $A \in \mathbb{CA}_{(\lambda_i - (v-1)\mathsf{y}_{i+1})}(N; i+1, k, v)$. □

The lower bound on the λ_i given by 2.22 can be strengthened as follows.

Lemma 2.5.14 *Let A be a $\mathsf{CA}_\lambda(N; t, k, v)$ and for all $1 \leq i < t$ let τ_i denote an arbitrary i-way interaction, then we have*

$$\rho_A(\tau_i) \geq \mathsf{CAN}_\lambda(t-i, k-i, v), \tag{2.30}$$

i.e. every $\mathsf{CA}_\lambda(N; t, k, v)$ is $\boldsymbol{\lambda}$-balanced for $\lambda_i = \mathsf{CAN}_\lambda(t-i, k-i, v)$ for all $1 \leq i < t$ and $\lambda_t = \lambda$.

Proof: Let $A = (a_1, \ldots, a_k) \in \{0, 1, \ldots, v-1\}^{N \times k}$ be a $\mathsf{CA}_\lambda(N; t, k, v)$, and $\{a_{p_1}, \ldots, a_{p_i}\}$ an arbitrary combination of i columns of A. Further, let $\tau_i = \{(p_1, u_1), \ldots, (p_i, u_i)\}$ denote an i-way interaction, which appears covered the least number of times in these columns, say $\rho_A(\tau_i) = \ell$. Now consider the subarray of A that is composed by these ℓ rows that cover τ_i and consider the projection of these rows onto the columns $\{a_1, \ldots, a_k\} \setminus \{a_{p_1}, \ldots, a_{p_i}\}$. In this array, denoted by A_{k-i}, each $(t-i)$-way interaction is covered at least Assume $\tau_{t-i} = \{(p_{i+1}, u_{i+1}), \ldots, (p_t, u_t)\}$ is an arbitrary $(t-i)$-way interaction. Since A is a $\mathsf{CA}_\lambda(N; t, k, v)$, we know that the t-way interaction $\tau_i \cup \tau_{t-i}$ must be covered at least λ times by the rows of A, more precisely, by the ℓ rows of A that cover τ_i. Thus τ_{t-i} must also be covered λ times by the rows of A_{k-i}. Thus A_{k-i} is a $\mathsf{CA}(\ell; t-i, k-i, v)$, and hence $\ell \geq \mathsf{CAN}(t-i, k-i, v)$. □

The above lemma shows, that enforcing $\boldsymbol{\lambda}$-balance with $\lambda_i \leq \mathsf{CAN}(t-i, k-i, v)$ does not remove any solutions from the search space. The analogous result can be shown for $\boldsymbol{\mathsf{y}}$-balance and packing arrays.

Lemma 2.5.15 *Let B be a $\mathsf{PA}_\mathsf{y}(N; t, k, v)$ and for all $1 \leq i < t$ let τ_i denote an arbitrary i-way interaction, then we have*

$$\rho_B(\tau_i) \leq \mathsf{PAN}_\mathsf{y}(t-i, k-i, v), \tag{2.31}$$

i.e. every $\mathsf{PA}_\mathsf{y}(N; t, k, v)$ is $\boldsymbol{\mathsf{y}}$-balanced for $\mathsf{y}_i = \mathsf{PAN}_\mathsf{y}(t-i, k-i, v)$ for all $1 \leq i < t$ and $\mathsf{y}_t = \mathsf{y}$.

Proof: Let $B = (b_1, \ldots, b_k) \in \{0, 1, \ldots, v-1\}^{N \times k}$ be a $\mathsf{PA}_{\boldsymbol{y}}(N; t, k, v)$, and $\{b_{p_1}, \ldots, b_{p_i}\}$ an arbitrary combination of i columns of B. Further, let $\tau_i = \{(p_1, u_1), \ldots, (p_i, u_i)\}$ denote an i-way interaction, which appears covered the maximal number of times in these columns, say $\rho_B(\tau_i) = m$. Now consider the subarray of B that is composed by these m rows that cover τ_i and the projection of these rows onto the columns $\{b_1, \ldots, b_k\} \setminus \{b_{p_1}, \ldots, b_{p_i}\}$. In this array, denoted by B_{k-i}, each $(t-i)$-way interaction is covered at most $\boldsymbol{y}$ times: Assume $\tau_{t-i} = \{(p_{i+1}, u_{i+1}), \ldots, (p_t, u_t)\}$ is an arbitrary $(t-i)$-way interaction. Since B is a $\mathsf{PA}_{\boldsymbol{y}}(N; t, k, v)$, we know that the t-way interaction $\tau_i \cup \tau_{t-i}$ is covered at most $\boldsymbol{y}$ times by the rows of B, more precisely, by the m rows of B that cover τ_i. Thus also τ_{t-i} appears covered at most $\boldsymbol{y}$ times by the rows of B_{k-i}. Thus B_{k-i} is a $\mathsf{PA}_{\boldsymbol{y}}(m; t-i, k-i, v)$, and hence $m \leq \mathsf{PAN}_{\boldsymbol{y}}(t-i, k-i, v)$. □

Corollary 2.5.16 *To summarize, from the above propositions we can conclude that for classification of non-equivalent $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs (with N rows and v symbols) for various $\boldsymbol{\lambda}, \boldsymbol{y}$ we can restrict attention to the following $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_t)$ and $\boldsymbol{y} = (y_1, \ldots, y_t)$ vectors without missing any interesting case:*

$$\lambda_i \leq \left\lfloor \frac{N}{v^i} \right\rfloor, \text{ for } 1 \leq i \leq t, \tag{2.32}$$

$$y_i \geq \left\lceil \frac{N}{v^i} \right\rceil, \text{ for } 1 \leq i \leq t, \tag{2.33}$$

$$\lambda_i \geq v \cdot \lambda_{i+1}, \text{ for } 1 \leq i < t, \tag{2.34}$$

$$y_i \leq v \cdot y_{i+1}, \text{ for } 1 \leq i < t, \tag{2.35}$$

$$y_{i+1} \leq y_i - (v-1)\lambda_{i+1}, \text{ for } 1 \leq i < t, \tag{2.36}$$

$$\lambda_{i+1} \geq \lambda_i - (v-1)y_{i+1}, \text{ for } 1 \leq i < t, \tag{2.37}$$

$$\lambda_i \geq N - (v^i - 1)y_i, \text{ for } 1 \leq i \leq t, \tag{2.38}$$

$$y_i \leq N - (v^i - 1)\lambda_i, \text{ for } 1 \leq i \leq t. \tag{2.39}$$

Further, for the classification of non-equivalent $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with k columns we can restrict attention to the following $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_t)$ and $\boldsymbol{y} = (y_1, \ldots, y_t)$ vectors:

$$\lambda_i \geq \mathsf{CAN}_{\lambda_t}(t-i, k-i, v), \tag{2.40}$$

$$y_i \leq \mathsf{PAN}_{y_t}(t-i, k-i, v). \tag{2.41}$$

A Combinatorial Construction for Balanced CAs

When we apply the **Product**-construction for CAs from [205] to balanced CAs we obtain the following assertion.

Proposition 2.5.17 *Let $A \in \mathbb{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N; t, k, v)$ and $B \in \mathbb{CA}^{\boldsymbol{y}'}_{\boldsymbol{\lambda}'}(M; t, k, w)$, with $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_t)$, $\boldsymbol{y} = (y_1, \ldots, y_t)$, $\boldsymbol{\lambda}' = (\lambda_1', \ldots, \lambda_t')$ and $\boldsymbol{y}' = ({y_1}', \ldots, {y_t}')$. Then for the product $A \otimes B$ of A and B we have*

$$A \otimes B \in \mathsf{CA}^{\boldsymbol{y} \cdot \boldsymbol{y}'}_{\boldsymbol{\lambda} \cdot \boldsymbol{\lambda}'}(NM; t, k, vw),$$

where $\boldsymbol{\lambda} \cdot \boldsymbol{\lambda}'$ *denotes component-wise multiplication, and* $A \otimes B := \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_M \end{pmatrix}$, C_ℓ *is a* $N \times k$ *array over the alphabet* $\{0, \ldots, v-1\} \times \{0, \ldots, w-1\}$ *and* $C_\ell(i,j) = (a_{i,j}, b_{\ell,j})$ *for* $\ell \in \{1, \ldots, M\}$*, that is matrix* C_ℓ *is constructed by taking the point-wise pairs of the elements of* A *with the elements in the* ℓ*-th row of* B*.*

Proof: Let $\tau = \{(p_1, (x_1, y_1)), \ldots, (p_i, (x_i, y_i))\}$ be an i-way interaction over $\{0, \ldots, v-1\} \times \{0, \ldots, w-1\}$, then $\rho_{A \otimes B}(\tau) = \rho_A(\{(p_1, x_1), \ldots, (p_i, x_i)\}) \cdot \rho_B(\{(p_1, y_1), \ldots, (p_i, y_i)\})$, hence $\lambda_i \lambda_i' \leq \rho_{A \otimes B}(\tau) \leq \boldsymbol{y}_i \boldsymbol{y}_i'$. □

Hypotheses Related to Balance and Optimal CAs

Like the SCEH heuristic used in [197] and as mentioned in Subsection 2.5.2, constraints imposed by $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance will remove valid solutions if the $\boldsymbol{\lambda}$ or $\boldsymbol{y}$ vector is too strong, i.e. too large or too small respectively. If we are interested to classify all non-equivalent balanced CAs of a certain size, we want to use $\boldsymbol{\lambda}$, $\boldsymbol{y}$ vectors that reduce the search space but do not remove any valid solutions from it. The bounds from Corollary 2.5.16 give a first indication for this *minimal* balance that is present in CAs. If the goal is to find a CA with a small number of rows N and use $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance as a pruning heuristic while maximizing the number of columns k, then we want to use strong $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors that do not remove all CAs. The selection of such $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors is difficult - to say the least, because little is known about the *maximal* balance that is present in (optimal) CAs. Below we give a brief overview of the known results and hypotheses found in the literature.

The special case of constructing optimal binary CAs of strength two is solved since the beginning of the 1970's and is given by Sperner-type theorems for the constructions of independent families of sets, given in [206] or [100]. From these results it is known that $\mathsf{CAN}(2, k, 2) = \min\{N : k \leq \binom{N-1}{\lceil \frac{N}{2} \rceil}\}$ and that an optimal $\mathsf{CA}(N; 2, k, 2)$ can be constructed by juxtaposing k vectors of constant weight $\lceil N/2 \rceil$ that have a leading 0 entry. In other words, we know that $\mathsf{CAK}^{(\lceil N/2 \rceil, \lceil N/2 \rceil - 1)}_{(\lfloor N/2 \rfloor, 1)}(N; 2, 2) = \binom{N-1}{\lceil N/2 \rceil}$, where the bound $\boldsymbol{y}_2 = \lceil N/2 \rceil - 1$ can be derived from inequality (2.36).

In [177] it was possible to generalize this result to binary CAs on Graphs . *Covering arrays on graphs* are generalizations of CAs of strength two, where all subarrays of two columns that correspond to adjacent vertices of a given graph have to cover all pairs over the alphabet. If the given graph is the complete graph we obtain the generic case of CAs. In [177, Theorem 5] was shown that if there exists a *binary CA on a graph* that has N rows, then there must also exist a *binary CA on a graph* with N rows which columns have constant weight $\lfloor N/2 \rfloor$ (or equivalently $\lceil N/2 \rceil$ by virtue of symbol permutations.) Viewing $(\boldsymbol{\lambda}, \boldsymbol{y})$-balance as a pruning method for (optimal) CA generation, the above result says that for the special case of binary CAs of strength two, the most restrictive

$\boldsymbol{\lambda}$-vector can be used for pruning, while being ensured that at least one solution remains in the reduced search space.

Similar results for the more general case of CAs such as higher strength or higher alphabets are absent and subject to ongoing investigations. For example, a conjecture to generalize Theorem 5 of [177] for higher alphabets was formulated for CAs on graphs in [177], and for CAs of strength two in [161]. To the best of my knowledge, the validity of this conjecture is still unknown.

Conjecture 2.5.18 (Conjecture 1 of [161]) *If there exists a* $\mathsf{CA}(N;2,k,v)$ *then there exists a CA with the same number of rows where every value occurs between* $\left\lfloor \frac{N}{v} \right\rfloor$ *and* $\left\lceil \frac{N}{v} \right\rceil$ *times in every column. In other terms:*

$$\mathbb{CA}(N;2,k,v) \neq \emptyset \;\Rightarrow\; \mathbb{CA}^{(\lceil N/v \rceil, N)}_{(\lfloor N/v \rfloor, 1)}(N;2,k,v) \neq \emptyset.$$

Note that in $\mathbb{CA}^{(\lceil N/v \rceil, N)}_{(\lfloor N/v \rfloor, 1)}(N;2,k,v)$ *we have used the trivial upper bound* N *on the occurrence of 2-way interactions. Alternatively we could have used* $\mathsf{y}_2 = \lceil N/v \rceil - v + 1$, *again derived from* (2.36).

A more general conjecture was formulated in [207] related to the *Sub-Combination Equalization Heuristic* reviewed in the introduction, see (2.16).

Conjecture 2.5.19 (SCEH Conjecture 5.6 of [207]) *Given that a* $\mathsf{CA}(N;t,k,v)$ *exists, then there also exists a* $\mathsf{CA}(N;t,k,v)$, *say* A, *where each* i*-way interaction appears almost the same number of times for all* $i = 1,\ldots,t$*:*

$$\forall i \in \{1,\ldots,t\}, \forall \tau, \tau' \in \mathbb{T}_{v,k,i} :\; |\rho_A(\tau) - \rho_A(\tau')| \leq 1.$$

Expressed in other terms: For $\boldsymbol{\lambda} = (\lfloor N/v^i \rfloor)_{i=1}^t$ *and* $\boldsymbol{y} = (\lfloor N/v^i \rfloor + 1)_{i=1}^t$

$$\mathbb{CA}(N;t,k,v) \neq \emptyset \;\Rightarrow\; \mathbb{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,v) \neq \emptyset.$$

This conjecture however does not hold, as can be seen by the counterexample given below and the classification results given in Subsection 2.5.5. Maybe the authors of [207] have identified themselves that this conjecture does not hold as it did not appear in this form in their follow-up work [197].

Example 2.5.20 *The only two non-equivalent CAs in* $\mathbb{CA}(15;3,12,2)$ *are:*

$$A_1 = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&1&1&0&0&1&1&1&0\\
0&0&1&1&0&1&1&1&0&1&1&0\\
0&1&0&1&0&0&0&1&1&0&1&0\\
0&1&1&0&1&1&0&1&0&0&0&1\\
0&1&1&1&1&0&1&0&1&1&0&1\\
1&0&0&1&1&1&1&1&1&0&0&1\\
1&0&1&0&0&0&0&1&1&1&0&0\\
1&0&1&1&1&0&0&0&0&0&1&1\\
1&1&0&0&1&0&1&1&0&1&1&0\\
1&1&0&1&0&1&0&0&0&1&0&1\\
1&1&1&0&0&1&1&0&1&0&1&1\\
1&1&1&1&1&1&1&0&1&0&0&0
\end{pmatrix}, \quad A_2 = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&1&1&1&0&0&0&1&1&1\\
0&0&1&0&1&1&0&1&1&0&0&1\\
0&0&1&1&0&1&1&0&1&0&1&0\\
0&0&1&1&1&0&1&1&0&1&0&0\\
0&1&0&0&1&1&1&0&1&1&0&0\\
0&1&1&1&0&0&0&1&0&0&1&1\\
1&0&0&0&1&1&1&0&1&1&0&0\\
1&0&1&1&0&0&0&1&0&0&1&1\\
1&1&0&0&1&1&1&1&0&0&1&0\\
1&1&0&1&0&1&0&1&1&1&0&0\\
1&1&0&1&1&0&1&0&1&0&0&1\\
1&1&1&0&0&1&1&0&0&1&0&1\\
1&1&1&0&1&0&0&0&1&1&1&0
\end{pmatrix},$$

where A_1 *is a* $\mathsf{CA}^{(8,5,4)}_{(7,3,1)}(15;3,12,2)$ *and* A_2 *is a* $\mathsf{CA}^{(8,6,4)}_{(7,2,1)}(15;3,12,2)$*. We can see that Conjecture 2.5.19 does not hold, since e.g. for the* 3*-way interactions in the first three columns of* A_1*:* $\tau = \{(1,0),(2,0),(3,0)\}$ *and* $\tau' = \{(1,0),(2,0),(3,1)\}$*, we have*

$$\rho_{A_1}(\tau) - \rho_{A_1}(\tau') = 4 - 1 = 3;$$

and in A_2 *consider e.g. the* 2*-way interactions in the first two columns* $\tau = \{(1,0),(2,0)\}$ *and* $\tau' = \{(1,0),(2,1)\}$*, where we have*

$$\rho_{A_2}(\tau) - \rho_{A_2}(\tau') = 6 - 2 = 4;$$

The classification results given in Table 2.6 show that A_1 *and* A_2 *are the only two non-equivalent CAs in* $\mathbb{CA}(15;3,12,2)$*.*

2.5.5 Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced Covering Arrays

In this subsection we present classification results for $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs. These results were obtained by means of a column extension algorithm (Algorithm 2 in [139]) that combines symmetry breaking with a SAT or PB constraint solver for computing the columns suitable for extension. For the details we kindly refer the interested reader to [139, Sec. 5]. In addition, the results of Subsection 2.5.4, in particular Corollary 2.5.16, can be used to aide the computational search.

We computed the numbers of non-equivalent CAs appearing in the intersections defining $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs:

$$\mathbb{CA}_{\boldsymbol{\lambda}}(N;t,k,v) \cap \mathbb{PA}_{\boldsymbol{y}}(N;t,k,v),$$

for some $t \in \{2,3,4,5,6\}$, $v \in \{2,3,4,5\}$, N up to our computational limits and all relevant $\boldsymbol{\lambda}$, $\boldsymbol{y}$ vectors as given by the bounds in Corollary 2.5.16. For these given parameters we compute all non-equivalent $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,v)$ for all $t \le k \le \mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)$ as our computational resources permit.

The number of considered parameter combinations N, t, k, v and the sometimes large number of relevant balance vectors for these combinations do not allow for a compact presentation that contains the detailed classification results. We hence provide the entirety of our results in form of an online appendix [208] in the columns headed by "**Full classification**". Of special interest are thereby those balanced CAs that achieve the *maximal number of columns* $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)$ for the respective parameters. We collect these results in separate tables which can also grow very large and are thus again provided online under [208] in the columns headed by "$\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ **tables**". Further, several relevant $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ tables are also provided in the Appendix 6.2, as far as their size allows their presentation.

For the creation of these mentioned $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ tables we fixed the number of rows N, the strength t and the alphabet size v. The different $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors appear in the header respectively in the first column of a table, see Table 2.6 for an example. For specific $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ vectors, the value of $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,v)$ appears as **bold** number in the respective cell of the table. The number of non-equivalent balanced CAs with CAK columns appears as superscript of a table entry. Whenever the result was obtained by computational search, we denote the required runtime next to it in the column headed "T", referring to our implementation of Algorithm 2 in [139] that uses clasp [209] as PB solver. The computations have been performed on a machine with an AMD EPYC 7502P processor with 32 cores at 2.5 GHz base clock and 3.35 GHz boost clock and 128GB of RAM. When a classification result is obtained by deduction from another result based on a theoretical result, we indicate this with a Greek letter in the "T" column. The Greek letters refer to the respective bounds of Corollary 2.5.16 that are violated. In these cases the classification result is the same as for the class of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs obtained when equality holds in the respective bound(s). Since no computational search is performed for these cases, there is also no time to be reported. Further, as these deduced results are of limited interest we have reduced their font size to improve readability of the tables. The entries labeled "t.o." indicate a timeout, i.e. that the exact classification for the specific instance went beyond our computational resources. Therefore, some table entries contain a "-" when the computational search for this specific class of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs is skipped as it contains a set of balanced CAs for which the classification already yielded a timed out. The key to read these tables is given by Table 2.5, including the mapping between bounds and Greek letters. As a representative sample of our computations, the results for parameters $N = 15$, $t = 3$, $v = 2$ are given in Table 2.6.

For the sake of a more compact representation, we condensed the information of these $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ tables and present some results by means of figures, which capture the essential information of the corresponding table. Due to the number of the results we thereby focus on what we believe are some interesting or characteristic instances. We split the presentation of our results in several subsections according to the alphabet size of the classified balanced CAs.

Table 2.5: Key to the $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ tables [208] and Table 2.6: abbreviations and their meaning.

Table entry	Meaning
ω	Computation not required based on (2.36)
κ	Computation not required based on (2.37)
ζ	Computation not required based on (2.38)
ψ	Computation not required based on (2.39)
t.o.	timeout, not finished within 3 600 seconds
-	Computation skipped due to previous timeout

$\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,k,2)$: Classification of binary balanced CAs

We separate the presentation of the results with respect to the strength of the binary balanced CAs.

$\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;3,k,2)$: Classification of Binary Balanced CAs of Strength $t=3$. We present the classification results of $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;3,k,2)$ for $N\in\{12,13,14,15,16\}$, by means of Figures 2.3 - 2.7, which capture the essential information of the results given in the Tables [210] – [214]. Note that, when the considered number of rows N is a covering array number for the given parameters, we indicate this with an asterisk as a superscript of that number in the caption of the figure. In these figures we depict each entry $\mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$ of the corresponding table for the *most restrictive vectors* $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ which yield this entry. In other words, we focus on the maximal $\boldsymbol{\lambda}$ vectors and the minimal $\boldsymbol{y}$ vectors that yield a certain number of non-equivalent balanced CAs with $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N,3,2)$ number of columns. Further, when $\boldsymbol{\lambda}\geq\boldsymbol{\lambda}'$ and $\boldsymbol{y}\leq\boldsymbol{y}'$, then the entry $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)=\mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$ appears *inside* of $\mathsf{CAK}^{\boldsymbol{y}'}_{\boldsymbol{\lambda}'}(N;t,v)=\mathsf{CAK}^{\boldsymbol{y}'^{\#}}_{\boldsymbol{\lambda}'}$. **We use these graphics to depict the increase in quantity (a higher number (#) of non-equivalent balanced CAs) or the increase in quality (a higher number of columns $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$) for weakened balance vectors.** If we replace an entry $\mathsf{CAK}^{\boldsymbol{y}'}_{\boldsymbol{\lambda}'}(N;t,v)$ in these figures with the set of all $(\boldsymbol{\lambda},\boldsymbol{y})$-balanced CAs with an arbitrary number of columns, then the depicted relations represent subset relations. From this point of view, the given $\mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$ values represent two properties of these sets, that is the maximal number of columns of arrays in this set and the number of non-equivalent arrays with that number of columns.

The case $N=12$ is very simple, there exists exactly one $\mathsf{CA}^{(6,3,2)}_{(6,3,1)}(12;3,11,2)$ up to equivalence for the strongest balance vectors. There do not exist balanced CAs with more columns, nor do there exist more balanced CAs with 11 columns for weaker balance vectors. The resulting Figure 2.3 and Table [210] appear in the simplest form.

The case $N=13$ (Figure 2.4 and Table [211]) carries already some structure. Table [211] shows that we get three different sets of non-equivalent balanced CAs: one containing a single $((6,3,1),(7,4,2))$-balanced CA with 5 columns, one containing a single $((6,2,1),(7,4,2))$-balanced CA with 8 columns, and one containing 8 non-equivalent $((6,2,1),(7,4,3))$-balanced CAs with 11 columns. Figure 2.4 summarizes the results

Table 2.6: $N = 15^{*}$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 15$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ *time*, where # represents the number of non-equivalent $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs and the time is given in seconds.

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(7,3,1)		(6,3,1)		(7,2,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(8,4,2)	$\mathbf{8}^{1}$	*0*	$\mathbf{8}^{1}$	ζ	$\mathbf{8}^{1}$	κ	$\mathbf{8}^{1}$	ζ,κ	$\mathbf{8}^{1}$	ζ,κ	$\mathbf{8}^{1}$	ζ,κ
(8,4,3)	$\mathbf{8}^{3}$	*0*	$\mathbf{8}^{3}$	ζ	$\mathbf{8}^{3}$	κ	$\mathbf{8}^{3}$	ζ,κ	$\mathbf{8}^{3}$	ζ,κ	$\mathbf{8}^{3}$	ζ,κ
(8,5,3)	$\mathbf{11}^{21}$	*16*	$\mathbf{11}^{21}$	ζ	$\mathbf{11}^{21}$	*22*	$\mathbf{11}^{21}$	ζ	$\mathbf{11}^{21}$	ζ	$\mathbf{11}^{21}$	ζ
(8,5,4)	$\mathbf{12}^{1}$	*67*	$\mathbf{12}^{1}$	ζ	$\mathbf{12}^{1}$	*80*	$\mathbf{12}^{1}$	ζ	$\mathbf{12}^{1}$	ζ	$\mathbf{12}^{1}$	ζ
(9,5,3)	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{22}$	*18*	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{22}$	*27*	$\mathbf{11}^{22}$	ζ	$\mathbf{11}^{22}$	ζ
(9,6,3)	$\mathbf{11}^{21}$	ψ,ω	$\mathbf{11}^{23}$	*19*	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{23}$	*24*	$\mathbf{11}^{23}$	ζ	$\mathbf{11}^{23}$	ζ
(9,5,4)	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{1}$	*168*	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{1}$	*199*	$\mathbf{12}^{1}$	ζ	$\mathbf{12}^{1}$	ζ
(9,6,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	*316*	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*391*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(9,6,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	*406*	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*498*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(8,6,3)	$\mathbf{11}^{21}$	ω	$\mathbf{11}^{21}$	ζ,ω	$\mathbf{11}^{21}$	*21*	$\mathbf{11}^{21}$	ζ	$\mathbf{11}^{21}$	ζ	$\mathbf{11}^{21}$	ζ
(8,6,4)	$\mathbf{12}^{1}$	ω	$\mathbf{12}^{1}$	ζ,ω	$\mathbf{12}^{2}$	*79*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(8,6,5)	$\mathbf{12}^{1}$	ω	$\mathbf{12}^{1}$	ζ,ω	$\mathbf{12}^{2}$	*70*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(9,7,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*338*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(9,7,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*438*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(9,7,6)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*430*	$\mathbf{12}^{2}$	ζ	$\mathbf{12}^{2}$	ζ
(10,5,3)	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{22}$	ψ	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{22}$	ψ	$\mathbf{11}^{22}$	*26*	$\mathbf{11}^{22}$	ζ
(10,6,3)	$\mathbf{11}^{21}$	ψ,ω	$\mathbf{11}^{23}$	ψ	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{23}$	ψ	$\mathbf{11}^{23}$	*25*	$\mathbf{11}^{23}$	ζ
(10,5,4)	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{1}$	*192*	$\mathbf{12}^{1}$	ζ
(10,6,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*391*	$\mathbf{12}^{2}$	ζ
(10,7,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*346*	$\mathbf{12}^{2}$	ζ
(10,8,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*342*	$\mathbf{12}^{2}$	ζ
(10,6,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*466*	$\mathbf{12}^{2}$	ζ
(10,7,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*483*	$\mathbf{12}^{2}$	ζ
(10,8,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*395*	$\mathbf{12}^{2}$	ζ
(10,7,6)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*475*	$\mathbf{12}^{2}$	ζ
(10,8,6)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*437*	$\mathbf{12}^{2}$	ζ
(10,8,7)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*443*	$\mathbf{12}^{2}$	ζ
(11,6,3)	$\mathbf{11}^{21}$	ψ,ω	$\mathbf{11}^{23}$	ψ	$\mathbf{11}^{21}$	ψ	$\mathbf{11}^{23}$	ψ	$\mathbf{11}^{23}$	ψ	$\mathbf{11}^{23}$	*22*
(11,6,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*326*
(11,7,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*347*
(11,8,4)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*348*
(11,6,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*468*
(11,7,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*464*
(11,8,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*492*
(11,9,5)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*392*
(11,7,6)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*477*
(11,8,6)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*496*
(11,9,6)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*398*
(11,8,7)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ	$\mathbf{12}^{2}$	*467*
(11,9,7)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*382*
(11,9,8)	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{1}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	ψ,ω	$\mathbf{12}^{2}$	*384*

of Table [211]. This case shows already, that in some cases when we ask for an array with *maximal balance* and a maximal number of columns, the answer is not necessarily definite. The $\mathsf{CA}^{(7,4,2)}_{(6,3,1)}(13;3,5,2)$ achieves maximum balance, but has only 5 columns. The $\mathsf{CA}^{(7,4,2)}_{(6,2,1)}(13;3,8,2)$ does not achieve the maximum balance of 2-way interactions, but has 8 columns, while a $\mathsf{CA}^{(7,4,3)}_{(6,3,1)}(13;3,11,2)$ does not achieve the maximum balance of 3-way interactions and has 11 columns.

The case $N = 14$ appears different, bearing a fully nested structure, see Figure 2.5 (and Table [212]), with a unique $((7,3,1),(7,4,2))$-balanced CA with 8 columns in the center and increasing quantities of non-equivalent balanced CAs with 11 columns for a chain of monotonically less restrictive $(\boldsymbol{\lambda},\boldsymbol{y})$-balance vectors.

The case $N = 15$ (Figure 2.6 and Table [213], respectively Table 2.6) reveals again a structure with two branches induced by two chains of less restrictive $(\boldsymbol{\lambda},\boldsymbol{y})$-balance vectors. Also, as mentioned in Example 2.5.20 the instance $N = 15$ documents a counter example to the SCEH Conjecture of [207]. As the results show there exists no $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(15;3,12,2)$ with $\boldsymbol{y}_i \in \{\lambda_i, \lambda_i + 1\}$. Furthermore, it is noteworthy that the instance $N = 15$ is a case where N is a CAN value, i.e. $\mathsf{CAN}(3,12,2) = 15$, and that $12 = \mathsf{CAK}(15;3,2)$ is the maximal number of columns such a CA with 15 rows can have. The $\mathsf{CA}(15;3,12,2)$ are so to say maximal in both dimensions. Nevertheless, the dissection of the set of binary CAs of strength $t = 3$ with $N = 15$ by means of $(\boldsymbol{\lambda},\boldsymbol{y})$-balance reveals a non-trivial and interesting structure.

The case $N = 16$ (Figure 2.7 and) obeys again a fully nested structure, with a unique $\mathsf{CA}^{(8,4,3)}_{(8,4,1)}(16;3,14,2)$ in the center. We note, that this is the second case apart from $N = 14$, where the maximum number of columns, in this case 14, is reached with the second strongest $\boldsymbol{\lambda},\boldsymbol{y}$ vectors. In the very center of the graphic appears the up to isomorphism unique $\mathsf{CA}^{(8,4,2)}_{(8,4,2)}(16;3,8,2)$, which constitutes an $\mathsf{OA}(16,8,2,3)$. Furthermore, we note that for this instance the consumed computation time was 63 hours.

Finally, for the case $N = 17$ we could not provide a full classification of balanced CAs $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(17;3,k,2)$, due to excessive run times. The computation for $((9,7,6),(8,2,1))$-balanced CAs finished after about $4.42 \cdot 10^7$ seconds, i.e., after about 512 days. The results show that there are three non-equivalent balanced CAs with 16 columns, i.e. three non-equivalent $\mathsf{CA}^{(9,7,6)}_{(8,2,1)}(17;3,16,2)$, and further that no such balanced CA with 17 columns could be found. This computation determines $\mathsf{CAK}^{(8,2,1)}_{(9,7,6)}(17;3,2) = 16$. These partial classification results can be found in Table 14 in the Appendix. In fact, the inequalities given in Corollary 2.5.16 allow to lift the balance constraints and to generalize this result leading to the following conclusions.

Corollary 2.5.21 ($\mathsf{CAK}(17;3,2) = 16$ **and** $\mathsf{CAN}(3,20,2) = 18$) *The maximum number of columns k for which a $\mathsf{CA}(17;3,k,2)$ exists is $k = 16$, i.e.,*

$$\mathsf{CAK}(17;3,2) = 16. \tag{2.42}$$

Further, this implies that the smallest number of rows N for which a $\mathsf{CA}(N;3,k,2)$ with $k \in \{17,18,19,20\}$ exists is $N = 18$, i.e.

$$\mathsf{CAN}(3,17,2) = \mathsf{CAN}(3,18,2) = \mathsf{CAN}(3,19,2) = \mathsf{CAN}(3,20,2) = 18. \tag{2.43}$$

Proof: From the computational result documented in Table 14 in the Appendix, we obtain $\mathsf{CAK}^{(9,7,6)}_{(8,2,1)}(17;3,2) = 16$. For determining $\mathsf{CAK}(17;3,2)$, when we determine $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(17;3,2)$ for the weakest balance vectors $\boldsymbol{\lambda}$, $\boldsymbol{y}$, i.e. such that do not impose any balance constraints on the arrays, then we have determined $\mathsf{CAK}(17;3,2)$. For example, from the basic inequality in (2.34) we get $\boldsymbol{\lambda} \geq (4,2,1)$ and in combination with (2.39) we get $\boldsymbol{y} \leq (13,11,10)$, thus $\mathsf{CAK}(17;3,2) = \mathsf{CAK}^{(13,11,10)}_{(4,2,1)}(17;3,2)$, as these balance vectors do not impose any effective balance constraints. The inequalities in Corollary 2.5.16 allow to further restrict the search to $\boldsymbol{\lambda} = (\lambda_1,\lambda_2,\lambda_3)$ and $\boldsymbol{y} = (y_1,y_2,y_3)$ that fulfil the given inequalities, that impose increased upper and lower bounds, while not imposing any effective constraints on the balance of the array. In particular when determining existence of a $\mathsf{CA}(3,17,2)$ inequality (2.40) implies that $\lambda_1 \geq \mathsf{CAN}(2,16,2) = 8$, where $\mathsf{CAN}(2,16,2) = 8$ is known due to [100] and [99], see also Theorem 2.2.11. Together with inequality (2.39) this yields further the bound $y_1 \leq N - (v^i - 1)\lambda_1 = 9$ for y_1, and with (2.36) we obtain the bounds for y_2 and for y_3: $y_2 \leq y_1 - (v-1)\lambda_2 = 7$, and $y_3 \leq y_2 - (v-1)\lambda_3 = 6$. So the balance vectors $\boldsymbol{\lambda} = (8,2,1)$ and $\boldsymbol{y} = (9,7,6)$) do not impose effective constraints on the balance of a $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(17;3,k,2)$ with $k \geq 17$ columns. This means that any $\mathsf{CA}(17;3,k,2)$ is also a $\mathsf{CA}^{(9,7,6)}_{(8,2,1)}(17;3,k,2)$, for $k \geq 17$, and further

$$\mathsf{CAK}(17;3,2) = k \Leftrightarrow \mathsf{CAK}^{(9,7,6)}_{(8,2,1)}(17;3,2) = k, \text{ when } k \geq 17. \tag{2.44}$$

From the computational search we get $\mathsf{CAK}^{(9,7,6)}_{(8,2,1)}(17;3,2) = 16$, and hence from the equivalence in (2.44) we know that $\mathsf{CAK}(17;3,2)$ cannot be 17 or larger. It follows that $\mathsf{CAK}(17;3,2) = 16$.

Due to $\mathsf{CAN}(t,k,v) = \min\{N : \mathsf{CAK}(N;t,v) \geq k\}$ (see also [70]), a further consequence of the above is $\mathsf{CAN}(3,17,2) = \min\{N : \mathsf{CAK}(N;3,2) \geq 17\} > 17$. Further, due to Chateauneuf and Kreher [205, Thm 4.6] it is known that $\mathsf{CAN}(3,20,2) \leq \mathsf{CAN}(3,10,2) + \mathsf{CAN}(2,10,2) = 18$, where $\mathsf{CAN}(2,10,2)$ follows again from [100] and [99] (see also Theorem 2.2.11), and $\mathsf{CAN}(3,10,2) = 12$ is known, e.g., due to Hnich et al. [69, Tab. 1], and can also be deduced from the results in Table [215] and Table [216] (Table 9 and Table 10 in the Appendix). Consequentially, we have $17 < \mathsf{CAN}(3,17,2) \leq \mathsf{CAN}(3,18,2) \leq \mathsf{CAN}(3,19,2) \leq \mathsf{CAN}(3,20,2) \leq 18$, which shows equation (2.43). □

$\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N,4,k,2)$: Classification of binary balanced CAs of strength $t = 4$. We shift attention to the classification of binary balanced CAs of strength $t = 4$ for the cases $N \in \{18,\ldots,25\}$, represented by Figures 2.8 - 2.15 and documented in Tables [217] – [224]. As before, when the considered number of rows N is a covering array number for

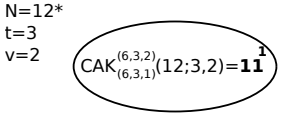


Figure 2.3: $N = 12^*$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 12$ rows, strength $t = 3$ and a binary alphabet, see also Table [210].

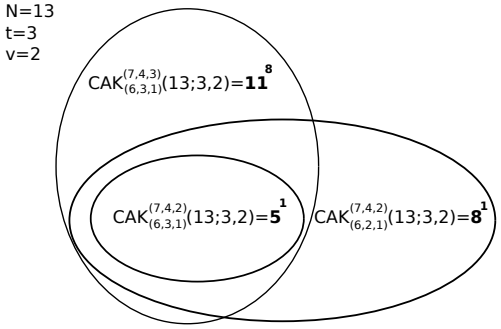


Figure 2.4: $N = 13$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 13$ rows, strength $t = 3$ and a binary alphabet, see also Table [211].

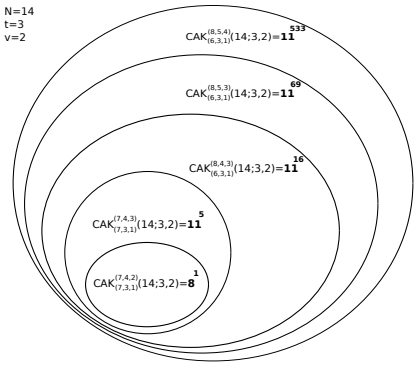


Figure 2.5: $N = 14$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 14$ rows, strength $t = 3$ and a binary alphabet, see also Table [212].

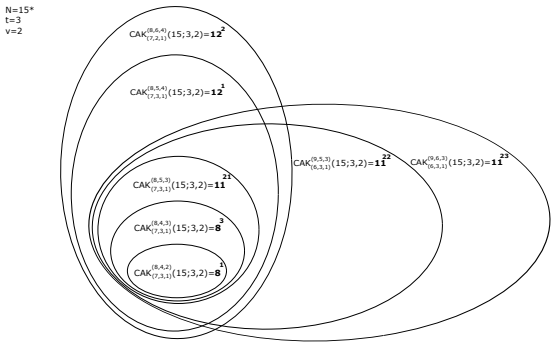


Figure 2.6: $N = 15^*$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 15$ rows, strength $t = 3$ and a binary alphabet, see also Table [213].

the given parameters, we indicate this with an asterisk as a superscript of that number in the caption of the figure.

Again, we present the essential information of our computations by means of figures condensing the information of the tables, but now we change the representation from a Venn diagram style to a lattice style, for reasons of compactness and readability. We want to mention that such lattice structures and representations do also appear in other, related works in the area. See for example [204] or [225]. As before, in the figures we depict each entry $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ of the corresponding table for the most restrictive vectors $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ which yield this entry. Now, when $\boldsymbol{\lambda} \geq \boldsymbol{\lambda}'$ and $\boldsymbol{y} \leq \boldsymbol{y}'$, then the entry $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}}(N;t,v) = \mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ appears *below* of $\mathsf{CAK}_{\boldsymbol{\lambda}'}^{\boldsymbol{y}'}(N;t,v) = \mathsf{CAK}_{\boldsymbol{\lambda}'}^{\boldsymbol{y}'^{\#}}$. **These graphics**

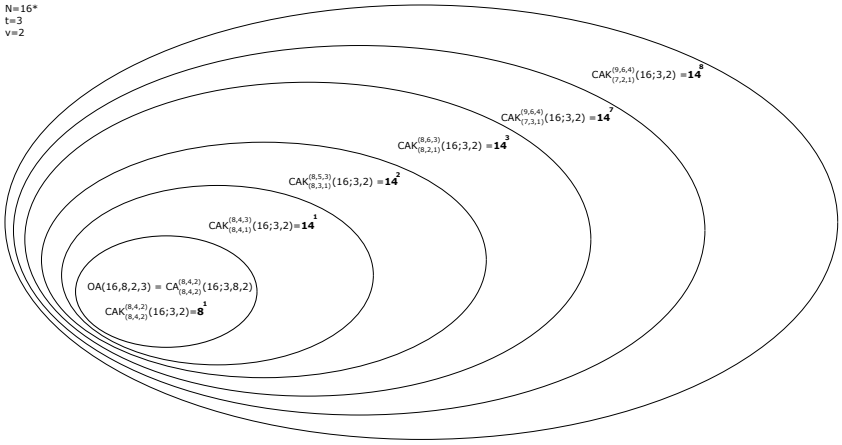


Figure 2.7: $N = 16^*$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 3$ and a binary alphabet, see also Table [214].

depict again the increase in quantity (a higher number (#) of non-equivalent balanced CAs) or the increase in quality (a higher number of columns $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$) for weakened balance vectors. In fact these figures can be read as lattices as we want to exhibit more clearly in the sequel.

Remark 2.5.22 *As mentioned in the beginning of Subsection 2.5.4 when $\boldsymbol{\lambda} \geq \boldsymbol{\lambda}'$ and $\boldsymbol{y} \leq \boldsymbol{y}'$ then $\mathbb{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N; t, k, v) \subseteq \mathbb{CA}^{\boldsymbol{y}'}_{\boldsymbol{\lambda}'}(N; t, k, v)$. Let us define $(\boldsymbol{\lambda}, \boldsymbol{y}) \preccurlyeq (\boldsymbol{\lambda}', \boldsymbol{y}')$ if and only if $\boldsymbol{\lambda} \geq \boldsymbol{\lambda}'$ and $\boldsymbol{y} \leq \boldsymbol{y}'$, for $\boldsymbol{\lambda}, \boldsymbol{\lambda}', \boldsymbol{y}, \boldsymbol{y}' \in \mathbb{N}^t$. Then the relation $\preccurlyeq$ defines a partial order on the set of pairs $(\boldsymbol{\lambda}, \boldsymbol{y})$ of balance vectors, where* smaller respectively $\preccurlyeq$ *indicates stronger balance vectors. In fact $(\mathbb{N}^t \times \mathbb{N}^t, \preccurlyeq)$ is the direct product of the partial orders $(\mathbb{N}^t, \geq)$ and $(\mathbb{N}^t, \leq)$. As the latter two represent lattices, also $(\mathbb{N}^t \times \mathbb{N}^t, \preccurlyeq)$ is a lattice. To clarify, given two pairs of balance vectors $(\boldsymbol{\lambda}, \boldsymbol{y})$ and $(\boldsymbol{\lambda}', \boldsymbol{y}')$, we make explicit their infimum and supremum. We consider the component wise minima and maxima of these vectors respectively:*

1. *$\boldsymbol{\lambda}_w$ where $\boldsymbol{\lambda}_w(i) := \min\{\boldsymbol{\lambda}(i), \boldsymbol{\lambda}'(i)\}$, and $\boldsymbol{y}_w$ where $\boldsymbol{y}_w(i) := \max\{\boldsymbol{y}(i), \boldsymbol{y}'(i)\}$, then we have:*

$$(\boldsymbol{\lambda}, \boldsymbol{y}) \vee (\boldsymbol{\lambda}', \boldsymbol{y}') = (\boldsymbol{\lambda}_w, \boldsymbol{y}_w). \tag{2.45}$$

2. *$\boldsymbol{\lambda}_m$ where $\boldsymbol{\lambda}_m(i) := \max\{\boldsymbol{\lambda}(i), \boldsymbol{\lambda}'(i)\}$, and $\boldsymbol{y}_m$ where $\boldsymbol{y}_m(i) := \min\{\boldsymbol{y}(i), \boldsymbol{y}'(i)\}$, then we have:*

$$(\boldsymbol{\lambda}, \boldsymbol{y}) \wedge (\boldsymbol{\lambda}', \boldsymbol{y}') := (\boldsymbol{\lambda}_m, \boldsymbol{y}_m). \tag{2.46}$$

In the Hasse diagram of this lattice, instead of the elements $(\boldsymbol{\lambda}, \boldsymbol{y})$ we can now enter the information $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N; t, v) = \mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$. However, for most pairs of $(\boldsymbol{\lambda}, \boldsymbol{y}) \in \mathbb{N}^t \times \mathbb{N}^t$ this information will be duplicated or redundant. For too strong balance vectors there

do not exist any balanced CAs (we enter $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)=0^0$ *to represent this) and at a certain point, when balance vectors are weakened sufficiently enough, then there are no new balanced CAs found at all. Again, as for the tables, Corollary 2.5.16 gives some bounds that tell us which* $(\boldsymbol{\lambda},\boldsymbol{y})$ *pairs we have to consider to cover all the relevant information. Figures 2.8 - 2.15 capture the essential information of the belonging Tables [217] - [224]. That means at the bottom of a diagram we mostly depict the information* $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)=\mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$ *for strongest* $(\boldsymbol{\lambda},\boldsymbol{y})$ *vectors for which* $(\boldsymbol{\lambda},\boldsymbol{y})$*-balanced CAs exist. In some cases we also depict a* $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)=0^0$ *entry when we want to emphasize that there do not exist such* $(\boldsymbol{\lambda},\boldsymbol{y})$*-balanced CAs. At the top of a diagram we see the information* $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)=\mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$ *for the weakest balance vectors that yield any new balanced CAs, not found for other balance vectors. In other words, any further weakening of* $\boldsymbol{\lambda}$ *or* $\boldsymbol{y}$ *does not yield any more (balanced) CAs. Nevertheless, in some cases when there are two or more such pairs of balance vectors, we add with* dotted lines *an additional top element to recover a sub-lattice structure, see Figure 2.15. Notably, in none of the documented cases such a closure of the diagrams was required at the bottom of the diagram by means of* $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)=0^0$ *entries, i.e. there are always unique strongest balance vectors for which* $(\boldsymbol{\lambda},\boldsymbol{y})$*-balanced CAs exist.*

For $N=18$ and $N=19$ (Figures 2.8 and 2.9) we can see the typical nested structure as a result of a chain of less restrictive $(\boldsymbol{\lambda},\boldsymbol{y})$-balance vectors. Since only the quantity of balanced CAs is increasing for less restrictive $(\boldsymbol{\lambda},\boldsymbol{y})$-balance vectors and the number of columns is constant, in these cases, the relations of the partial orders represented by the figures can be read as subset relations.

The case $N=20$ (Figure 2.10) then again exhibits a comparatively rich structure with several nested substructures, in fact again nested sets of arrays, since the CAK values are constant 5. Ultimately, all nested sets are contained in the set of $((8,4,2,1),(12,8,6,5))$-balanced CAs. Notably, the cases $N=19$ and $N=20$ are the very first in our documentation where there does not exist a balanced CA for the strongest balance vectors for any number of columns, i.e. there exists no $\mathsf{CA}^{(10,5,3,2)}_{(9,4,2,1)}(19;4,k,2)$ and no $\mathsf{CA}^{(10,5,3,2)}_{(10,5,2,1)}(20;4,k,2)$ for any k.

For $N=21$ (Figure 2.11) we obtain again the classification consisting of a unique array of highest balance, and no other arrays for weaker balance. For $N=22$ (Figure 2.12 and Table [221]) and $N=23$ (Figure 2.13 and Table [222]) we see again an increasing complexity of embedded (subsets of) balanced CAs, while for $N=24$ the sets of balanced CAs collapses again to the simplest case (Figure 2.14). This increase and decrease in the diversity of balanced CAs could be connected to the CAN numbers. While all values in the tables (and in the figures) represent $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)$ values, i.e. the maximal achievable number of columns for the respective parameters, when we consider the smallest N for which the $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)$ number of columns can be achieved, this certainly represents a special case. For example, when we consider the case $N=25$ (Figure 2.15) we can see again an increase in the diversity of balanced CAs. An array with CAN rows and $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N;t,v)$ columns may allow less variability in the occurrence of i-way interactions.

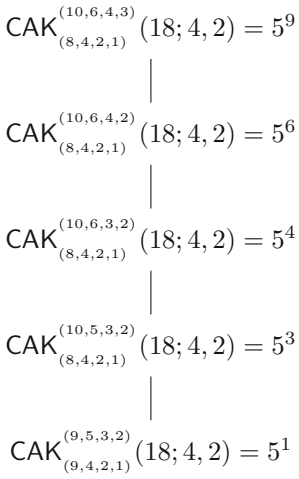


Figure 2.8: $N = 18$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 18$ rows, strength $t = 4$ and a binary alphabet, see also Table [217].

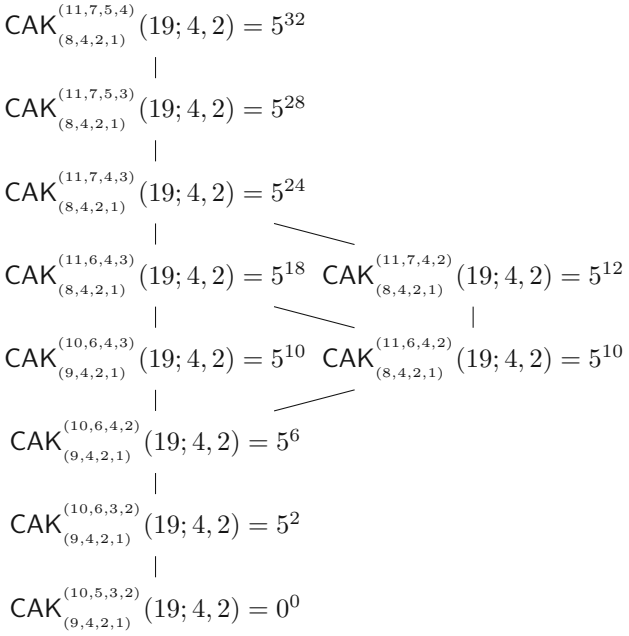


Figure 2.9: $N = 19$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 19$ rows, strength $t = 4$ and a binary alphabet, see also Table [218].

$\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(52, 5, k, 2)$: A Case of Binary Balanced CAs of Strength $t = 5$

We want to highlight the classification results for $\mathbb{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(52; 5, k, 2)$, which are given in Table [188] and are summarized in Figure 2.16. For the creation of this table we have removed additional $\boldsymbol{\lambda}$ vectors based on the bound in (2.40), which allows us for $k \geq 8$ to consider only vectors $\boldsymbol{\lambda} \succcurlyeq (24, 12, 6, 2, 1)$. From Lemma 2.5.14 we know that for weaker $\boldsymbol{\lambda}$ vectors, there will be no new balanced CAs found with $k \geq 8$: each 3-way interaction τ_3 must be covered at least 6 times, $\rho_A(\tau_3) \geq \mathsf{CAN}(5-3, 8-3, 2) = \mathsf{CAN}(2, 5, 2) = 6$, and hence we can restrict to $\boldsymbol{\lambda}$ vectors with $\lambda_3 \geq 6$. Then $\lambda_2 \geq 2 \cdot 6$ and $\lambda_1 \geq 2 \cdot 12$ follows from (2.35).

Besides this, we believe that the case of $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(52, 5, k, 2)$ is interesting because $52 = \mathsf{CAN}(5, 8, 2)$ is a CAN number, see e.g. [226] for verification. Nevertheless, we need to weaken $\boldsymbol{\lambda}$ to $(26, 12, 6, 2, 1)$ (paired with $\boldsymbol{y} = (26, 14, 8, 4, 3)$) in order to find the first optimal CA with $8 = \mathsf{CAK}(52; 5, 2)$ columns, see Figure 2.16. In other words, we have to consider the second weakest $\boldsymbol{\lambda}$ vector in order to find one of the optimal $\mathsf{CA}(52; 5, 8, 2)$'s. This observation clearly contradicts the intuitive speculation, that optimal CAs obey strong balance properties.

One may speculate that the non-existence of a $\mathsf{CA}^{\boldsymbol{y}}_{(26,13,6,2,1)}(52, 5, 8, 2)$ can be explained

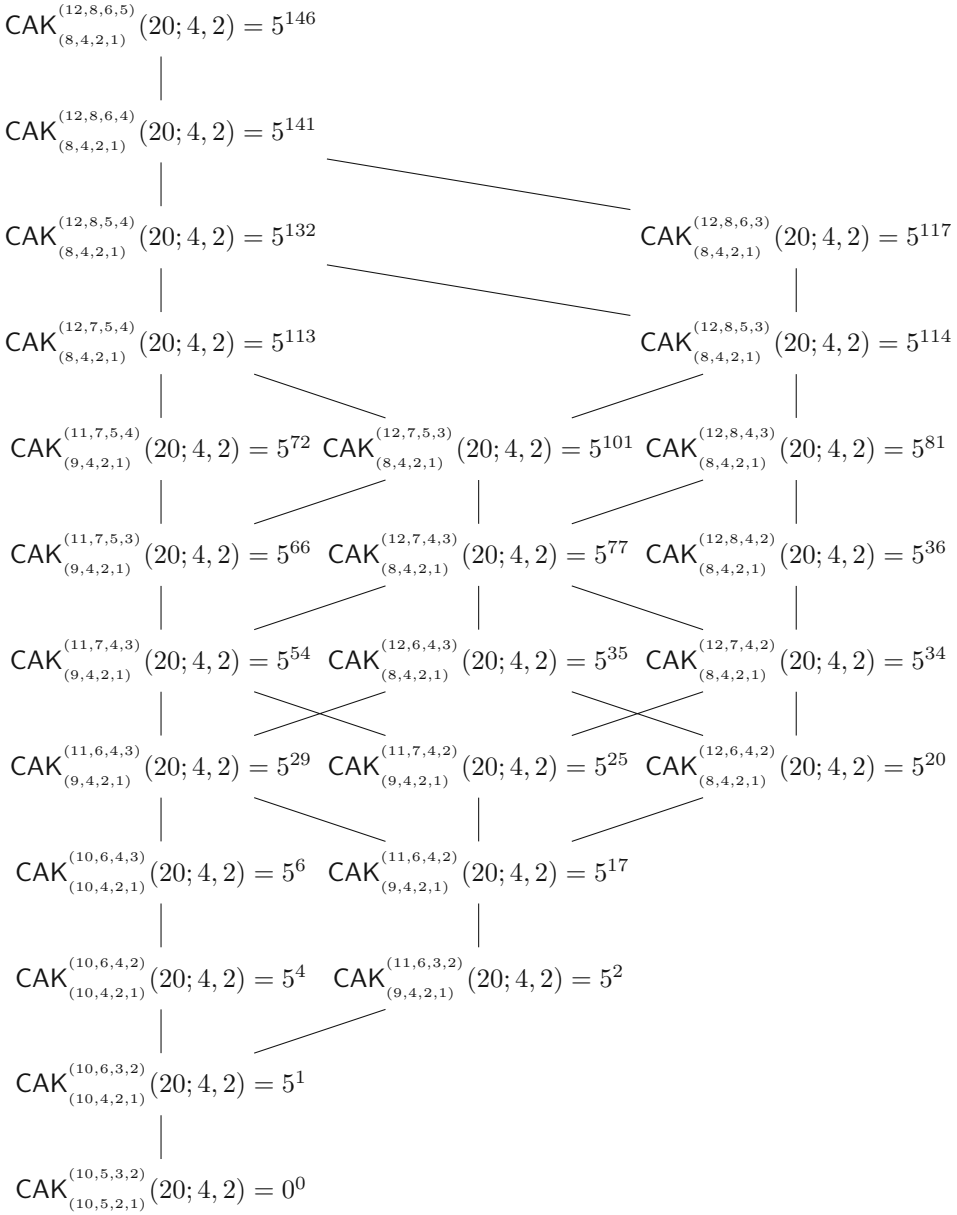


Figure 2.10: $N = 20$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 20$ rows, strength $t = 4$ and a binary alphabet, see also Table [219].

$$\mathsf{CAK}^{(11,6,3,2)}_{(10,5,2,1)}(21;4,2) = 6^{1}$$

Figure 2.11: $N = 21^{*}$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 21$ rows, strength $t = 4$ and a binary alphabet, see also Table [220].

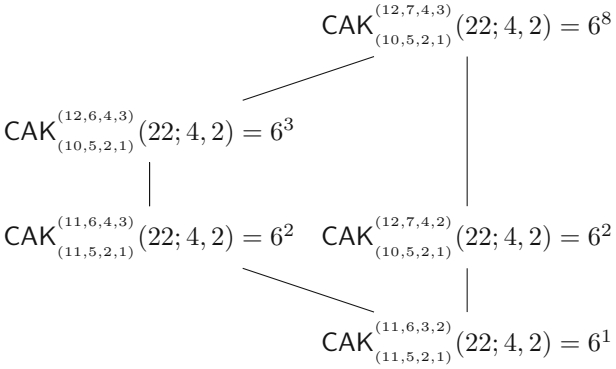


Figure 2.12: $N = 22$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 22$ rows, strength $t = 4$ and a binary alphabet, see also Table [221].

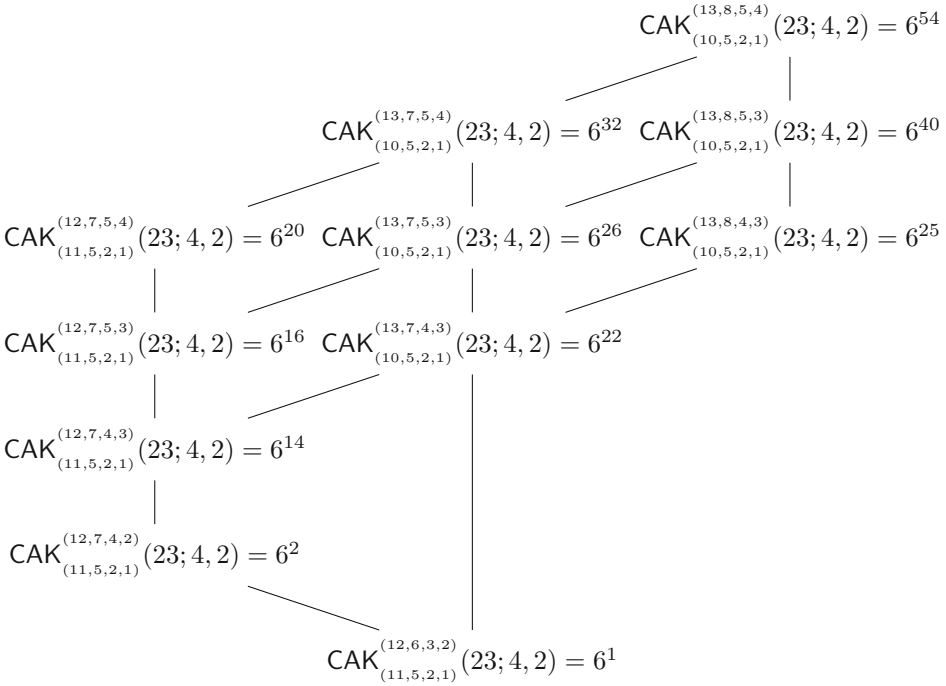


Figure 2.13: $N = 23$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 23$ rows, strength $t = 4$ and a binary alphabet, see also Table [222].

$$\mathsf{CAK}^{(12,6,3,2)}_{(12,6,3,1)}(24;4,2) = 12^1$$

Figure 2.14: $N = 24^*$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 24$ rows, strength $t = 4$ and a binary alphabet, see also Table [223].

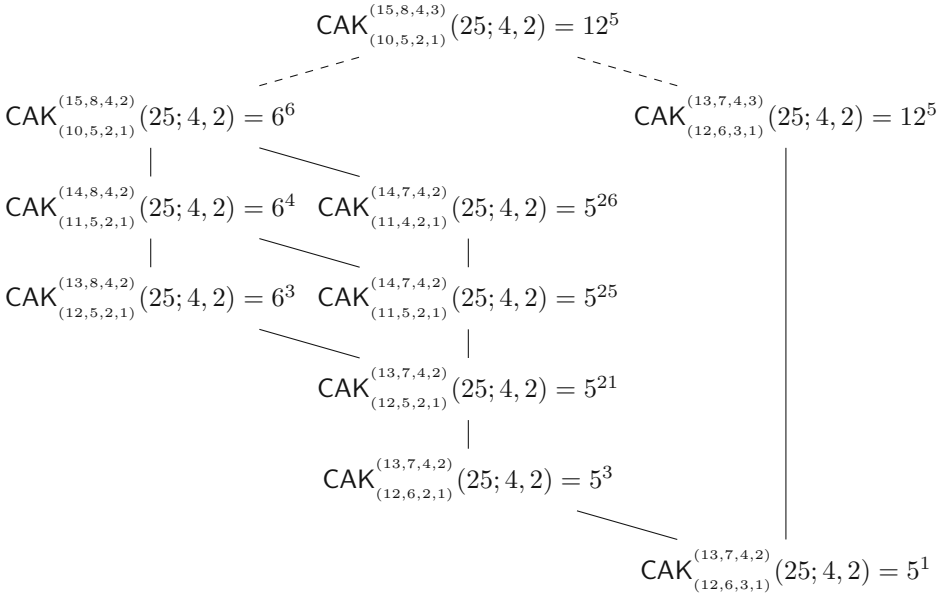


Figure 2.15: $N = 25$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 25$ rows, strength $t = 4$ and a binary alphabet, see also Table [224].

by the non-existence of some Orthogonal Array of lower strength that such a balanced CA would constitute. A $\mathsf{CA}^{\boldsymbol{y}}_{(26,13,6,2,1)}(52, 5, 8, 2)$ would constitute an $\mathsf{OA}(52, 8, 2, 2)$ of index 13, but the latter does exist and cannot explain the non-existence of the former: We even know that an $\mathsf{OA}(52, 51, 2, 2)$ of Paley type exists – from which any 8 columns constitute an $\mathsf{OA}(52, 8, 2, 2)$. For the construction of the $\mathsf{OA}(52, 51, 2, 2)$, the interested reader is referred to [24, Chapter 7], in particular Theorem 7.5 for the construction of Orthogonal Arrays from Hadamard matrices and Table 7.29 for the construction of a Hadamard matrix of order 52. We believe that this observation indicates that balanced CAs are notable combinatorial objects and deserve to be investigated in their own right.

$\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N, t, k, 3)$: Classification of Ternary Balanced CAs

For ternary balanced CAs we cannot provide as many classification results as for binary balanced CAs, due to the significantly increased runtime of our classification algorithm caused by the larger alphabet. For strength $t = 2$ and $t = 3$ all results can be found at [227]. The overall picture created by the results for ternary balanced CAs is similar as for binary balanced CAs. Again as an overall theme, we can see a reduced diversity in the sets of balanced CAs when the number of rows is a CAN value. We comment on this observation also in the next subsection.

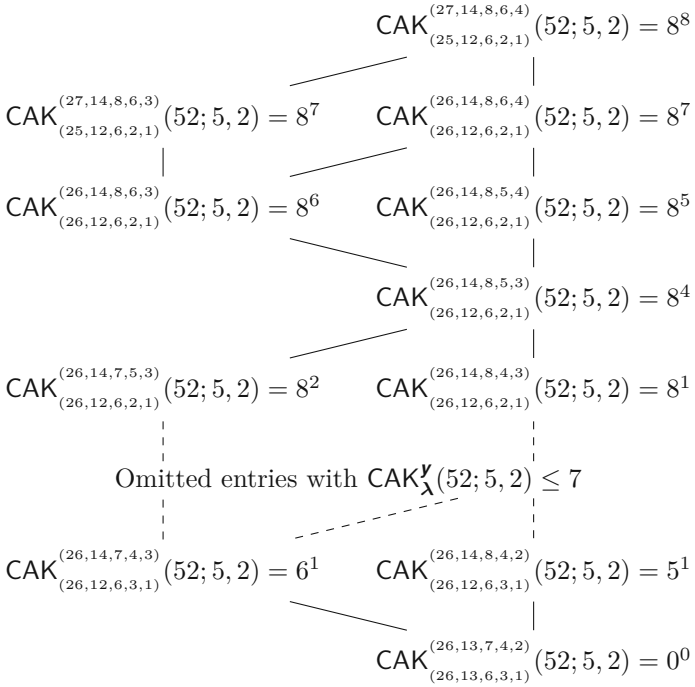


Figure 2.16: $N = 52^*$, $t = 5$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{\gamma})$-balanced CAs with $N = 52$ rows, strength $t = 5$ and a binary alphabet, see also Table [188].

Connections and Distinction to Existing Classification Results in the Literature

Several existing works in the literature pertain to the classification of CAs, their generalizations or specializations. We briefly discuss the most relevant below.

Colbourn et al. consider *radius covering arrays* in [159]. A radius covering array of strength t with radius r has the property that in each subarray comprised of t columns for each t-way interaction there exists at least one row that differs in at most r positions from the t-way interaction. For $r = 0$ we obtain the case of CAs. In [159] explicit constructions for radius covering arrays are developed and bounds on the size are derived. In some cases the minimal number of rows is determined by computation and a full classification is performed. In terms of classification, the connection to the presented work is the following: classification of CAs appears as a special case in both works, for $r = 0$ in [159] and for the weakest balance vectors in our work. Hence, the second column ($r = 0$) of Table 7 in [159] contains classification results of CAs, which appear also in our tables as the entries for the weakest balance vectors, which do not pose any balance constraints, found in the bottom right entry of the respective table. It is worth to mention that tables in [159] present the $\mathsf{CAN}(t, k, v)$ for different k, while results presented in this paper present $\mathsf{CAK}(N; t, v)$ for different N. Further, we reproduced the results of Tables

4 and 5 of [159], however only the last two entries of each column are presented in our Tables [216], [228], [229], [230] and [231] respectively. We extend partly the classification of binary CAs in Table 7 of [159] by finding the unique $\mathsf{CA}(8;2,35,2)$ (see [232]), we determined that there are exactly two non-equivalent $\mathsf{CA}(15;3,12,2)$ (see Figure 2.6 or Table [213]) and that there are exactly eight non-equivalent $\mathsf{CA}(16;3,14,2)$ (see Figure 2.7 or Table [214]). Further, for ternary CAs we determined that there are exactly four non-equivalent $\mathsf{CA}(13;2,9,3)$, a unique $\mathsf{CA}(27,3,4,3)$ and a unique $\mathsf{CA}(33;3,6,3)$.

Izquierdo-Marquez and Torres-Jimenez perform in [189] CA classification with a column extension algorithm similar to Algorithm 2 in [139]. Despite our algorithm being faster in all instances that we computed, we were not able to reproduce the same quantity of classification results. We believe, that this is mainly due to smaller computational resources – primarily time. However, all classification results we obtained agree with those reported in [189]. Note again, that [189] presents the $\mathsf{CAN}(t,k,v)$ for different k, while we present $\mathsf{CAK}(N;t,v)$ for different N.

Kokkala et al. classify CAs and *uniform* CAs (UCAs) of strength $t=2$ and alphabet sizes $v\in\{3,4,5,6\}$ in [161]. Clarifying the connection between their work and ours, in [161] the number of non-equivalent CAs for $t=2$ and given N,k,v is given in the fifth column of Table 1. Again, in the $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ tables in the online appendix [208] the number of non-equivalent CAs with a maximal number of columns k for given N,t,v can be found in the exponent of the bottom right entries of the tables. These entries are identical to the ones above the "0" entries in the fifth column of Table 1 in [161]. More interestingly, the number of non-equivalent uniform CAs, given in the sixth column of Table 1 in [161], can be found in the respective table for N,t,v in our tables, when considering the weakest $\boldsymbol{\lambda}$, $\boldsymbol{y}$ vectors that have $\lambda_1=\lfloor N/v\rfloor$ and $y_1=\lceil N/v\rceil$, i.e. the right most column for $\boldsymbol{\lambda}$ with $\lambda_1=\lfloor N/v\rfloor$ and the lowest row for $\boldsymbol{y}$ with $y_1=\lceil N/v\rceil$. However, we did not manage to reproduce all results of [161], which is at least partly explained due to less computational resources and different algorithmic design. For the cases (*uniform* CAs and CAs) where our classifications are overlapping we obtained the same results as [161] in all but one case: we have found exactly 2495 non-equivalent $\mathsf{UCA}(19;2,5,4)$ instead of the reported 1495 in [161]. Based on the equality of all other common classification results, we suppose that a typo is the reason for the difference in numbers.

In the last paragraph of [161] the authors note that the definition of *uniform* CAs applies to CAs of any strength (not only the considered $t=2$) and raise the question "*if it is always possible to find an optimal covering array, of any strength, that is also uniform*", in other words a generalization of Conjecture 2.5.18 of [161] to higher strength. In our view, the generalization of *uniformity* is not necessarily unique. It could be understood as uniformity in single columns in higher strength CAs, simply replacing "$t=2$" with a $t>2$ in Conjecture 2.5.18; but it could also be understood as uniformity in the appearance of $(t-1)$-way interactions of higher strength CAs, i.e. replacing "value" with "$(t-1)$-way interaction" and "N/v" with "N/v^{t-1}" in Conjecture 2.5.18. Finally, it could be understood as uniformity in the appearance of i-way interactions for all $1\leq i\leq t-1$ in higher strength CAs. This last version leads exactly to the notion of

$(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs when $\boldsymbol{\lambda}$ is maximal and $\boldsymbol{y}$ is minimal on the first $t-1$ entries. For the latter two interpretations, our experiments show that it is not always possible to find such CAs with the same number of columns and rows. Table [211] shows that there exist $\mathsf{CA}(13; 3, 11, 2)$ but for the strongest balance vectors we can only find a CA with five columns $\mathsf{CA}^{(7,4,2)}_{(6,3,1)}(13; 3, 5, 2)$. Further, Table [214] ($N = 16$) documents such a case where N is a CAN number. That uniformity of $(t-1)$-way interactions cannot always be achieved is also documented by the instance $N = 15$, $t = 3$, $v = 2$ in Table [213], since there are at most 8 columns in a $((6, 3, 1), (8, 4, 3))$-balanced CA, i.e. $\mathsf{CA}^{(8,4,3)}_{(6,3,1)}(15; 3, 8, 2)$, while there exist $\mathsf{CA}(15; 3, 12, 2)$, but these are $((7, 3, 1), (8, 5, 4))$-balanced at best. Likely the authors of [161] anyhow meant the first generalization of uniformity described above, i.e uniformity in single columns for higher strength CAs. For this case we want to highlight that in our experiments we always found an array with $\mathsf{CAK}(N; t, v)$ number of columns that was $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced where $\lambda_1 = \lfloor N/v \rfloor$ and $\boldsymbol{y}_1 = \lceil N/v \rceil$, i.e. we cannot falsify Conjecture 2.5.18, also not when considering its generalization for $t > 2$.

To conclude this discussion on connections to related work, we want to mention that the authors of [161] do hypothesize that nonuniform CAs "*will be more abundant when k is not maximal for a given N*". We also noted the same phenomenon in our experiments, as the diversity in the sets of balanced CAs is reduced when the number of rows is a CAN value. We believe that the increased diversity of balanced CAs with non-CAN number of rows may be explained by additional degrees of freedom due to the existence of rows that are not necessary to provide the required coverage of t-way interactions. However, there exist cases of CAN (and CAK arrays) where the structure of the classification results is non trivial, where the number of columns increases for weaker balance vectors, i.e. no CAK arrays are found for the strongest balance vectors, or the number of non-equivalent arrays increases. This is even true when the CAK number of columns is attained as the case $N = 16$ in Figure 2.7 demonstrates.

The classification results that also take into account the balance properties of CAs show once more the rich structure that of this class of as objects of combinatorial design theory. With this we conclude for now the theoretical considerations of covering arrays, the structures underlying combinational combinatorial testing and turn to theoretical aspects of a notion used for sequential combinatorial testing.

2.6 Automata Theoretic Models for Sequential Combinatorial Testing

Formal languages are well-established formalisms to model the behavior of event-driven or stimuli-driven (software) systems, which are typically represented by finite automata [233]. In this section we consider a problem arising in sequential combinatorial test set generation existing in the literature and treat it with methods of automata theory and formal languages. We start by providing some context to sequential CT, and by outlining the motivation for the work presented in this section. Then we mention some

related works from automata theory and formal languages. In Subsection 2.6.1 we provide some preliminaries with regards to automata theory. Thereafter we discussion the connection between the arising problem of sequential CT and existing notions of products of automata theory, where we establish a connection to the *infiltration product*. We then define the *reduced shuffle product* for words over a finite alphabet Σ, and for formal power series in $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$. We show that it is commutative, and further show properties that put the new product in relation to existing ones. These properties induce the new product's applicability to the problem arising in sequential CT. Finally we give a construction of a product automaton that can be understood as a *reduced shuffle automaton* for two given words. Before we come to the technicalities, we provide some context on sequential CT and the motivation for the subsequent work, viewing sequential combinatorial test set generation through the lens of automata theory. For this we also provide some preliminaries on automata theory.

Context. Pivotal early works on sequential CT are represented for example by [234] and [75]. The key idea of sequential CT, is to test event-driven (software) systems with a set of test sequences - each element of a sequence represents an input-event to the SUT. The defining characteristic of the set of test sequences is thereby that any sequence of t events, that *may* appear as a subsequence of an input sequence, *does so in at least one* of the test sequences, see also [122, Def. 8]. This concept is also referred to as *t-way sequence coverage* in the combinatorial testing literature, see for example [122], and applied in order to achieve a certain degree of coverage of the input space of the SUT. In terms of automata and formal languages the same concept is known as *piecewise subword*, see for example [235]. There exist slightly deviating notions of sequential combinatorial test sets, as depending on the use case different properties of them are desired. Notions range from more constrained ones, i.e. SCAs, containing sequences of fixed length where each event has to appear exactly once, see e.g. [63], to such where the sequences can be of arbitrary length and repetitions of events can appear, as in [123]. There are different approaches to the generation of sequential combinatorial test sets. For example, the authors of [122] use incremental extension algorithms based on a strongly connected component decomposition of the graph underlying the finite automaton of the SUT. In [157] a conditional expectation algorithm is proposed for the generation of SCAs. Further, also meta-heuristic approaches have been followed, e.g., simulated annealing in [236]. However, despite the plurality of notions, and manifold approaches to generation, there is one characteristic communality: the aim to find a *small* test set (i.e. of low cardinality), containing *short* sequences (or words) while the desired *coverage criteria* (pertaining the appearance of sub-words of a fixed length) are satisfied.

Motivation. Bombarda and Gargantini propose to use an automata theoretic approach to this problem in [123], in particular to the generation of sequential combinatorial test sets for event-driven software systems – where repetitions of events and different test sequence lengths are allowed. They consider so-called *t-wise automata* which accept all words that contain a given word of length t as subword, we will introduce them formally later in Section 2.6.2. In order to derive executable tests for a system under test Bombarda

and Gargantini [123] propose to intersect (or multiply) multiple t-wise automata with an automaton modeling the SUT. The resulting automaton is then used to derive a test sequence that *covers* the t-way sequences, i.e. that contains the corresponding words of length t as a subsequence. Once more we point out that the goal is thereby to compute a test sequence that is as short as possible. However, the Hadamard product of automata occurring in this approach can grow rapidly very large and represents a bottleneck of the approach, as is also recognized by the authors of [123]. We kindly refer the interested reader to [123] for more details. A potential minimization of the appearing interim product automata would likely be time consuming as well: it is known that in general the computation of minimal automata is a hard problem [237], [238].

Inspired by [123], the motivation for the work presented in this section is to propose a product operator that allows to consider the product of two automata, each accepting exclusively a word w_1 respectively w_2, instead of the Hadamard product of the t-wise automata corresponding to w_1 and w_2. The resulting accepted language would contain less words, while w_1 and w_2 still would appear as subwords of any accepted word. Also we could still find the shortest word with this property in the resulting language. We will introduce such a product operator and investigate its properties, where we also outline its applicability for combinatorial t-way sequence generation, by intersecting t-wise automata. As we are primarily interested in finding an automata theoretic *model* for this product construction appearing in sequential CT, we focus exclusively on its formal properties and only discuss also its realization for the generation of t-way sequence test sets.

Related Work to Language Operators and Automata Theory. The study of language operators has a long history in the formal languages and automata theory. We briefly mention some related works below. Pin and Sakarovitch study operators that preserve rationality in [239], where amongst others, also the shuffle and infiltration product are studied. The primary aim of their paper is to show that many known constructions applied to languages can be represented as a particular case of a "*general and simple approach*", that is representing the language operation as the inverse of a transduction, if possible, and to construct a monoid accepting the resulting language, based on monoids accepting the original languages, and a matrix representation of the transduction. In this section we do not apply the method proposed in [239], instead we follow an elementary approach to establish our results.

The authors of [240] present a comparative study, dedicated to some products of automata. Using the fact that the set of weighted languages can be represented as a final coalgebra, they use coinduction to show that the *Newton transform* generalizes from infinite sequences to weighted languages. The concept of coinduction also allows them to compactly establish connections between, and to show properties of, products of automata. As such, the infiltration product appears as the newton transform of the Hadamard product, and similarly, the shuffle product can be related to the concatenation. The established connection via the Newton transform, represents another way to show that the infiltration product preserves rationality.

In [241] Kari studies variants of deletion operations generalizing the left/right quotient of languages. The primary focus of the work is how the considered deletions can be expressed by a combination of other operations. In the secondary focus lie closure properties of different language families with respect to deletion operators. For example, it is shown that the sequential deletion of *any language* from a regular language results in yet another regular set. Conceptually, the reduced shuffle operator studied hereafter can be understood as operator with an inherent (partial) deletion operator. Although, we will not establish any connections between the proposed reduced shuffle product and any deletion operator, we do consider [241] to be related work in a broadened sense at least. The study of representing the proposed reduced shuffle product, via an existing product operator in combination with a deletion operator can be considered as future work.

2.6.1 Preliminaries Related to Finite Automata

We begin by giving the basic notions for automata we are using in this section. We use standard notation following [242] and [243].

Let Σ be a non-empty set, called the *alphabet*, its elements are also called *letters*. The set of sequences (or words) of length n over Σ is denoted by Σ^n, more precisely $\Sigma^n := \{(a_0, \ldots, a_{n-1}) \mid \forall i < n\, a_i \in \Sigma\}$. The set of finite sequences over Σ is called the set of *words over* Σ and is denoted by Σ^*, i.e., $\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n$, and the empty word is denoted by ε. Further, $|v|$ denotes the length of a word, i.e. for $v = (a_0, \ldots, a_{n-1})$, we have $|v| = n$, and for a language $L \subseteq \Sigma^*$ we denote its cardinality as $\#L$.

The set of words is equipped with a product $v \cdot w$ by concatenation, i.e., for $v = (a_0, \ldots, a_{n-1})$ and $w = (b_0, \ldots, b_{m-1})$ we have $v \cdot w = (a_0, \ldots, a_{n-1}, b_0, \ldots, b_{m-1})$. In the following we denote words also more compactly as $v = a_0 \ldots a_{n-1}$ as a concatenation (of letters), omitting the operator "$\cdot$". However, for the sake of a more compact notation the concatenation of words is also written as vw, by omitting the product operator. In the sequel we will also consider other operators on words, we assume thereby that concatenation is binding stronger than these operators. By $v{\restriction}[i, j]$ we denote the subword of v from the i-th letter to the j-th. In particular, for $0 \leq i < j \leq n-1$, we have $v{\restriction}[i, j] = (a_i, \ldots, a_j)$, for $i = j$: $v{\restriction}[i, i] = (a_i)$, and for $j < i$ we define $v{\restriction}[i, j] = \varepsilon$ as the empty word. The set Σ^* equipped with the concatenation $\cdot$ and with ε as neutral element forms a monoid. We say v is a *subword* of $w = (w_0, \ldots, w_{n-1})$, if there is a set of indices $I = \{i_0, \ldots, i_k\}$ with $0 \leq i_0 < \ldots < i_k \leq |w|$ for which $v = w_I := (w_{i_0}, \ldots, w_{i_k})$. In this case w is also called a *super-string* of v.

Semirings and Formal Power Series. A semiring is an algebra $(S, +, \cdot, 0, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, with distribution laws ($(a+b)\cdot c = a\cdot c+b\cdot c$ and $c\cdot(a+b) = c\cdot a+c\cdot b$ for all $a, b, c \in S$), and $0\cdot 0 = 0\cdot a = 0$, $\forall a \in S$. Recall that in formal language theory, the Kleene-star L^* of a language $L \subseteq \Sigma^*$ is defined by $L^* = \cup_{n\in\mathbb{N}} L^n$, where L^n is the set of words resulting from concatenations of exactly n words of L [243]. We also consider *formal power series* over the monoid Σ^* with coefficients coming from a semiring S, formally these are mappings from Σ^* to a set S,

denoted $S\langle\langle\Sigma^*\rangle\rangle$, they can be written as $r = \sum_{w\in\Sigma^*}(r,w)w$, where (r,w) is the value of w under r. The subset $\{r \in S\langle\langle\Sigma^*\rangle\rangle : (r,w) = 0 \text{ for all but finitely many } w \in \Sigma^*\}$ is called *polynomials* and denoted $S\langle\Sigma^*\rangle$. The Kleene-star operator can also be generalized to formal power series over semirings S (i.e. mappings $r : \Sigma^* \to S$ for semirings S), and matrices $S^{n\times n}$ over semirings S – provided that S is a *Conway semiring*.

Definition 2.6.1 (Conway Semiring [243]) *A* Conway semiring *is an algebra* $(S,+,\cdot,0,1,^*)$*, such that* $(S,+,\cdot,0,1)$ *is a semiring, and* * *is an unary operation such that the following equalities hold*

1. *The* sum star identity *holds for all* $a,b \in S$*:* $(a+b)^* = (a^*b)^*a^*$.
2. *The* product star identity *holds for all* $a,b \in S$*:* $(ab)^* = 1 + a(ba)^*b$.

In the following we consider the Boolean semiring $(\mathbb{B},+,\cdot,0,1,^*)$ equipped with the a star-operation $0^* = 1$ and $1^* = 1$ is a Conway semiring.

Definition 2.6.2 (Complete Semiring [243]) *A* complete semiring *is a semiring* $(S,+,\cdot,0,1)$*, such that*

1. $(S,+,0)$ *is a complete monoid, i.e. arbitrary (infinite) sums are well defined,*
2. *and the distributivity of* $\cdot$ *over* $+$ *holds also for infinite sums.*

If S is a complete semiring, then a star operator can be defined as $a^* = \sum_{n\in\mathbb{N}} a^n$ for all $a \in S$, making $(S,+,\cdot,0,1,^*)$ a *complete star semiring*. It is known that this definition of the star operator * makes $(S,+,\cdot,0,1,^*)$ also a Conway semiring, see [243] and references therein.

The Boolean semiring $(\mathbb{B},+,\cdot,0,1,^*)$ is also a complete star semiring [243]. In this section we will mainly consider power series in $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$, i.e. over Σ^* with coefficients in $\mathbb{B}$. They can be identified with the corresponding formal language, via their support. For ease of notation we use the notation $\subseteq$ for power series over $\mathbb{B}$, when it is actually meant for the corresponding formal languages. It is known that for a Conway semiring S also the power series $S\langle\langle\Sigma^*\rangle\rangle$ can be interpreted as a Conway semiring [243, Thm. 2.7]. To illustrate these concepts we give a simple example.

Example 2.6.3 *Let* $\Sigma = \{a,b,c\}$*. The words over* Σ *are all finite sequences of the letters* a,b *and* c*, for instance* $bcba, aa, \varepsilon$ *and* $bbbbba$*. An example of a formal power series in* $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ *is* $r = 1\varepsilon + 1aa + 1bcc$*. The coefficients* (r,w) *of* r *are given by* $(r,w) = 1$ *for* $w \in \{\varepsilon, aa, bcc\}$*, and* $(r,w) = 0$ *for* $w \in \Sigma^* \setminus \{\varepsilon, a, a, bcc\}$*. Moreover, the formal language corresponding to* r *is given by* $\{\varepsilon, aa, bcc\}$*.*

The following definition is a re-statement from [242], also to become familiar with the notation used hereafter. We define finite (non-deterministic) S-automata for a Conway semiring S, analog to *S'-automata* in [243], and similar to *$\mathbb{K}$-automata* in [242]

Definition 2.6.4 *Let S be a Conway semiring, then a* finite (non-deterministic) S-automaton *(or simply* automaton *in the following) $\mathfrak{A}$ is a tuple $\langle Q, \Sigma, E, I, T\rangle$ where*

(1) Q is a non-empty finite set, called the set of states *of $\mathfrak{A}$;*

(2) Σ is a non-empty finite set, called the alphabet *of $\mathfrak{A}$;*

(3) $I, T \subseteq Q$ where I is called the set of initial states *and T the set of* final states *of $\mathfrak{A}$.*

(4) $E \subseteq Q \times S\langle\langle\Sigma^*\rangle\rangle \times Q$, called the set of transitions *of $\mathfrak{A}$.*

In the following we will consider finite (non-deterministic) $\mathbb{B}$-automata (termed *automata* for short), i.e. the elements of E could contain formal power series in $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$, however, mostly we consider simply such automata where the elements of E contain elements of Σ.

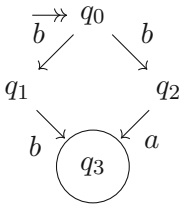
We may alter the notation for the sake of improved readability at some places, where we interpret E as an element of $\mathscr{P}(\Sigma)^{Q\times Q}$, instead of a subset of $Q \times \Sigma \times Q$, and refer to it as the *transition matrix*. The elements of the transition matrix are also referred to as *weights* (of transitions). So for $p, q \in Q$ the entry $E(p, q)$ is the set $\{a \in \Sigma \mid (p, a, q) \in E\}$. We denote this matrix again by E, and also write $E(p, q)+a$ instead of $E\cup\{(p, a, q)\}$. We say that an automaton $\mathfrak{A}' = \langle Q', \Sigma', E', I', T'\rangle$ is a subautomaton of $\mathfrak{A} = \langle Q, \Sigma, E, I, T\rangle$, if $Q' \subseteq Q, \Sigma' \subseteq \Sigma, E' \subseteq E, I' \subseteq I$ and $T' \subseteq T$. If $(p, a, q) \in E$ we write $p \xrightarrow{a} q$ and call p the *source* and q the *destination* of the *transition* (p, a, q). A *computation* c from p to q in $\mathfrak{A}$ is a sequence of transitions $\langle(p_i, a_i, q_i) \mid i < n\rangle$ where $p_{i+1} = q_i$ for each $i < n$ and with $p_0 = p$ and $q_{n-1} = q$. So, we have $p_i \xrightarrow{a_i} q_i = p_{i+1} \xrightarrow{a_{i+1}} q_{i+1}$ for all $i < n$, and therefore the computation can be written as

$$p \xrightarrow{a_0} p_1 \xrightarrow{a_1} p_2 \ldots p_{n-2} \xrightarrow{a_{n-2}} p_{n-1} \xrightarrow{a_{n-1}} q. \tag{2.47}$$

A computation is *successful* if its source p is an initial state and its destination q is a final state. A word $w \in \Sigma^*$ is accepted by $\mathfrak{A}$ if it is the sequence of labels a_i of a successful computation in $\mathfrak{A}$. The set of words accepted by $\mathfrak{A}$ is called *the language accepted* or *recognized* by $\mathfrak{A}$. From the point of view of graph theory, finite non-deterministic automata can be considered as directed weighted graphs where the set of vertices corresponds to the set of states and the directed edges correspond to the transitions. If (p, a, q) is a transition, then this can be understood as a directed edge from p to q with weight a. From this perspective, a computation from p to q in the automaton is a path from p to q in the directed graph.

In the following we will consider automata $\mathfrak{A}$ that accept a single word, which can be pictured as in (2.47), with initial state p and final state q. We call such automata *linear automata.*, and may denote them by $\mathfrak{A}_w$. We further follow the convention that the empty word $\varepsilon \in \Sigma^*$ is considered as the label set of a computation from p to p for each $p \in Q$.

Example 2.6.5 *Let* $\mathfrak{A} = \langle\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{(q_0, b, q_1), (q_0, b, p_2), (q_1, b, q_3), (q_2, a, q_3)\}, \{q_0\}, \{q_3\}\rangle$. *The graph representation of* $\mathfrak{A}$ *is given in Figure 2.17, as is the matrix presentation of* $E = \{(q_0, b, q_1), (q_0, b, p_2), (q_1, b, q_3), (q_2, a, q_3)\}$. *In the graph representation we indicate the initial state by "*$\twoheadrightarrow$*" and the final state by a circle. In* $\mathfrak{A}$ *there are two successful computations:* $q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_3$ *and* $q_0 \xrightarrow{b} q_2 \xrightarrow{a} q_3$, *therefore the language accepted by* $\mathfrak{A}$ *is* $\{ba, bb\}$.



$$E = \begin{array}{c|cccc} & q_0 & q_1 & q_2 & q_3 \\ \hline q_0 & 0 & b & b & 0 \\ q_1 & 0 & 0 & 0 & b \\ q_2 & 0 & 0 & 0 & a \\ q_3 & 0 & 0 & 0 & 0 \end{array}$$

Figure 2.17: On the left, the graph representation of the automaton $\mathfrak{A}$ given in Example 2.6.5; on the right its transition matrix E indexed with the states q_0, q_1, q_2, q_3 for better readability.

Definition 2.6.6 *(See also [242, Definition 2.3]) An automaton* $\mathfrak{A} = \langle Q, A, E, I, T\rangle$ *is normalised if:*

(i) the set of initial states is a singleton set $I = \{p\}$ *which is not the destination of any transition of* $\mathfrak{A}$*;*

(ii) the set of final states is a singleton set $T = \{q\}$ *which is not the source of any transition of* $\mathfrak{A}$*;*

(iii) p *is distinct from* q.

It is known that if L *is a language recognised by a finite* S*-automaton for a Conway semiring* S, *then there is a finite normalized* S*-automaton that recognizes* L, *see also [243, Theorem 2.11.].*

Power series of $S\langle\langle\Sigma^*\rangle\rangle$ *are connected to finite automata via the Theorem of Kleene-Schützenberger [244, Thm. 2.5]: A power series is rational if and only if it is the behavior of a cycle-free finite automaton over the semiring* S *and the alphabet* Σ.

When we compute the square of the transition matrix E, *again indexed by the states* q_i *as*

$$E^2 = \begin{array}{c|cccc} & q_0 & q_1 & q_2 & q_3 \\ \hline q_0 & 0 & 0 & 0 & bb + ba \\ q_1 & 0 & 0 & 0 & 0 \\ q_2 & 0 & 0 & 0 & 0 \\ q_3 & 0 & 0 & 0 & 0 \end{array}$$

where we find in position (q_i, q_j) of E all paths from state q_i to the state q_j in the automaton $\mathfrak{A}$. E^3 is the zero matrix, which tells us that there are no paths in $\mathfrak{A}$ of length (greater) equal to three. The accepted language of $\mathfrak{A}$ can then be computed as a matrix product, multiplying $E + E^2$ with the incidence vector of the set of initial states from left, and with that of the final states from the right as:

$$\begin{aligned}(1,0,0,0)\cdot(E+E^2)\cdot(0,0,0,1)^T &= (1,0,0,0)\cdot\begin{pmatrix}0&b&b&bb+ba\\0&0&0&b\\0&0&0&a\\0&0&0&0\end{pmatrix}\cdot(0,0,0,1)^T\\ &= bb+ba.\end{aligned}$$

The observation made in the previous example does generalizes to paths of arbitrary length and for automata in general.

Lemma 2.6.7 *(See [242, Lemma 2.11]) Let $\mathfrak{A} = \langle Q, \Sigma, E, I, T\rangle$ be an (S-)automaton (for a Conway semiring S), then for every integer n, E^n is the matrix containing the sums of the weights of transitions appearing in paths of length n, in particular $E^n(q_i, q_j)$ is the sum of the weights of transitions of all paths of length n from q_i to q_j in $\mathfrak{A}$.*

The above lemma indicates already to the importance of matrices for describing the behaviour of automata, as upon multiplication of E^n with the incidence vectors of initial (I) and final states (T), we can easily obtain the sum of weights of all paths on length n corresponding to accepted computations. When the transition weights only consist of letters, i.e. words of length one, then this cpmputation yields all accepted words of length n. Intuitively, if we could compute $\sum_{n\in\mathbb{N}} E^n$, then $I\cdot(\sum_{n\in\mathbb{N}} E^n)\cdot T$ yields the entire accepted language by an automaton with transition matrix E. If S is a complete semiring, than $\sum_{n\in\mathbb{N}} E^n$ is also defined for matrices in $(S\langle\langle\Sigma^*\rangle\rangle)^{n\times n}$, yielding a well defined notion of E^*. However, also more generally the star of a matrix can also be defined for Conway semirings – indeed each complete star semiring is a Conway semiring [243, Theorem 3.4.].

The formalization of this interpretation allows to describe the behaviour of automata by means of matrices and rational expressions, and allows therefore to formulate results for automata using matrix arithmetics. This will be used in Subsection 2.6.4.

The Star of a Matrix and the Behaviour of an Automaton

Definition 2.6.8 *For a star semiring S, a star operator can be defined for the semiring of $n\times n$ matrices over S $(S^{n\times n}, +, \cdot, \mathbf{0}, id_n, ^*)$ inductively as follows ([243]):*

- *For $n = 1$: for all $E = (a) \in S^{1\times 1}$: $E^* := (a^*)$.*

- *For $n > 1$: for $E \in S^{n\times n}$, partition E into blocks: $E = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $a \in S^{1\times 1}$, $b \in S^{1\times(n-1)}$, $c \in S^{(n-1)\times 1}$ and $d \in S^{(n-1)\times(n-1)}$ and define*

$$E^* := \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tag{2.48}$$

where $A := (a + bd^*c)^*$, $B := Abd^*$, $C := Dca^*$ and $D := (d + ca^*b)^*$.

Therefore, when computing the star of a matrix E^* for $E \in S^{n\times n}$, the entries of E^* ultimately compute as rational expressions in the underlying star semiring $(S, +, \cdot, 0, 1, ^*)$. Further, it can be shown, that for Conway semirings, the separation of a matrix into sub-matrices is not bound to the one used in Definition 2.6.8:

Theorem 2.6.9 (Matrix star identity) *([243, Theorem 2.7]) Let $(S, +, \cdot, 0, 1, ^*)$ be a Conway semiring, and $E \in S^{n\times n}$ be an element of the Conway semiring $(S^{n\times n}, +, \cdot, \mathbf{0}, id_n, ^*)$, where we decompose as $E = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, into blocks A, B, C and D, where A and D are again square matrices: $A \in S^{n_1\times n_1}$, $D \in S^{n_2\times n_2}$ with $n = n_1 + n_2$. Then*

$$E^* = \begin{pmatrix} (A + BD^*C)^* & A^*B(D + CA^*B)^* \\ D^*C(A + BD^*C)^* & (D + CA^*B)^* \end{pmatrix}. \tag{2.49}$$

The notion of the star of a matrix allows to define the following.

Definition 2.6.10 (Behaviour of an S-automaton [243]) *Let S be a Conway semiring, and $\mathfrak{A} = \langle Q, \Sigma, E, I, T\rangle$ be a finite S-automaton, then* the behaviour *of $\mathfrak{A}$ is defined as*

$$|\mathfrak{A}| := I \cdot E^* \cdot T. \tag{2.50}$$

It is known that for complete star semirings the star of a matrix introduced as above coincides with the notion of $E^* = \sum_{n\in\mathbb{N}} E^n$ [243]. Together with the property in Lemma 2.6.7, it does not come as a surprise that the behaviour of a finite S-automaton over a complete semiring does coincide with the accepted language of the automaton.

Theorem 2.6.11 *(See [243, Theorem 2.10]) Let S be a complete semiring and $\mathfrak{A}$ be a finite S-automaton, then $|\mathfrak{A}|$ is the sum of the weights of all paths from an initial state to a final state.*

As a final comment of this compact review of notions and concepts of automata theory, we mention that an immediate consequence of the equality of *accepted language* and the *behaviour* of an automaton is that any language recognizable by an S-automaton over Σ is also rational, as it can be obtained as a rational expression in $(S\langle\langle\Sigma^*\rangle\rangle, +, \cdot, 0, 1, ^*)$. The *Kleene-Schützenberger Theorem* [243, Thm. 2.12] establishes also the inverse direction, that for each rational language there exists an automaton having this language as behaviour.

Lemmata for Linear Automata

Below we formulate some properties that follow immediately from the matrix star identity (Lemma 2.6.9), which will be helpful for the technical proofs in Subsection 2.6.4.

Corollary 2.6.12 *Let S be a continuous semiring and E a $Q \times Q$ matrix over S.*

1. *If $E = \begin{pmatrix} A & \mathbf{0} \\ \mathbf{0} & D \end{pmatrix}$, where A and D are square matrices, then $E^* = \begin{pmatrix} A^* & \mathbf{0} \\ \mathbf{0} & D^* \end{pmatrix}$.*
2. *If $E = \begin{pmatrix} A_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & A_r \end{pmatrix}$ is a block-diagonal matrix with square matrices A_i for $i = 1, \ldots, r$, then*

$$E^* = \begin{pmatrix} A_1^* & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & A_r^* \end{pmatrix}. \tag{2.51}$$

Proof: The first statement follows directly from Theorem 2.6.9, since C, and D are zero-matrices. The second statement then follows by induction on r. □

In the following we will fequently consider matrices where a word appears in the upper secondary diagonal, and that are zero otherwise. That is, for a word of length ℓ, $w = w_1 w_2 \cdots w_\ell \in \Sigma^*$ let us define $E_w \in S\langle\langle\Sigma\rangle\rangle^{(\ell+1)\times(\ell+1)}$ by $E_w(i,j) := \begin{cases} w_i & , \ j = i+1 \\ 0 & , \ \text{otherwise} \end{cases}$, or more illustrative:

$$E_w = \begin{pmatrix} 0 & w_1 & 0 & \cdots & 0 \\ 0 & 0 & w_2 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & 0 & w_\ell \\ 0 & & \cdots & & 0 \end{pmatrix}. \tag{2.52}$$

The matrix E_w can be interpreted as the transition matrix of an automaton with initial state q_0 and final state q_ℓ that accepts only the word w – it is the transition matrix of a

linear automaton. It is clear that $E_w^k(i,j) = w_i \cdots w_{j-1}$, if $j - i = k$ and $E_w^k(i,j) = 0$ otherwise, and thus since $E_w^* = \sum_{k=0}^{\ell+1} E_w^k$ we have $E_w^*(i,j) = w_i \cdots w_{j-1}$, if $j > i$ and $E_w^*(i,j) = 0$ otherwise.

Further, we consider matrices $E_w \backslash a$, where a word w appears again in the upper secondary diagonal (similar as in equation (2.52)), but all occurrences of a given letter $a \in \Sigma$ are replaced by 0, i.e. for $w = w_1 w_2 \cdots w_\ell \in \Sigma^*$ we define $E_w \backslash a \in S\langle\langle \Sigma \rangle\rangle^{(\ell+1)\times(\ell+1)}$ by

$$E_w \backslash a(i,j) \quad := \quad \begin{cases} w_i & , \ w_i \neq a \wedge j = i+1, \\ 0 & , \ \text{otherwise.} \end{cases} \tag{2.53}$$

Lemma 2.6.13 *Let $w \in \Sigma^*$ be a word of lengh ℓ, and $E_w \backslash a \in S\langle\langle \Sigma \rangle\rangle^{(\ell+1)\times(\ell+1)}$ as defined in* (2.53)*, then*

$$(E_w \backslash a)^*(i,j) = \begin{cases} w_i \cdots w_{j-1}, & \text{for } j > i \text{ and } a \notin \{w_i, \ldots, w_{j-1}\} \\ \varepsilon, & \text{for } i = j \\ 0, & \text{otherwise} \end{cases} .$$

Proof: The assertion follows from the fact that $(E_w \backslash a)^*(i,j)$ contains the sum of weights of all paths from the i-th state to the j-th state in the corresponding automaton. □

Known Products of Finite Automata

The Hadamard product of power series generalizes the notion of intersections of languages, see [243]: if r_1 and r_2 are the characteristic series of the languages L_1 and L_2, then $r_1 \odot r_2$ is the characteristic series of $L_1 \cap L_2$. This translates to finite automata as follows:

Proposition 2.6.14 (Hadamard Product of finite Automata, see also [242]) *Let $\mathfrak{A}_1 = \langle Q_1, \Sigma, E_1, I_1, T_1 \rangle$ and $\mathfrak{A}_2 = \langle Q_2, \Sigma, E_2, I_2, T_2 \rangle$ be two finite automata, then the Hadamard product of their behaviours $|\mathfrak{A}_1| \odot |\mathfrak{A}_2|$ is recognized by the automaton $\mathfrak{A}_1 \odot \mathfrak{A}_2 = \langle Q_1 \times Q_2, \Sigma, E_\odot, I_1 \times I_2, T_1 \times T_2 \rangle$, where the transition matrix is described by*

$$E_\odot = \{(p_1, q_1), a, (p_2, q_2) : (p_1, a, p_2) \in E_1 \wedge (q_1, a, q_2) \in E_2\}.$$

We refer to the automaton described above as *the Hadamard product automaton* of $\mathfrak{A}_1$ and $\mathfrak{A}_2$, although this is not really correct. If we are very precise, then the formulation *the* Hadamard product automaton is not justified, because there exist multiple automata having $|\mathfrak{A}_1| \odot |\mathfrak{A}_2|$ as behaviour. For example in the definition above some states might be never reached, so called *dead states* which can be ommitted without changing the behaviour. This ultimately leads to the problem of finding a minimal automaton that recognizes a language [242, Sec. 3.3]. However, we do not treat this problem here, and stick with the formulation *the automaton* when relating to products of automata, as we are primarily concerned with the behaviour.

Definition 2.6.15 (Shuffle Product over a Semiring S [242]) *For $v, w \in \Sigma^*$ the* shuffle product[4] *$v \sqcup\!\sqcup w$ of v and w, is a homogeneous polynomial in $S\langle\Sigma^*\rangle$, which can be defined by induction on $|v| + |w|$ by*

$$\begin{aligned} \forall v \in \Sigma^* : \quad & v \sqcup\!\sqcup \varepsilon = \varepsilon \sqcup\!\sqcup v = v, \\ \forall a, b \in \Sigma, \forall v, w \in \Sigma^* : \quad & va \sqcup\!\sqcup wb = (v \sqcup\!\sqcup wb)a + (va \sqcup\!\sqcup w)b. \end{aligned}$$

The shuffle product is linearly extended to $S\langle\langle\Sigma^*\rangle\rangle$ by:

$$\forall r, u \in S\langle\langle\Sigma^*\rangle\rangle : \quad r \sqcup\!\sqcup u = \sum_{v,w \in \Sigma^*} (r, v)(u, w) v \sqcup\!\sqcup w.$$

Accompanying with the above recursive, and rather technical definition, we also state the following characterization of the shuffle product of two words v and w, as the set of words h such that v and w appear as disjoint subwords of h, see also [242]:

$$v \sqcup\!\sqcup w \quad = \quad \{h \in \Sigma^* : \exists I, J \subseteq [|h|] : I \cup J = [|h|] \wedge I \cap J = \emptyset \wedge h_I = v \wedge h_J = w\}$$

Definition 2.6.16 (Infiltration Product over a Semiring S [242]) *For $v, w \in \Sigma^*$ the* infiltration product *$v \uparrow w$ of v and w, is a polynomial in $S\langle\Sigma^*\rangle$, which can be defined by induction on $|v| + |w|$ by*

$$\begin{aligned} \forall v \in \Sigma^* : \quad & v \uparrow \varepsilon = \varepsilon \uparrow v = v, \\ \forall a, b \in \Sigma, \forall v, w \in \Sigma^* : \quad & va \uparrow wb = (v \uparrow wb)a + (va \uparrow w)b + \delta_{a,b}(v \uparrow w)a, \end{aligned}$$

where $\delta_{a,b} = 1$ if $a = b$ and $\delta_{a,b} = 0$ otherwise. The infiltration product is again linearly extended to $S\langle\langle\Sigma^*\rangle\rangle$ by:

$$\forall r, u \in S\langle\langle\Sigma^*\rangle\rangle : \quad r \uparrow u = \sum_{v,w \in \Sigma^*} (r, v)(u, w) v \uparrow w.$$

Again, we state the following characterization of the infiltration product of two words v and w, as the set of words h such that v and w appear as (not necessarily disjoint) subwords of h, see also [242]:

$$v \uparrow w \quad = \quad \{h \in \Sigma^* : \exists I, J \subseteq [|h|] : I \cup J = [|h|] \wedge h_I = v \wedge h_J = w\} \tag{2.54}$$

Examples of the shuffle- and infiltration product applied to words are given below in Example 2.6.21.

It is known that rational languages are closed under shuffle and infiltration [239], particularly, if $\mathfrak{A}_1 = \langle P, \Sigma, E_1, I_1, T_1\rangle$ and $\mathfrak{A}_2 = \langle Q, \Sigma, E_2, I_2, T_2\rangle$ are two automata

[4] Here we deviate from the notation used in [242], where the symbol ⧢ is used; instead we stick to the notation $\sqcup\!\sqcup$ used in [243].

recognizing L_1 and L_2, respectively, then $L_1 \uparrow L_2$ is recognized by the automaton $\mathfrak{A}_{L_1 \uparrow L_2} = \langle P \times Q, \Sigma, E_\uparrow, I_1 \times I_2, T_1 \times T_2 \rangle$ (see also [242, Chapter 1]), where

$$\begin{aligned} E_\uparrow \;:=\; & \{[(p,q),a,(p',q)] : (p,a,p') \in E_1 \wedge q \in Q\} \\ & \cup \{[(p,q),a,(p,q')] : p \in P \wedge (q,a,q') \in E_2\} \\ & \cup \{[(p,q),a,(p',q')] : (p,a,p') \in E_1 \wedge (q,a,q') \in E_2\} \end{aligned}$$

2.6.2 Connection Between Sequential Combinatorial Testing and Known Products of Automata

Let us first import the definition of t-wise automata from [123, Def. 5].

Definition 2.6.17 *For a given alphabet Σ and a word $w \in \Sigma^t$, $w = a_1, \ldots, a_t$, the* t-wise automaton *corresponding to w is the following automaton $\mathfrak{A}_{w\leq} = \langle Q, \Sigma, E, I, T \rangle$ (see also Figure 2.18), where $Q = \{p_0, p_1, \ldots, p_t\}$, $I = \{p_0\}$, $T = \{p_t\}$, and $E = \{(p_i, a_{i+1}, p_{i+1}) : 0 \leq i < t-1\}$, respectively denoted as a matrix*

$$E(i,j) = \begin{cases} \Sigma, & i = j, \\ a_j, & j = i+1, \\ 0, & else; \end{cases} \qquad i.e.: \quad E = \begin{pmatrix} \Sigma & a_1 & 0 & \cdots & 0 \\ 0 & \Sigma & a_2 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & \Sigma & a_t \\ 0 & & \cdots & & \Sigma \end{pmatrix}.$$

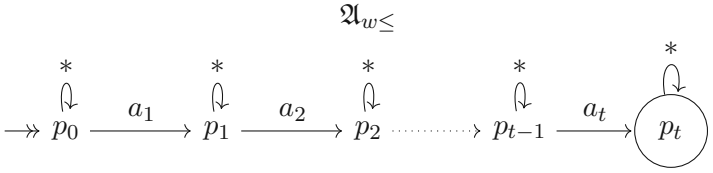


Figure 2.18: The t-wise automaton corresponding to $w = a_1 a_2 \ldots a_t$. Note that the weights "$*$" of loops represents any symbol of the alphabet, i.e. Σ.

It is apparent that a t-wise automaton corresponding to a word w accepts exactly the language consisting of all words which contain w as subwords: $|\mathfrak{A}_{w\leq}| = \{v \in \Sigma^* : w \text{ is a subword of } v\}$. This is also the motivation for our notation $\mathfrak{A}_{w\leq}$.

It is well known, that the Hadamard product ("the intersection") of automata accepts exactly the intersection of their behaviour [242, Sec. 3, Prop. 3.11]. Bombarda and Gargantini [123] make use of this fact, when iteratively considering the products of automata in their algorithms. The following theorem shows that the infiltration product of linear automata can be used to derive the Hadamard product of two t-wise automata.

Theorem 2.6.18 *Let $w_1, w_2 \in \Sigma^*$ be two words of length t. Then the addition of loops to each state of the infiltration product automaton of the two linear automata $\mathfrak{A}_{w_1}$ and $\mathfrak{A}_{w_2}$, results in an automaton with the same behaviour as the Hadamard product of the t-wise automata corresponding to w_1 and w_2. More formally, let $\mathfrak{A}_{w_1 \uparrow w_2} = \langle Q \times Q', \Sigma, E_\uparrow, I \times I', T \times T' \rangle$, be the infiltration product automaton of the two linear automata $\mathfrak{A}_{w_1}$ and $\mathfrak{A}_{w_2}$, then for $\mathfrak{A} := \langle Q \times Q', \Sigma, E, I \times I', T \times T' \rangle$ with $E := E_\uparrow \cup \{((p,q), \Sigma, (p,q)) : (p,q) \in Q \times Q'\}$ we have: $|\mathfrak{A}| = |\mathfrak{A}_{w_1 \leq} \odot \mathfrak{A}_{w_2 \leq}|$.*

Proof: To show the assertion we argue over subwords. Since $\mathfrak{A}_{w\leq}$ accepts all words that have w as subword, and the Hadamard product automaton accepts exactly the intersection of behaviours of the factor automata, we have $|\mathfrak{A}_{w_1 \leq} \odot \mathfrak{A}_{w_2 \leq}| = |\mathfrak{A}_{w_1 \leq}| \cap |\mathfrak{A}_{w_2 \leq}| = \{w \in \Sigma^* : w_1 \leq w \wedge w_2 \leq w\}$. The addition of loops to each state of an automaton results in an automaton that accepts a word if and only if it contains a word accepted by the original automaton as a subword. Thus, adding loops to the infiltration product automaton results in a behaviour equal to the language of all words that contain any word of $w_1 \uparrow w_2$ as subword: $\mathcal{L} = \bigcup_{w \in w_1 \uparrow w_2} \{\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \dots \Sigma^* a_n \Sigma^* : w = a_1 a_2 \dots a_n\}$. Since every $w \in w_1 \uparrow w_2$ has w_1 and w_2 as subword (see (2.54)), we immediately get $|\mathfrak{A}| \subseteq |\mathfrak{A}_{w_1 \leq} \odot \mathfrak{A}_{w_2 \leq}|$.

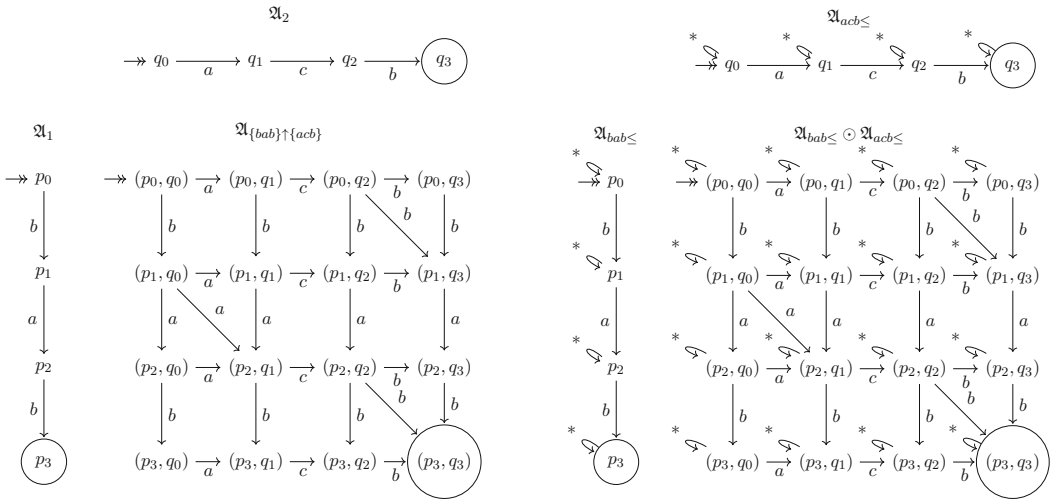
To show the other direction, we argue over minimal super-strings of w_1 and w_2, i.e. such where omission of a letter does not result in a super-string of w_1 and w_2 any more. From the characterization in equation (2.54) we get that $w_1 \uparrow w_2$ contains all super-strings of w_1 and w_2 with letters coming from w_1 or w_2 (or both). Since a minimal super-string of w_1 and w_2 cannot contain a letter not appearing in w_1, nor in w_2, $w_1 \uparrow w_2$ must already contain all minimal super-strings of w_1 and w_2. Since every super-string of w_1 and w_2, i.e. every word in $|\mathfrak{A}_{w_1 \leq} \odot \mathfrak{A}_{w_2 \leq}|$, is a super-string of a minimal super-string of w_1 and w_2, we get $|\mathfrak{A}_{w_1 \leq} \odot \mathfrak{A}_{w_2 \leq}| \subseteq |\mathfrak{A}|$. □

We illustrate the above theorem by means of the following.

Example 2.6.19 *We consider the words $w_1 = bab$, $w_2 = acb$ over $\Sigma = \{a, b, c\}$. The infiltration product $bab \uparrow acb$ consists of all words that are accepted by the automaton $\mathfrak{A}_{\{bab\}\uparrow\{acb\}}$ in Figure 2.19a, we do not give the explicit expression for $bab \uparrow acb$ as a formal polynom, but merely indicate it by: $bab \uparrow acb = acbbab + acbab + \dots + bacb + \dots + babacb$. For the two t-wise automata for $t = 3$ $\mathfrak{A}_{bab\leq}$ and $\mathfrak{A}_{acb\leq}$, we obtain their Hadamard product depicted in Figure 2.19b. When we add loops weighted with Σ (represented by "$*$" in the graphics) to each state of $\mathfrak{A}_{\{bab\}\uparrow\{acb\}}$, we obtain the Hadamard product $\mathfrak{A}_{bab\leq} \odot \mathfrak{A}_{acb\leq}$ of the 3-wise automata.*

2.6.3 Introducing a Reduced-Shuffle Product

In this section we define a reduced shuffle product and put it into context with the shuffle product and the infiltration product. Further, we give a construction of an automaton that represents the reduced shuffle product of two given automata, from which the rationality of the reduced shuffle product of two rational languages follows.



(a) Infiltration product $\mathfrak{A}_{\{bab\}\uparrow\{acb\}}$.

(b) Hadamard product $\mathfrak{A}_{bab\leq} \odot \mathfrak{A}_{acb\leq}$.

Figure 2.19: (a) The infiltration product automaton of the linear automata $\mathfrak{A}_{bab}$ and $\mathfrak{A}_{acb}$, which when adding loops weighted with the entire alphabet Σ (represented as $*$) to each state equals to (b) the Hadamard product of the 3-wise automata $\mathfrak{A}_{bab\leq}$ and $\mathfrak{A}_{acb\leq}$.

A Reduced-Shuffle Product

Definition 2.6.20 (Reduced-Shuffle Product over the Boolean Semiring $\mathbb{B}$) *For $v, w \in \Sigma^*$ the* reduced shuffle product *$v \dot{\sqcup\!\sqcup} w$ of v and w, is a polynomial in $\mathbb{B}\langle\Sigma^*\rangle$, which can be defined by induction on $|v| + |w|$ by*

$$\forall v \in \Sigma^* : \qquad v \dot{\sqcup\!\sqcup} \varepsilon = \varepsilon \dot{\sqcup\!\sqcup} v = v, \tag{2.55}$$

$$\forall a, b \in \Sigma, \forall v, w \in \Sigma^* : \qquad va \dot{\sqcup\!\sqcup} wb = \begin{cases} (v \dot{\sqcup\!\sqcup} wb)a + (va \dot{\sqcup\!\sqcup} w)b, & \text{if } a \neq b, \\ (v \dot{\sqcup\!\sqcup} w)a, & \text{if } a = b. \end{cases} \tag{2.56}$$

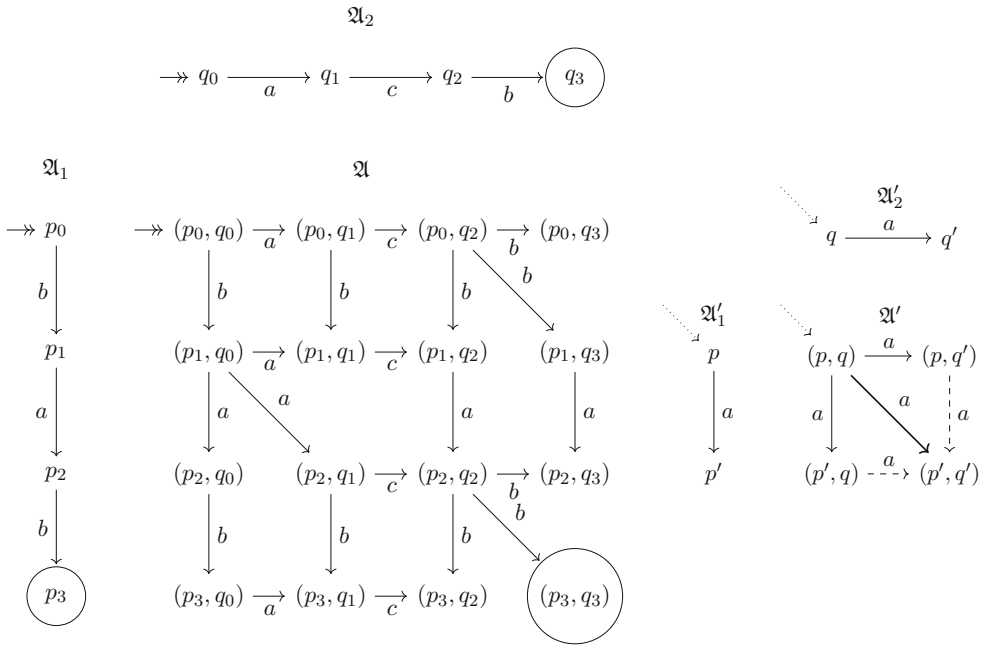
The reduced shuffle product is again linearly extended to $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ by:

$$\forall r, u \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle : \qquad r \dot{\sqcup\!\sqcup} u = \sum_{v,w \in \Sigma^*} (r, v)(u, w) v \dot{\sqcup\!\sqcup} w.$$

Example 2.6.21 *We compare the three products defined above by means of the words $v = ba$ and $w = a$ over the Boolean semiring $\mathbb{B}$:*

1. $ba \sqcup\!\sqcup a = (b \sqcup\!\sqcup a)a + (ba \sqcup\!\sqcup \varepsilon)a = ((\varepsilon \sqcup\!\sqcup a)b + (b \sqcup\!\sqcup \varepsilon)a)a + baa = aba + baa.$
2. $ba \uparrow a = (b \uparrow a)a + (ba \uparrow \varepsilon)a + (b \uparrow \varepsilon)a = ((\varepsilon \uparrow a)b + (b \uparrow \varepsilon)a)a + baa + ba = aba + baa + ba.$
3. $ba \dot{\sqcup\!\sqcup} a = (b \dot{\sqcup\!\sqcup} \varepsilon)a = ba.$

We can see that in this example the reduced shuffle product of v and w is the smallest language, with respect to number of words and also with respect to the length of words. It contains a single word that has v and w as a subword. Both, the shuffle product and the infiltration product also contain words having v and w as a subword, but they are at least as long as the word in the reduced shuffle product. Further, they contain more such words, i.e., they are larger languages with regards to cardinality. The infiltration product does contain the short word ba, but it also contains longer words. The shuffle product, considered over $\mathbb{B}$, contains two words of length three, but it does not contain the short word ba.



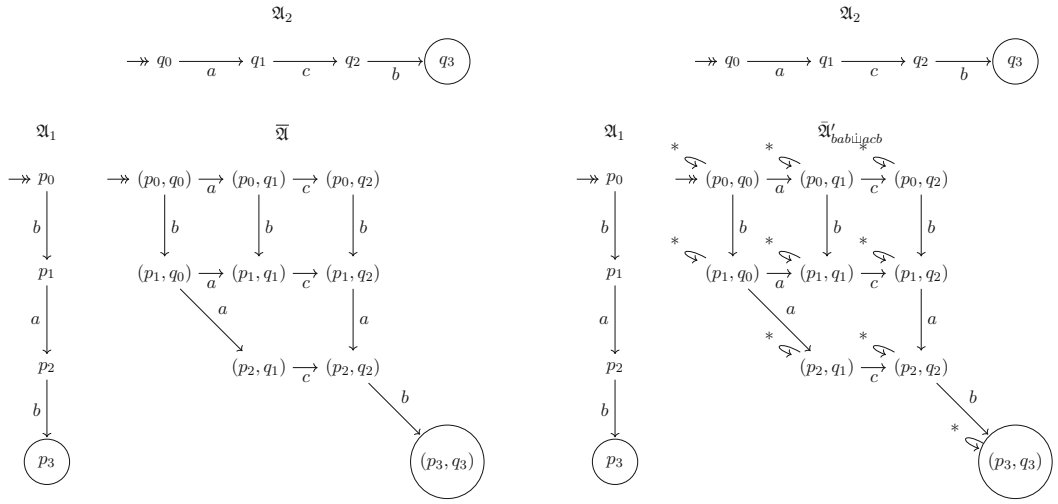
(a) Automaton $\mathfrak{A}$ with $|\mathfrak{A}| = |\mathfrak{A}_1|\sqcup\!\sqcup|\mathfrak{A}_2| = bab\sqcup\!\sqcup acb$. (b) Enforced synchronization.

Figure 2.20: (a) An automaton with the behaviour $bab\sqcup\!\sqcup acb$ and (b) a visualization of the enforced synchronization (bold diagonal transition) in reduced shuffle products of linear automata; the dashed transitions are removed in comparison to the infiltration product.

Example 2.6.19 (continuing from p. 124) *We consider again the words $w_1 = bab$, $w_2 = acb$ over $\Sigma = \{a, b, c\}$. The reduced shuffle product $bab\sqcup\!\sqcup acb$ explicitly computes as*

$$\begin{aligned} bab\sqcup\!\sqcup acb &= (ba\sqcup\!\sqcup ac)b \\ &= (b\sqcup\!\sqcup ac)ab + (ba\sqcup\!\sqcup a)cb \\ &= (\varepsilon\sqcup\!\sqcup ac)bab + (b\sqcup\!\sqcup a)cab + (b\sqcup\!\sqcup\varepsilon)acb \\ &= acbab + abcab + bacab + bacb. \end{aligned} \tag{2.57}$$

That is, bab⧢acb consists of all words that are accepted by the automaton $\mathfrak{A}_{\{bab\}⧢\{acb\}}$ *in Figure 2.21a (as we will show in general in Subsection 2.6.4). Several states including their incident transitions of* $\mathfrak{A}_{\{bab\}⧢\{acb\}}$ *in Figure 2.21a can be reduced, as they do not appear in any successful computation. Removing these states and transitions, results in a reduced automaton* $\mathfrak{A}'_{\{bab\}⧢\{acb\}}$ *with the same behaviour* $|\mathfrak{A}'_{\{bab\}⧢\{acb\}}| = |\mathfrak{A}_{\{bab\}⧢\{acb\}}|$.



(a) $\overline{\mathfrak{A}}$ with $|\bar{\mathfrak{A}}| = |\mathfrak{A}_1|⧢|\mathfrak{A}_2| = bab⧢acb$.

(b) Automaton $\overline{\mathfrak{A}}$ with added loops.

Figure 2.21: (a) A reduced automaton where dead states and transitions are removed from the automaton in Fig. 2.20a, with behaviour *bab⧢acb*, (b) the automaton in (a) with added loops on each state.

Remark 2.6.22 (Motivation and Connection to sequential CT) *In sequential CT as considered in [122] and [123] we are concerned with the derivation of test sequences that cover all t-way sequences that can be executed on an SUT that is modeled by means of an automaton. One test sequence can cover multiple t-way sequences, and when we are concerned with deriving a small number of short test sequences, then we want to cover many t-way sequences in a single test sequence. That is, in terms of automata theory, we are interested in a synchronized computation of many t-wise automata and the automaton representing the SUT. The synchronized computation of many t-wise automata can be attained in different ways, other than considering the Hadamard product of t-wise automata. To elaborate, we reflect on the above mentioned products of automata, we note once more on their characteristics which are made visible also in the Figures 2.21, 2.20 and 2.19. The Hadamard product ⊙ represents synchronized computations of its factor automata. The loops on each state of t-wise automata has as a result that their Hadamard product accepts words containing the interleaving of the two words corresponding to the t-wise automata. The shuffle product represents interleaved computation of its factor automata and the the infiltration product represents interleaved computation with an* optional *synchronization, if it is possible. The proposed reduced shuffle product represents an interleaved computation with enforced synchronization of its factor automata, whenever*

it is possible. When synchronizations are possible, this results in a product automaton that has less transitions than the one obtained via infiltration, respectively the Hadamard product of t-wise automata, while maintaining the property that the shortest words that cover both t-way sequences are still accepted by the reduced shuffle product automaton, as will be shown in Theorem 2.6.30 below.

Example 2.6.19 illustrates first, that the infiltration product provides a way to construct the Hadamard product (intersection) of two t-wise automata, and further, that the reduced shuffle product provides a way to construct an automaton that appears as a sub-automaton of the Hadamard product of two t-wise automata, but maintains the crucial property of accepting the shortest super-string of the words corresponding to the t-wise automata.

In the following we will generalize these observations for arbitrary words. Before that we show the following, basic and auxiliary result.

Proposition 2.6.23 *The reduced shuffle product acts commutatively on* $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$.

Proof: Commutativity on Σ^* can be shown via induction:
IB: $\varepsilon ⫛ w = w ⫛ \varepsilon = w$.
IH: For all w_1, w_2 with $|w_1| + |w_2| \leq n$ it holds that $w_1 ⫛ w_2 = w_2 ⫛ w_1$.
IS: Let $a, b \in \Sigma$ and $w_1, w_2 \in \Sigma^*$ with $|w_1| + |w_2| + 1 = n$.
Case 1: $a \neq b$: $w_1 a ⫛ w_2 b = (w_1 ⫛ w_2 b)a + (w_1 a ⫛ w_2)b = (w_2 ⫛ w_1 a)b + (w_2 b ⫛ w_1)a = w_2 b ⫛ w_1 a$. *Case 2:* $a = b$: $w_1 a ⫛ w_2 a = (w_1 ⫛ w_2)a = (w_2 ⫛ w_1)a = w_2 a ⫛ w_1 a$.
Commutativity on $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ follows from the linear extension of ⫛ from Σ^* to $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$. □

Remark 2.6.24 *We want to note that the reduced shuffle product, as defined in Definition 2.6.20 is not associative, as the following counter example shows. Assume* $a, c \in \Sigma$ *with* $a \neq c$, *then we have:* $a ⫛ (a ⫛ c) = a ⫛ (ac + ca) = a ⫛ ac + a ⫛ ca = aca + ac + ca$, *which is different from* $(a ⫛ a) ⫛ c = a ⫛ c = ac + ca$.

Proposition 2.6.25 *The reduced shuffle product of two words covers the two words. More precisely, given two words* $v \in \Sigma^*$ *and* $w \in \Sigma^*$, *they appear as subsequences of each of the words in* $v ⫛ w \in \mathbb{B}\langle\langle\Sigma^*\rangle\rangle$.

Proof: The proof is by induction on $|v|$.
IB: If $|v| = 0$ then $v = \varepsilon$. As ε is a subsequence of any word, v is a subsequence of each word in $v ⫛ w$ for any $w \in \Sigma^*$. Since $\varepsilon ⫛ w := w$, also w is a subsequence of each word in the shuffle product.
IH: Assume $|v| = n$ and v is a subsequence of each word in $v ⫛ w$ for any $w \in \Sigma^*$.
IS: We show that for $a \in \Sigma$, va is a subsequence of each word in $va ⫛ w$ for any $w \in \Sigma^*$. If $w = \varepsilon$, then $va ⫛ w = va$, and there is nothing to show.

Let $w = w'b$ for some $b \in \Sigma$ and $w' \in \Sigma^*$. By definition of $\dot{\sqcup}$

$$va\dot{\sqcup}w = \begin{cases} (v\dot{\sqcup}w)a + (va\dot{\sqcup}w')b \text{ if } a \neq b \\ (v\dot{\sqcup}w')a \text{ if } a = b. \end{cases}$$

By induction hypothesis va is a subsequence of each word in $va\dot{\sqcup}w'$ and v is a subsequence of each word in $v\dot{\sqcup}w$ and of each word in $v\dot{\sqcup}w'$. Therefore va is a subsequence of each word in $(va\dot{\sqcup}w')b$, $(v\dot{\sqcup}w)a$ and $(v\dot{\sqcup}w')a$. Using the commutativity of $\dot{\sqcup}$ it also follows that w is a subsequence of $v\dot{\sqcup}w$ for any $v \in \Sigma$. □

Proposition 2.6.26 *For $v, w \in \Sigma^*$, considering $v \uparrow w, v \sqcup\!\sqcup w$ and $v\dot{\sqcup}w$ as power series in $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle$ the following holds:*

$$v \uparrow w \supseteq v \sqcup\!\sqcup w + v\dot{\sqcup}w. \tag{2.58}$$

Proof: The proof is again by induction on $|v| + |w|$.
IB: $|v| + |w| = 0$: In this case $v = w = \varepsilon$ and $v \uparrow w = v \sqcup\!\sqcup w + v\dot{\sqcup}w = \varepsilon$.
IH: Assume $v \uparrow w \supseteq v \sqcup\!\sqcup w + v\dot{\sqcup}w$ for all $v, w \in \Sigma^*$ with $|v| + |w| = n$.
IS: For the induction step we show that $va \uparrow w \supseteq va \sqcup\!\sqcup w + va\dot{\sqcup}w$ for all $v, w \in \Sigma^*$ with $|v| + |w| = n$ and all $a \in \Sigma$. If $w = \varepsilon$, then $va \uparrow w = va = va \sqcup\!\sqcup w + va\dot{\sqcup}w$ (since multiplicities do not matter in $\mathbb{B}$).

Now let $w = w'b$ for some $b \in \Sigma$.

$$va \uparrow w = (v \uparrow w)a + (va \uparrow w')b + \delta_{a,b}(v \uparrow w')a,$$

which contains the following by induction hypothesis:

$$\begin{aligned}
&(v \sqcup\!\sqcup w + v\dot{\sqcup}w)a + (va \sqcup\!\sqcup w' + va\dot{\sqcup}w')b + \delta_{a,b}(v \sqcup\!\sqcup w' + v\dot{\sqcup}w')a \\
&= \begin{cases} (v \sqcup\!\sqcup w)a + (va \sqcup\!\sqcup w')b + (v\dot{\sqcup}w)a + (va\dot{\sqcup}w')b, \text{ if } a \neq b \\ (v \sqcup\!\sqcup w)a + (va \sqcup\!\sqcup w')a + (v\dot{\sqcup}w)a + (va\dot{\sqcup}w')a + (v \sqcup\!\sqcup w')a + (v\dot{\sqcup}w')a, \text{ if } a = b \end{cases} \\
&= \begin{cases} va \sqcup\!\sqcup w + va\dot{\sqcup}w, \text{ if } a \neq b \\ va \sqcup\!\sqcup w + (v \sqcup\!\sqcup w')a + va\dot{\sqcup}w + (v\dot{\sqcup}w)a + (va\dot{\sqcup}w')a \supseteq va \sqcup\!\sqcup w + va\dot{\sqcup}w, \text{ if } a = b. \end{cases}
\end{aligned}$$

Since $\sqcup\!\sqcup, \uparrow$ and $\dot{\sqcup}$ are commutative, we don't have to do the induction step for wa and the proof is finished. □

Note that the equality $v \uparrow w = v \sqcup\!\sqcup w + v\dot{\sqcup}w$ does not always hold, as the following example shows.

Example 2.6.27 *Let $v = w = ba$. We get the following equalities in $\mathbb{B}\langle\langle\{a,b\}^*\rangle\rangle$:*

1. $v \sqcup w = baba + bbaa$.
2. $v \uparrow w = baba + bbaa + bba + baa + ba$.
3. $v \dot{\sqcup} w = ba$.

We can see that $v \uparrow w \supseteq v \sqcup w + v \dot{\sqcup} w$ and $v \uparrow w \neq v \sqcup w + v \dot{\sqcup} w$ because $bba, baa \notin v \sqcup w + v \dot{\sqcup} w$.

Proposition 2.6.28 *The number of words in $v \dot{\sqcup} w$ is at most the number of words in $v \sqcup w$ for all $v, w \in \Sigma^*$.*

Proof: We denote the number of words in $v \dot{\sqcup} w$ and $v \sqcup w$ by $\#v \dot{\sqcup} w$ and $\#v \sqcup w$, and show the assertion by induction on $|v| + |w|$.
IB: If either $v = \varepsilon$ or $w = \varepsilon$ then $\#v \dot{\sqcup} w = \#v \sqcup w = 1$.
IH: We assume that we have shown the inequality for $|v| + |w| = n$.
IS: Now we show that the assertion holds for va with $a \in \Sigma$, and $w = w'b$ with $b \in \Sigma$:

$$\#va \dot{\sqcup} w = \begin{cases} \#((v \dot{\sqcup} w)a + (va \dot{\sqcup} w')b) \text{ if } a \neq b \\ \#(v \dot{\sqcup} w')a \text{ if } a = b. \end{cases}$$

For $a = b$, using the inductive hypothesis, we know that $\#(v \dot{\sqcup} w')a \leq \#(v \sqcup w')a = \#(v \sqcup w')ba \leq \#va \sqcup w$. For $a \neq b$ we know that $\#((v \dot{\sqcup} w)a + (va \dot{\sqcup} w')b) = \#(v \dot{\sqcup} w)a + \#(va \dot{\sqcup} w')b$, and, using the inductive hypothesis, we have $\#(v \dot{\sqcup} w)a + \#(va \dot{\sqcup} w')b \leq \#(v \sqcup w)a + \#(va \sqcup w')b$. Again, since $a \neq b$ we get $\#(v \sqcup w)a + \#(va \sqcup w')b = \#((v \sqcup w)a + (va \sqcup w')b) = \#va \sqcup w$. As above, using the commutativity of $\dot{\sqcup}$ and $\sqcup$ this finishes the proof. □

Note that the equality $\#v \dot{\sqcup} w = \#v \sqcup w$ does not always hold, as the following example shows.

Example 2.6.27 (continuing from p. 130) *We see immediately that $\#ba \sqcup ba = 2$ and $\#ba \dot{\sqcup} ba = 1$.*

The previous propositions and examples show that $\dot{\sqcup}$ fulfills the coverage property that v and w are covered by $v \dot{\sqcup} w$ and it is a smaller language than $\sqcup$ and $\uparrow$ with respect to length and number of words.

Proposition 2.6.29 *For $w_1, w_2 \in \Sigma^*$*

$$\inf\{|w| : w \in w_1 \dot{\sqcup} w_2\} = \inf\{|w| : w \in w_1 \uparrow w_2\}. \tag{2.59}$$

Proof: $\inf\{|w| : w \in w_1 \dot{\sqcup\!\sqcup} w_2\} \geq \inf\{|w| : w \in w_1 \uparrow w_2\}$ follows immediately from (2.58), since $w_1 \dot{\sqcup\!\sqcup} w_2 \subseteq w_1 \uparrow w_2$. In order to establish also $\inf\{|w| : w \in w_1 \dot{\sqcup\!\sqcup} w_2\} \leq \inf\{|w| : w \in w_1 \uparrow w_2\}$, we argue via shortest super strings of w_1 and w_2 and use an inductive argument regarding the number of synchronizations that appear in their computation in $\mathfrak{A}_{w_1 \uparrow w_2}$.
IB: No synchronization in the computation of a shortest word in $w_1 \uparrow w_2$, then $w_1 \uparrow w_2 = w_1 \sqcup\!\sqcup w_2 = w_1 \dot{\sqcup\!\sqcup} w_2$, and all words in $w_1 \uparrow w_2$ and $w_1 \dot{\sqcup\!\sqcup} w_2$ are of equal length $|w_1| + |w_2|$.
IH: If n synchronizations appear in the computation of a shortest word in $s \in w_1 \uparrow w_2$, then $s \in w_1 \dot{\sqcup\!\sqcup} w_2$.
IS: Assume $n+1$ synchronizations appear in the computation of a shortest word in $s \in w_1 \uparrow w_2$. Further assume $w_1 = v_1^1 a v_1^2$ and $w_2 = v_2^1 a v_2^2$ for $a \in \Sigma$ and $v_1^1, v_1^2, v_2^1, v_2^2 \in \Sigma^*$, such that $s \in (v_1^1 \uparrow v_2^1)at$, where $t \in v_1^2 \uparrow v_2^2$ and a is the first letter where synchronization is possible, i.e. in the computation of t there is no synchronization possible in its computation in $\mathfrak{A}_{w_1 \uparrow w_2}$ (it follows $t \in v_1^2 \sqcup\!\sqcup v_2^2$). It follows that there is an $h \in (v_1^1 \uparrow v_2^1)$, such that $s = hat$. Further, since s is a shortest super string of w_1 and w_2, h must be a shortest super string of v_1^1 and v_2^1 with a computation of at most n synchronizations: Otherwise, if there was a shortest super string h' of v_1^1 and v_2^1 of length $|v_1^1|+|v_2^1|-(n+1)$, there would be a shortest super string $s' = h'at \in (v_1^1 \uparrow v_2^1)at$ of length $|v_1^1|+|v_2^1|-(n+1)+1+|v_1^2|+|v_2^2| = |w_1|+|w_2|-(n+2)$; which is a contradiction to s being a shortest superstring of w_1 and w_2, i.e. a shortest word in $s \in w_1 \uparrow w_2$. It follows from the induction hypothesis that $h \in v_1^1 \dot{\sqcup\!\sqcup} v_2^1$, and further $s = hat \in (v_1^1 \dot{\sqcup\!\sqcup} v_2^1)at \subseteq w_1 \dot{\sqcup\!\sqcup} w_2$. □

Summarizing the observations of the previous propositions, it is possible to establish the following.

Theorem 2.6.30 *Let Σ be an alphabet and $L_1, L_2 \subseteq \Sigma^*$ two languages. Then the following hold:*

(i) The reduced shuffle is a sub-language of the infiltration product: $L_1 \dot{\sqcup\!\sqcup} L_2 \subseteq L_1 \uparrow L_2$.

(ii) The number of words in the reduced shuffle product is bounded from above by the number of words in the shuffle product, i.e., $\#(L_1 \dot{\sqcup\!\sqcup} L_2) \leq \#(L_1 \sqcup\!\sqcup L_2)$.

(iii) The supremum of the lengths of words in the reduced shuffle product is bounded from above by the supremum of the lengths of words in the shuffle product, i.e. $\sup\{|w| : w \in L_1 \dot{\sqcup\!\sqcup} L_2\} \leq \sup\{|w| : w \in L_1 \sqcup\!\sqcup L_2\}$.

(iv) The infimum of the lengths of words in the reduced shuffle prodcut is equal to the infimum of the lengths of words in the infiltration product, i.e. $\inf\{|w| : w \in L_1 \dot{\sqcup\!\sqcup} L_2\} = \inf\{|w| : w \in L_1 \uparrow L_2\}$.

Proof: The assertions follow, since the arguments from the propositions can be generalized linearly to languages. Point (iii), follows from the fact that the length of each word in the shuffle product is equal to the sum of the lengths of the shuffled words, while this is

is an upper bound to the length of a word in the reduced shuffle product of the same words, as identical letters might be merged. □

Combining the connection of the Hadamard product of t-wise automata and the infiltration product of words (Theorem 2.6.18), with the connection of the infiltration product and the reduced shuffle product (Theorem 2.6.30 point (iv)), we obtain the following.

Corollary 2.6.31 *Given two words $w_1, w_2 \in \Sigma^*$, any shortest word in $|\mathfrak{A}_{w_1\leq} \odot \mathfrak{A}_{w_2\leq}|$ is also accepted by $\mathfrak{A}_{w_1 ⧢ w_2}$.*

2.6.4 A Reduced Shuffle Automaton for Linear Automata

In the following we show that there exists a finite automaton $\mathfrak{A}$ accepting the reduced shuffle product of the behaviour of two given linear automata. We do so by describing (such) an automaton explicitly and show in the sequel that it has the desired behaviour.

Definition 2.6.32 *Let $\mathfrak{A}_{w_1} = \langle P, \Sigma, E_1, I_1, T_1\rangle$ and $\mathfrak{A}_{w_2} = \langle Q, \Sigma, E_2, I_2, T_2\rangle$ be linear automata, that accept w_1 and w_2 respectively, then we define $E_{w_1 ⧢ w_2}$ as follows*

$$
\begin{aligned}
E_{w_1 ⧢ w_2} \quad &:= \quad \{[(p,q),a,(p',q)] : (p,a,p') \in E_1 \wedge q \in Q\} \\
&\quad \cup \{[(p,q),a,(p,q')] : p \in P \wedge (q,a,q') \in E_2\} \\
&\quad \cup \{[(p,q),a,(p',q')] : (p,a,p') \in E_1 \wedge (q,a,q') \in E_2\} \\
&\quad \setminus \Big(\{[(p,q'),a,(p',q')] : (p,a,p') \in E_1 \wedge (q,a,q') \in E_2\} \\
&\quad \cup \{[(p',q),a,(p',q')] : (p,a,p') \in E_1 \wedge (q,a,q') \in E_2\}\Big) \\
&= E_{w_1 \uparrow w_2} \quad \setminus \Big(\{[(p,q'),a,(p',q')] : \exists (p,a,p') \in E_1 \wedge \exists (q,a,q') \in E_2\} \\
&\quad \cup \{[(p',q),a,(p',q')] : \exists (p,a,p') \in E_1 \wedge \exists (q,a,q') \in E_2\}\Big)
\end{aligned}
$$

In the following we will show that the transition matrix $E_{w_1 ⧢ w_2}$ – also referred to as the *reduced shuffle transition matrix* for w_1 and w_2 in the sequel - does indeed give rise to a reduced shuffle automaton of $\mathfrak{A}_{w_1}$ and $\mathfrak{A}_{w_2}$, see Theorem 2.6.37. However, in order to establish this result, we need several intermediate steps, with partly technical proofs. The following property will be useful in several arguments in the sequel. It reflects the recursive definition of the reduced shuffle product and the different cases induced by the occurring letters.

Lemma 2.6.33 *Let $v, w_2 \in \Sigma^*$ and j be the length of w_2. Then*

$$va \dot{\sqcup\!\sqcup} w_2 = (v \dot{\sqcup\!\sqcup} w_2) \cdot a + \sum_{\substack{k=0 \\ w_2(k-1)\neq a \\ a \notin w_2\restriction[k,j-1]}}^{j-1} (v \dot{\sqcup\!\sqcup} w_2\restriction[0,k-1]) \cdot a \cdot w_2\restriction[k,j-1] + \sum_{\substack{k=0 \\ w_2(k-1)= a \\ a \notin w_2\restriction[k,j-1]}}^{j-1} (v \dot{\sqcup\!\sqcup} w_2\restriction[0,k-2]) \cdot a \cdot w_2\restriction[k,j-1]. \tag{2.60}$$

Proof: The proof is by induction on $|w_2| = j$. For $j = 0$ we have $w_2 = \varepsilon$ and $va \dot{\sqcup\!\sqcup} \varepsilon = va$ and the sums for $k = 0$ to $k = -1$ are empty, so the right hand side of equation (2.60) is $(v \dot{\sqcup\!\sqcup} \varepsilon)a = va$.

For the induction step assume we have shown the equality in (2.60) for j and we show it for $j + 1$. We split the proof in two cases. First, assume w_2 ends with a, i.e. $w_2(j) = a$, then: $va \dot{\sqcup\!\sqcup} w_2 = (v \dot{\sqcup\!\sqcup} w_2\restriction[0, j-1]) \cdot a$ and the two sums on the right hand side of equation (2.60) are empty, because $a \in w_2\restriction[k, j]$. Second, assume $w_2(j) \neq a$: $va \dot{\sqcup\!\sqcup} w_2 = (v \dot{\sqcup\!\sqcup} w_2)a + (va \dot{\sqcup\!\sqcup} w_2\restriction[0, j-1])w_2(j)$. Applying the induction hypothesis for $w_2\restriction[0, j-1]$ to the second term, we further get:

$$\begin{aligned}(va \dot{\sqcup\!\sqcup} w_2\restriction[0,j-1])w_2(j) &= (v \dot{\sqcup\!\sqcup} w_2\restriction[0,j-1]) \cdot a \cdot w_2(j) \\ &+ \Bigg(\sum_{\substack{k=0 \\ w_2(k-1)\neq a \\ a \notin w_2\restriction[k,j-1]}}^{j-1} (v \dot{\sqcup\!\sqcup} w_2\restriction[0,k-1]) \cdot a \cdot w_2\restriction[k,j-1] \\ &+ \sum_{\substack{k=0 \\ w_2(k-1)= a \\ a \notin w_2\restriction[k,j-1]}}^{j-1} (v \dot{\sqcup\!\sqcup} w_2\restriction[0,k-2]) \cdot a \cdot w_2\restriction[k,j-1] \Bigg) \cdot w_2(j).\end{aligned}$$

The latter can easily be seen to be equal to

$$\sum_{\substack{k=0 \\ w_2(k-1)\neq a \\ a \notin w_2\restriction[k,j-1]}}^{j} (v \dot{\sqcup\!\sqcup} w_2\restriction[0,k-1]) \cdot a \cdot w_2\restriction[k,j] + \sum_{\substack{k=0 \\ w_2(k-1)= a \\ a \notin w_2\restriction[k,j]}}^{j} (v \dot{\sqcup\!\sqcup} w_2\restriction[0,k-2]) \cdot a \cdot w_2\restriction[k,j],$$

which after addition of $(v \dot{\sqcup\!\sqcup} w_2)a$ completes the proof. □

The following Lemma shows that the reduced shuffle transition matrix for prefixes of w_1 and w_2, is identical with the reduced shuffle transition matrix of w_1 and w_2, when

restricted to the corresponding set of states. The reduced shuffle transition matrix of words is compatible with restriction to prefixes, so to speak.

Lemma 2.6.34 (Compatibility with restriction to prefixes) *Let* $\mathfrak{A}_{w_1} = \langle P, \Sigma, E_{w_1}, I_1, T_1\rangle$ *and* $\mathfrak{A}_{w_2} = \langle P, \Sigma, E_{w_2}, I_2, T_2\rangle$ *be linear automata accepting* w_1 *and* w_2 *respectively (recall Definition 2.6.17), with* $Q_{w_1} = \{p_i : 0 \leq i \leq \ell_1\}$, $Q_{w_2} = \{q_j : 0 \leq j \leq \ell_2\}$. *Furthermore, let* $\mathfrak{A}_{w_1\restriction[0,k]}$ *and* $\mathfrak{A}_{w_2\restriction[0,m]}$ *be linear automata accepting* $w_1\restriction[0,k]$ *and* $w_2\restriction[0,m]$ *respectively, with* $Q_{w_1\restriction[0,k]} = \{p_i : 0 \leq i \leq k\}$, $Q_{w_2\restriction[0,k]} = \{q_j : 0 \leq j \leq k\}$. *Then for* $E_{w_1 \sqcup\!\sqcup w_2}$ *and* $E_{w_1\restriction[0,k] \sqcup\!\sqcup w_2\restriction[0,m]}$ *as described by Definition 2.6.32 it holds that*

$$E_{w_1 \sqcup\!\sqcup w_2}\restriction(\{p_i : 0 \leq i \leq k\} \times \{q_j : 0 \leq j \leq m\}) = E_{w_1\restriction[0,k] \sqcup\!\sqcup w_2\restriction[0,m]}.$$

Proof: By Definition 2.6.32 the set of transitions $E_{w_1\restriction[0,k] \sqcup\!\sqcup w_2\restriction[0,m]}$ is defined as

$$\begin{aligned} E_{w_1\restriction[0,k] \sqcup\!\sqcup w_2\restriction[0,m]} \quad = \quad & \{[(p_i, q_j), w_1(i), (p_{i+1}, q_j)] : i \leq m \wedge j \leq k\} \\ & \cup\{[(p_i, q_j), w_2(j), (p_i, q'_{j+1})] : i \leq m \wedge j \leq k\} \\ & \cup\{[(p_i, q_j), w_1(i), (p_{i+1}, q_{j+1})] : w_1(i) = w_2(j), i \leq m, j \leq k\} \\ & \setminus\Big(\{[(p_i, q_{j+1}), w_1(i), (p_{i+1}, q_{j+1})] : w_1(i) = w_2(j), i \leq m, j \leq k\} \\ & \cup\{[(p_{i+1}, q_j), w_1(i), (p_{i+1}, q_{j+1})] : w_1(i) = w_2(j), i \leq m, j \leq k\}\Big) \end{aligned}$$

and the set of transitions $E_{w_1 \sqcup\!\sqcup w_2}$ is defined as

$$\begin{aligned} E_{w_1 \sqcup\!\sqcup w_2} \quad = \quad & \{[(p_i, q_j), w_1(i), (p_{i+1}, q_j)] : i \leq \ell_1 \wedge j \leq \ell_2\} \\ & \cup\{[(p_i, q_j), w_2(j), (p_i, q'_{j+1})] : i \leq \ell_1 \wedge j \leq \ell_2\} \\ & \cup\{[(p_i, q_j), w_1(i), (p_{i+1}, q_{j+1})] : w_1(i) = w_2(j), i \leq \ell_1, j \leq \ell_2\} \\ & \setminus\Big(\{[(p_i, q_{j+1}), w_1(i), (p_{i+1}, q_{j+1})] : w_1(i) = w_2(j), i \leq \ell_1, j \leq \ell_2\} \\ & \cup\{[(p_{i+1}, q_j), w_1(i), (p_{i+1}, q_{j+1})] : w_1(i) = w_2(j), i \leq m, j \leq \ell_2\}\Big) \end{aligned}$$

Therefore, considering the transition matrices we have that

$$(E_{w_1 \sqcup\!\sqcup w_2})_{(p_{i_0}, q_{j_0}),(p_{i_1}, q_{j_1})} = \Big(E_{w_1\restriction[0,k] \sqcup\!\sqcup w_2\restriction[0,m]}\Big)_{(p_{i_0}, q_{j_0}),(p_{i_1}, q_{j_1})}$$

for $i_0, i_1 \leq k$ and $j_0, j_1 \leq m$.

Since $Q_{w_1\restriction[0,k]} = \{p_i : 0 \leq i \leq k\}$ and $Q_{w_2\restriction[0,m]} = \{q_j : 0 \leq j \leq m\}$ it follows that $E_{w_1 \sqcup\!\sqcup w_2}\restriction(\{p_i : 0 \leq i \leq k\} \times \{q_j : 0 \leq j \leq m\}) = E_{w_1\restriction[0,k] \sqcup\!\sqcup w_2\restriction[0,m]}$. □

The following lemma establishes the essential connection between $E_{v \sqcup\!\sqcup w_2}$ and $E_{va \sqcup\!\sqcup w_2}$.

Lemma 2.6.35 (Structure of $E_{va⧢w_2}$) *Let $w_1, w_2, v \in \Sigma^*$, $a \in \Sigma$ with $w_1 = va$, and let ℓ_1 be the length of v, ℓ_2 be the length of w_2. Further assume that $\mathfrak{A}_{v⧢w_2}$ is an automaton with transition matrix $E_{v⧢w_2}$ as described by Definition 2.6.32. Then for the automaton $\mathfrak{A}_{w_1⧢w_2} = \langle P \times Q, \Sigma, E_{va⧢w_2}, I = (p_0, q_0), T = (p_{\ell_1+1}, q_{\ell_2})\rangle$, where $P = \{p_0, \ldots, p_{\ell_1+1}\}$ and $Q = \{q_0, \ldots, q_{\ell_2}\}$, and $E_{va⧢w_2}$ again as described by Definition 2.6.32, the transition matrix $E_{va⧢w_2}$ can be represented as follows:*

$$E_{va⧢w_2} = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right), \text{ where:}$$

- $A = E_{v⧢w_2}$
- $B = \left(\begin{array}{c} \mathbf{0} \\ \hline \ldots \\ \hline \mathbf{0} \\ \hline \mathbf{a}' \end{array}\right)$ *consists of $\ell_1 + 1$ blocks $\mathbf{0}$ of size $(\ell_2 + 1) \times (\ell_2 + 1)$ with all entries 0 and one block $\mathbf{a}'$ of size $(\ell_2 + 1) \times (\ell_2 + 1)$ where for $0 \leq i, j \leq \ell_2$*

$$\mathbf{a}'(i,j) = \begin{cases} a \text{ if } i = j = 0 \\ a \text{ if } i = j > 0 \wedge w_2(j-1) \neq a \\ a \text{ if } i = j - 1 \geq 0 \wedge w_2(j-1) = a \\ 0 \text{ otherwise.} \end{cases}$$

- *C is the $(\ell_2 + 1) \times ((\ell_1 + 1) \cdot (\ell_2 + 1))$ 0-matrix.*
- *D is a $(\ell_2+1)\times(\ell_2+1)$ matrix with $D_{q_i,q_j} = \begin{cases} w_2(i) & , \text{ if } i+1 = j \text{ and } w_2(i) \neq a, \\ 0 & , \text{ otherwise.} \end{cases}$*

  In other words $D = E_{w_2} \setminus a$.

Proof: By Lemma 2.6.34 A is the matrix which describes transformations from (p_i, q_j) to (p_k, q_m) with $0 \leq i, k \leq \ell_1$ and $0 \leq j, m \leq \ell_2$, as claimed above.

B is the matrix which describes transformations from (p_i, q_j) to (p_{ℓ_1+1}, q_k) with $0 \leq i \leq \ell_1$, $0 \leq j, k \leq \ell_2 + 1$. By definition of $\mathfrak{A}_{va⧢w_2}$ there are no transformations from (p_i, q_j) to (p_{ℓ_1+1}, q_k) if $i < \ell_1$, therefore the first ℓ_1 blocks are 0. For p_{ℓ_1} and p_{ℓ_1+1} since $(p_{\ell_1}, a, p_{\ell_1+1}) \in E_{va}$ we have by definition of $E_{va⧢w_2}$, for $0 \leq j \leq \ell_2$:

- $((p_{\ell_1}, q_0), a, (p_{\ell_1+1}, q_0)) \in E_{va⧢w_2}$,
- $((p_{\ell_1}, q_j), a, (p_{\ell_1+1}, q_j)) \in E_{va⧢w_2}$ if $(q_{j-1}, a, q_j) \notin E_{w_2}$,
- $((p_{\ell_1}, q_j), a, (p_{\ell_1+1}, q_{j+1})) \in E_{va⧢w_2}$ if $(q_{j-1}, a, q_j) \in E_{w_2}$.

C is the matrix which describes transformations from (p_{ℓ_1+1}, q_j) to (p_i, q_k) with $0 \leq i \leq \ell_1$ and $0 \leq j, k \leq \ell_2$. By definition of $\mathfrak{A}_{va \sqcup\!\sqcup w_2}$ there are no transformations which reduce the index of p_j, therefore C contains only 0 entries.

Finally, D is the matrix which describes transformations from (p_{ℓ_1+1}, q_j) to (p_{ℓ_1+1}, q_k) with $0 \leq j, k \leq \ell_2$. By definition of $E_{va \sqcup\!\sqcup w_2}$ these are the transformations $((p_{\ell_1+1}, q_j), x, (p_{\ell_1+1}, q_{j+1}))$ for which there is a transformation (q_j, x, q_{j+1}) in $\mathfrak{A}_2$ and no transformation $(p_{\ell_1}, w_2(j), p_{\ell_1+1})$ in $\mathfrak{A}_1$, which holds for $x = w_2(j)$, as $w_1(\ell_1 + 1) \neq w_2(j)$. □

Lemma 2.6.36 (Star of $E_{va \sqcup\!\sqcup w_2}$) *Let $w_1, w_2, v \in \Sigma^*$, $a \in \Sigma$ with $w_1 = va$, further let ℓ_1 be the length of v, ℓ_2 be the length of w_2. We consider $\mathfrak{A}_{v \sqcup\!\sqcup w_2}$ with transition matrix $E_{v \sqcup\!\sqcup w_2}$ as given in Definition 2.6.32 and assume further that $(E^*_{v \sqcup\!\sqcup w_2})_{(p_0,q_0),(p_{\ell_1},q_j)} = v \dot{\sqcup\!\sqcup} w_2{\restriction}[0, j-1]$ for every $j \leq \ell_2$. Then for the automaton $\mathfrak{A}_{w_1 \sqcup\!\sqcup w_2} = \langle P \times Q, \Sigma, E_{va \sqcup\!\sqcup w_2}, I, T \rangle$, where $P = \{p_0, \ldots, p_{\ell_1+1}\}$, $Q = \{q_0, \ldots, q_{\ell_2}\}$, $I = (p_0, q_0)$ and $T = (p_{\ell_1+1}, q_{\ell_2})$ it holds that:* $E^*_{va \sqcup\!\sqcup w_2} = \left(\begin{array}{c|c} A^* & F^* \\ \hline 0 & D^* \end{array}\right)$ *where F^* is an $((\ell_1 + 1) \cdot (\ell_2 + 1)) \times (\ell_2 + 1)$-matrix with*

$$F^*_{(p_0,q_0),q_j} = va \dot{\sqcup\!\sqcup} (w_2{\restriction}[0, j-1]) \text{ for all } j \in \{0, \ldots, \ell_2\}.$$

In particular, for $j = \ell_2$ we have $F^*_{(p_0,q_0),q_{\ell_2}} = E^*_{va \sqcup\!\sqcup w_2}((p_0, q_0), (p_{\ell_1+1}, q_{\ell_2})) = va \sqcup\!\sqcup w_2$.

Proof: By Lemma 2.6.35 we know that $E_{va \sqcup\!\sqcup w_2} = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$ with C containing only 0 entries, therefore, by Theorem 2.6.9 we get that $E^*_{va \sqcup\!\sqcup w_2} = \left(\begin{array}{c|c} A^* & A^* \cdot B \cdot D^* \\ \hline 0 & D^* \end{array}\right)$, so $F^* = A^* \cdot B \cdot D^*$.

To decide $(A^* \cdot B \cdot D^*)_{(p_0,q_0),q_j}$ first note that by Lemma 2.6.13

$$D^*_{q_i,q_j} = \begin{cases} \varepsilon \text{ if } i = j \\ w_2{\restriction}[i, j-1] \text{ if } w_2{\restriction}[i, j-1] \text{ does not contain } a \\ 0 \text{ otherwise} \end{cases}.$$

Let us now look at $(A^* \cdot B \cdot D^*)_{(p_0,q_0)}$. So, let $q_c \in Q$ be arbitrary, then $(A^* \cdot B \cdot D^*)_{(p_0,q_0),q_c} = \sum_{q \in Q} (A^* \cdot B)_{(p_0,q_0),q} \cdot D^*_{q,q_c}$. Focusing on the first term, we further get

$$\begin{aligned} (A^* \cdot B)_{(p_0,q_0),q} &= (A^*_{p_0} \cdot B)_{q_0,q} \\ &= \sum_{(p',q') \in P \times Q} A^*_{(p_0,q_0),(p',q')} B_{(p',q'),q} \\ &= \sum_{p' \in P} \sum_{q' \in Q} A^*_{(p_0,q_0),(p',q')} B_{(p',q'),q} \\ &= \sum_{q' \in Q} A^*_{(p_0,q_0),(p_{\ell_1},q')} \mathbf{a}'_{q',q}. \end{aligned}$$

Where the last identity holds since, $B_{(p',q'),q} = 0$ for $p' \neq p_{\ell_1}$, and $B_{(p_{\ell_1},q'),q} = \mathbf{a}'_{q',q}$. Since further, by assumption for $q' = q_j$ we have $A^*_{(p_0,q_0),(p_{\ell_1},q_j)} = v \dot{\sqcup} w_2 \restriction [0, j-1]$, we get

$$(A^* \cdot B)_{(p_0,q_0),q} \quad = \quad \sum_{j=0}^{\ell_2} (v \dot{\sqcup} (w_2 \restriction [0, j-1])) \cdot \mathbf{a}'_{q_j,q}$$

Now, using that $\mathbf{a}'_{q_j,q} = \begin{cases} a \text{ if } q = q_j, j = 0 \\ a \text{ if } q = q_j, j > 0 \wedge w_2(j-1) \neq a \\ a \text{ if } q = q_{j+1}, j \geq 0 \wedge w_2(j) = a \\ 0 \text{ otherwise.} \end{cases}$, we get

$$(v \dot{\sqcup} (w_2 \restriction [0, j-1])) \cdot \mathbf{a}'_{q_j,q} = \begin{cases} (v \dot{\sqcup} (w_2 \restriction [0, -1]))a = va \text{ if } q = q_j, j = 0, \\ (v \dot{\sqcup} w_2 \restriction [0, j-1])a \text{ if } q = q_j, j > 0, w_2(j-1) \neq a, \\ (v \dot{\sqcup} w_2 \restriction [0, j-1])a \text{ if } q = q_{j+1}, j \geq 0, w_2(j) = a, \\ 0, \text{ otherwise.} \end{cases} \tag{2.61}$$

Note that for $j > 0$, by shifting j to $j-1$, the third case can be reformulated to $(v \dot{\sqcup} w_2 \restriction [0, j-2]) \cdot \mathbf{a}'_{q_{j-1},q} = (v \dot{\sqcup} w_2 \restriction [0, j-2]) \cdot a$ if $q = q_j$ and $w_2(j-1) = a$. Additionally, the first case above appears as the special case for $j = 0$ of the second case in (2.61). Thus we get

$$(A^* \cdot B)_{(p_0,q_0),q} \quad = \quad \sum_{j=0, w_2(j-1) \neq a}^{\ell_2} (v \dot{\sqcup} w_2 \restriction [1, j-1]) \cdot a + \sum_{j=0, w_2(j-1)=a}^{\ell_2} (v \dot{\sqcup} w_2 \restriction [1, j-2]) \cdot a$$

Hence we get that

$$\begin{aligned} (A^* \cdot B \cdot D^*)_{(p_0,q_0),q_c} \quad &= \quad \sum_{j=0}^{\ell_2} (A^* \cdot B)_{(p_0,q_0),q_j} \cdot D^*_{q_j,q_c} \\ &= \quad \sum_{j=0, w_2(j-1) \neq a}^{\ell_2} (v \dot{\sqcup} w_2 \restriction [1, j-1]) \cdot a \cdot D^*_{q_j,q_c} \\ &+ \quad \sum_{j=0, w_2(j-1)=a}^{\ell_2} (v \dot{\sqcup} w_2 \restriction [1, j-2]) \cdot a \cdot D^*_{q_j,q_c}. \end{aligned}$$

Now consider two cases: First, assume $w_2(c-1) = a$. It follows that $D^*_{q_j,q_c} = 0$ whenever $j \neq c$ and $D^*_{q_c,q_c} = 1$, so the sum above has only one non-zero term: $(v \dot{\sqcup} w_2 \restriction [0, c-2]) \cdot a$, as desired in the statement.

Second, assume $w_2(c-1) \neq a$, then we have

$$\begin{aligned}(A^* \cdot B \cdot D^*)_{(p_0,q_0),q_c} &= \sum_{\substack{j=0 \\ w_2(j-1)\neq a}}^{\ell_2} (v \dot{\sqcup} w_2 \restriction [0, j-1]) \cdot a \cdot D^*_{q_j,q_c} \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)=a}}^{\ell_2} (v \dot{\sqcup} w_2 \restriction [0, j-2]) \cdot a \cdot D^*_{q_j,q_c},\end{aligned}$$

and since $D^*_{q_j,q_c} = 0$ for $j > c$, and $D^*_{q_c,q_c} = 1$ we further have:

$$\begin{aligned}(A^* \cdot B \cdot D^*)_{(p_0,q_0),q_c} &= (v \dot{\sqcup} (w_2 \restriction [0, c-1])) \cdot a \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)\neq a \\ a \notin w_2 \restriction [j,c-1]}}^{c-1} (v \dot{\sqcup} w_2 \restriction [0, j-1]) \cdot a \cdot w_2 \restriction [j, c-1] \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)= a \\ a \notin w_2 \restriction [j,c-1]}}^{c-1} (v \dot{\sqcup} w_2 \restriction [0, j-2]) \cdot a \cdot w_2 \restriction [j, c-1].\end{aligned} \tag{2.62}$$

Now we split again in two cases: First assume $w_2(c-2) \neq a$: In this case, the above sum in (2.62) is equal to

$$\begin{aligned}&(v \dot{\sqcup} w_2 \restriction [0, c-1]) \cdot a + (v \dot{\sqcup} w_2 \restriction [0, c-2]) \cdot a \cdot w_2(c-1) \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)\neq a \\ a \notin w_2 \restriction [j,c-2]}}^{c-2} (v \dot{\sqcup} w_2 \restriction [0, j-1]) \cdot a \cdot w_2 \restriction [j, c-2] \cdot w_2(c-1) \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)= a \\ a \notin w_2 \restriction [j,c-2]}}^{c-2} (v \dot{\sqcup} w_2 \restriction [0, j-2]) \cdot a \cdot w_2 \restriction [j, c-2] \cdot w_2(c-1) \\ &= (v \dot{\sqcup} w_2 \restriction [0, c-1]) \cdot a + \Big((v \dot{\sqcup} w_2 \restriction [0, c-2]) \cdot a \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)\neq a \\ a \notin w_2 \restriction [j,c-2]}}^{c-2} (v \dot{\sqcup} w_2 \restriction [0, j-1]) \cdot a \cdot w_2 \restriction [j, c-2] \\ &+ \sum_{\substack{j=0 \\ w_2(j-1)= a \\ a \notin w_2 \restriction [j,c-2]}}^{c-2} (v \dot{\sqcup} w_2 \restriction [0, j-2]) \cdot a \cdot w_2 \restriction [j, c-2] \Big) \cdot w_2(c-1)\end{aligned}$$

Using Lemma 2.6.33 the last sum is equal to $(v \dot{\sqcup} w_2 \restriction [0, c-1]) \cdot a + (va \dot{\sqcup} w_2 \restriction [0, c-2]) w_2(c-1) = va \dot{\sqcup} w_2 \restriction [0, c-1]$, as desired.

Now assume $w_2(c-2) = a$: In this case, the sum in (2.62) is equal to

$$\begin{aligned}
& (v \dot{\sqcup} w_2{\restriction}[0,c-1]) \cdot a + (v \dot{\sqcup} w_2{\restriction}[0,c-3]) \cdot a \cdot w_2(c-1) \\
+ & \sum_{\substack{j=0 \\ w_2(j-1)\neq a \\ a \notin w_2{\restriction}[j,c-2]}}^{c-2} (v \dot{\sqcup} w_2{\restriction}[0,j-1]) \cdot a \cdot w_2{\restriction}[j,c-2] \cdot w_2(c-1) \\
+ & \sum_{\substack{j=0 \\ w_2(j-1)= a \\ a \notin w_2{\restriction}[j,c-2]}}^{c-2} (v \dot{\sqcup} w_2{\restriction}[0,j-2]) \cdot a \cdot w_2{\restriction}[j,c-2] \cdot w_2(c-1) \\
= & \quad (v \dot{\sqcup} w_2{\restriction}[0,c-1]) \cdot a + \Big((v \dot{\sqcup} w_2{\restriction}[0,c-3]) \cdot a \\
+ & \sum_{\substack{j=0 \\ w_2(j-1)\neq a \\ a \notin w_2{\restriction}[j,c-2]}}^{c-2} (v \dot{\sqcup} w_2{\restriction}[0,j-1]) \cdot a \cdot w_2{\restriction}[j,c-2] \\
+ & \sum_{\substack{j=0 \\ w_2(j-1)= a \\ a \notin w_2{\restriction}[j,c-2]}}^{c-2} (v \dot{\sqcup} w_2{\restriction}[0,j-2]) \cdot a \cdot w_2{\restriction}[j,c-2] \Big) \cdot w_2(c-1)
\end{aligned}$$

Using Lemma 2.6.33, for $w_2{\restriction}[0,c-1]$ of length $c-1$ and the fact that $w_2{\restriction}[c-1,c-2] = \varepsilon$, the last sum is equal to $(v \dot{\sqcup} w_2{\restriction}[0,c-1])a + (va \dot{\sqcup} w_2{\restriction}[0,c-2])w_2(c-1) = va \dot{\sqcup} w_2{\restriction}[0,c-1]$, as desired. □

We are now in position to establish that the transition matrix $E_{w_1 \dot{\sqcup} w_2}$ as defined in Definition 2.6.32 does yield an automaton accepting the reduced shuffle products of two words w_1, w_2.

Theorem 2.6.37 *Let $\mathfrak{A}_{w_1} = \langle P, \Sigma, E_1, I_1, T_1 \rangle$ and $\mathfrak{A}_{w_2} = \langle Q, \Sigma, E_2, I_2, T_2 \rangle$ be linear automata, that accept w_1 and w_2 respectively, then*

$$\mathfrak{A}_{w_1 \dot{\sqcup} w_2} := \langle P \times Q, \Sigma, E_{w_1 \dot{\sqcup} w_2}, I_1 \times I_2, T_1 \times T_2 \rangle \tag{2.63}$$

has the behaviour $|\mathfrak{A}_{w_1 \dot{\sqcup} w_2}| = |\mathfrak{A}_1| \dot{\sqcup} |\mathfrak{A}_2|$, i.e. $\mathfrak{A}_{w_1 \dot{\sqcup} w_2}$ accepts $w_1 \dot{\sqcup} w_2$,.

Proof: Let again $|w_1| = \ell_1$ and $|w_2| = \ell_2$. We show by induction on the pre-fixes of w_1, i.e. for $-1 \leq k < \ell_1$, that $|\mathfrak{A}_{w_1{\restriction}[0,k] \dot{\sqcup} w_2}| = w_1{\restriction}[0,k] \dot{\sqcup} w_2$.

For the induction basis $k = -1$, we have $w_1{\restriction}[0,k] = \varepsilon$. So $w_1{\restriction}[0,k] \dot{\sqcup} w_2 = w_2$. The automaton $\mathfrak{A}_{w_1{\restriction}[0,k]}$ consists of $P = \{p_0\}$, $E = \emptyset$ and $I = T = \{p_0\}$. So by Definition 2.6.32 $E_{w_1{\restriction}[0,k] \dot{\sqcup} w_2} = \{((p_0,q),a,(p_0,q')) : (q,a,q') \in E_{w_2}\}$, $P \times Q = \{p_0\} \times Q$, $I \times I_2 = \{p_0\} \times I_2$ and $T \times T_2 = \{p_0\} \times T_2$. It is easy to see that $\mathfrak{A}_2$ is equivalent to $\mathfrak{A}_{w_1{\restriction}[0,k] \dot{\sqcup} w_2}$ and hence $|\mathfrak{A}_{w_1{\restriction}[0,k] \dot{\sqcup} w_2}| = |\mathfrak{A}_2| = \{w_2\} = \{w_1{\restriction}[0,k] \dot{\sqcup} w_2\}$. Now assume the induction hypothesis holds for all k and we show that it holds for $k+1$.

For the induction step, we use Lemma 2.6.36 for $v = w_1{\restriction}[0,k]$ and $a = w_1(k+1)$, to get that $(E^*_{w_1{\restriction}[0,k+1]⧢w_2})_{(p_0,q_0),(p_{k+2},q_{\ell_2})} = w_1{\restriction}[0,k+1]⧢w_2$. Since (p_0,q_0) is the only initial state of $\mathfrak{A}_{w_1{\restriction}[0,k+1]⧢w_2}$ and (p_{k+2},q_{ℓ_2}) its only final state, we get that $|\mathfrak{A}_{w_1{\restriction}[0,k+1]⧢w_2}| = \{w_1{\restriction}[0,k+1]⧢w_2\}$. Since $w_1{\restriction}[0,\ell_1-1] = w_1$ this finishes the proof. □

Concluding Remarks

The sequential combinatorial test sets considered in [122] and in [123] are largely equivalent, deviating only in the underlying formal concept of labeled transition systems, respectively Mealy machines used to model the SUT. However, we may assume that the SUT is modeled by means of a finite labeled transition system, respectively that the SUTs output alphabet does not influence test sequence generation. Then we can define sequential combinatorial test sets that achieve full t-way sequence coverage in the sense of [122] and in [123], using the notions and notation introduced below in Subsection 2.6.1, as follows.

Definition 2.6.38 *For a given finite automaton $\mathfrak{A} = \langle Q,\Sigma,E,I,T\rangle$ that accepts a language $\mathcal{L}$, a* sequential combinatorial test sets that achieves full t-way sequence coverage *is a set of words $\mathcal{S} \subseteq \mathcal{L}$ accepted by $\mathfrak{A}$ with the property that for all $s \in \Sigma^t$ which appear as subsequence of some word accepted by $\mathfrak{A}$, there exists a word in $\mathcal{S}$ that has s as subsequence.*

To conclude the thoughts outlined in Remark 2.6.22 and the connection of the reduced shuffle product to sequential CT, Corollary 2.6.31 shows how the reduced shuffle product of linear automata can be used instead of the Hadamard product of t-wise automata for the constructions of sequential combinatorial test sets. We can replace the Hadamard products in algorithms as described in the literature [123, Algorithm 1 & 2], by the respective reduced shuffle products, to reduce the size (i.e. number of transitions and possibly states) of the appearing automata, without losing computability of the shortest test sequence. Thereby we could start by constructing a reduced shuffle product automaton for words of length t, and the derivation of an accepted word, covering both t-way sequences. This way, it is also not necessary to introduce loops on the states. Iteratively, this word may then used in a reduced shuffle product with further words of length t. Construction of the reduced shuffle product automaton for given linear automata is described by Definition 2.6.32 together with Theorem 2.6.37 described as presented above, while the construction of such an automaton for arbitrary give automata is studied in [140].

Towards applications, as future work it would be interesting to use the reduced shuffle product for the derivation of combinatorial test sets achieving full t-way sequence coverage. Regarding theoretical aspects, it would be interesting to investigate alternative notions of reduced shuffle products, with respect to associativity or with respect to yielding even more compact product automata.

2.7 Summary

In this chapter we have first surveyed the the results of this thesis together with the current state of the art on complexity problems for covering arrays in Section 2.2. A condensed summary of the results is provided by the diagram in Figure 2.2 and Table 2.1. These make apparent that as soon as we give up structure in terms of the column selections that can appear as input to problems related to CAs, it was possible for researchers to prove some hardness results. However, especially for CAs and MCAs, further research is needed in order to prove or disprove statements about the computational hardness of the respective decision or search problems.

Mapping the optimal covering arrays generation problem to the minimal set cover problem was briefly reviewed in Section 2.3, where also a weighted density notion was described, which can be used in an algorithm descending from a generalization of this mapping. We also showed that such a heuristic algorithm can be used for computing VCAs, where for some instances optimal VCAs were found.

Thereafter, in Section 2.4 a way to model CAs of arbitrary strengths as solutions of multivariate polynomial equation systems was presented. Some initial equation systems were solved with algebraic solvers based on Gröbner bases or search techniques supported by supercomputing. Even though the presented modeling is only considered for CAs over v-ary alphabets, a generalization for other classes of covering arrays (e.g. mixed level or variable strength) might be possible.

In Section 2.5 we introduced the notion of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs, which appear as intersections of CAs and PAs. The classification results for balanced CAs can be understood as a dissection of classes of CAs regarding the appearance of i-way interactions for $i \leq t$. We formulated first observations for the balance vectors $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ and proved bounds for them which can tell for which balance vectors it is relevant to consider the respective classes of balanced CAs. These bounds can be used to reduce the number of balance vectors considered in the computational search performing the classification of non-equivalent balanced CAs. The obtained classification results were put into context with existing results from the literature. The computational results, together with the bounds on $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ allowed us to determine $\mathsf{CAK}(17; 3, 2) = 16$, and consequentially $\mathsf{CAN}(3, k, 2) = 18$ for $k \in \{17, 18, 19, 20\}$. We further highlighted some observations that point towards $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs having a rich structure and an independent existence from Orthogonal Arrays, allowing separate and dedicated treatment. As part of such, one of the most intriguing questions would be: *Given parameters N, t, k and v, what are the strongest balance vectors $\boldsymbol{\lambda}$, $\boldsymbol{y}$ for which a $\mathsf{CA}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}(N; t, k, v)$ exists?*

Finally, in Section 2.6 we introduced and investigated a *reduced-shuffle* product of words, respectively of languages, that reflects the needs of finding short test sequences that contain given sequences, when designing sequential combinatorial test sets for testing SUTs that are represented as finite automata. When modeling event-driven (software) systems by finite automata, products of automata play a special role, as they can model the composition of systems in different ways. The introduced *reduced-shuffle* product, can be

interpreted as interleaved composition of systems with *enforced synchronization* whenever possible. For the purposes of sequential combinatorial test set generation the *reduced-shuffle* can be better suited than existing product operators, like the *shuffle* product, representing the interleaved composition *without any synchronization*, and the *infiltration* product, representing the interleaved composition with *optional synchronization*. We further showed that the infiltration of words corresponds to the Hadamard product of t-wise automata as considered in [123]. Consequentially, the reduced-shuffle product provides an alternative to the Hadamard product of automata as used in [123]. By virtue of its inherent deletion operation, it allows to derive shorter words from the resulting reduced-shuffle product automaton, when compared to a Hadamard product automaton. However, the generation of sequential combinatorial test sets analogue to the approach proposed in [123] remains subject to future work, as well as the study of formal aspects of the reduced-shuffle product, for example to investigate connections to existing products of automata in combination with deletion operators.

CHAPTER 3

Algorithmic Techniques for Covering Array Generation

The third chapter of this thesis pertains to the algorithmic generation of covering arrays. We have seen in the previous chapter that the computational complexity of the optimal CA (in the narrow sense) generation problem remains unknown, and that closely related problems are computationally hard. Informally speaking, one can certainly call the optimal CA generation problem, as well as related problems, a challenging task. Consequently, this raises the question for heuristic generation of non-optimal, but optimized CAs with the objective of minimizing their number of rows. In a wider sense, this may include mathematical or combinatorial constructions, greedy heuristics, metaheuristics, or randomized algorithms.

The argument that *optimized* CA generation is a difficult problem to solve is underlined by the observation that the current state of the art with regards to upper bounds on CANs (i.e., the number of rows of optimal CAs) is provided by a plurality of algorithms and constructions. The best-known upper bounds on CAN are recorded in [97], with the respective source achieving the result indicated next to the bound. An extensive literature survey of CA generation methods is provided by [98], including mathematical, recursive, exact, greedy heuristic, and metaheuristic methods.

The intention of this chapter is to pioneer the usage of novel approaches for the generation of CAs, which may inspire the development of further methods that can contribute to optimized combinatorial test set generation. Further, we want to provide the means to extend the usage of existing methods for CA generation so that their practical applicability allows to generate CAs for larger CA instances. In other words, the goals of this chapter are

- to enrich the collection of CA generation approaches with new methods, and
- to extend the usage of existing CA generation methods.

Therefore, in the following sections we will examine two novel, previously unexplored algorithmic approaches for CA generation. One of them is – to the best of my knowledge – the first approach for CA generation that makes use of artificial neural networks. The other one is a hybrid algorithm combining a the well-known IPO greedy heuristic for CA generation [174] with a quantum-inspired algorithm [245]. We will also present a memory-efficient variant of a well-known greedy algorithm for CA generation, making use of a simple combinatorial construction to reduce the memory usage of an underlying data structure. We start with a small overview of work related to CA generation and artificial neural networks.

3.1 Related Literature

As is also mentioned in [98] different CA generation methods have different features, capabilities and limitations. Depending on the given parameters of a CA instance, some methods are more or less promising for the generation of a *small* CA, i.e. one with a small number of rows for given parameters t, k and v. This is also the reason for the plurality of methods representing the state of the art recorded at [97]. We expand on this thought by means of some examples, but a holistic treatment of this matter clearly goes beyond the scope of this thesis. We refer the interested reader to the literature [98, Section VIII], where also an idea of the terms *small, medium* and *large* with regards to CA generation problems is given.

For example, *mathematical construction* can be very fast, provided that they require little or no search at all to be performed. In some cases, e.g. [101], the result will even be an optimal CA. However, often mathematical constructions have the downside that they can only be applied for specific CA parameters t, k or v instances, e.g., the constructions in [101] and [246] only work for alphabets with the size of a prime-power. *Exact methods* based on SAT solving [69], or integer programming [153] can return optimal solutions, however they usually only work for small CA instances, as they usually rely on full enumeration, or at least, traversing large parts of their search spaces, which suffer from combinatorial explosion. A further example are *metaheuristic methods*, which are often well suited for small to medium sized CA problems, and, provided an appropriate model, they can be applied for versatile parameter settings due to the generality of their formulation. A downside of these methods is that they may be inapplicable to large CA generation problems, due to their long execution time. Depending on the algorithm and the problem encoding, it may be not even possible to return a CA that is far from optimal, but still covers the desired t-way interactions, can be returned. In some cases metaheuristic methods are combined with combinatorial constructions to hybridized algorithms. For example, the authors of [113] present an approach for CA construction using *augmented annealing*, which combines simulated annealing with recursive combinatorial constructions combining CAs with difference covering arrays or Hadamard matrices. Such approaches can benefit from the generality of heuristic searches and the scalability of combinatorial constructions. As fourth and last example we mention *heuristic algorithms* such as IPO [174], DDA [71] or AETG [36], which are

often fast and return non-optimal, but reasonably sized CAs. Notable, DDA is the first polynomial time (polynomial in k for fixed t and v) algorithm that can generate CAs that satisfy the logarithmic bound, i.e. with $N \in O(\log k)$ rows [71]. Heuristic methods can be adopted or applied to many CA generation problems, such as CAs with constraints, CAs over mixed alphabet sizes or variable-strength CAs, which is one reason why they are often underlying CA generation tools. For example IPO algorithms are underlying [117] and [78], to name two of the most prominent CA generation tools.

As outlined above, various approaches have been followed for CA generation, however, to the best of my knowledge, there does not exist any report on generation of covering arrays based on artificial neural networks – despite the recent increased attention for these methods.

3.2 Artificial Neural Networks for Covering Array Generation

In this section we demonstrate that Artificial Neural Networks (ANNs) can be applied for CA construction. To conduct our studies, we rely on existing connections in-between combinatorial optimization problems as well as on applications of neural networks to combinatorial optimization problems. More explicit, we make use of the mapping of the CA generation problem to the set cover problem, as reviewed in Section 2.3 and as presented [144], in combination with neural network approaches for solving set cover problems, presented in [247] and [248]. A combination of these works together with some extensions, in terms of learning cycles and additional necessary adjustments to the CA generation problem, make possible this first study of neural networks for CA generation.

The work presented in this chapter is based on the publication [142] which is an extension of a conference version [141]. We examine the generation of CAs via Boltzmann neural networks and Hopfield neural networks. The starting point for both algorithms is the mapping of the CA generation problem to SC problems, as reviewed in Section 2.3. For the algorithm using Boltzmann neural networks, we treat different versions, we first consider more problem specific weight assignments. Second, we consider a version with an additional feedback loop that modifies the weights of a Boltzmann machine and third, one variant that employs learning techniques, allowing the network to undergo structural changes to adjust the various connections encountered in the graph representation of the considered instances.

We start by providing some background to artificial neural networks and combinatorial optimization, and thereafter give the necessary notions with regard to artificial neural networks used in this section. Then, we show how an existing algorithm using Boltzmann machines can be used for CA generation, which is improved later by integrating various learning techniques. Next, we present Hopfield neural networks for the construction of CAs. Finally, we provide an experimental evaluation of the presented approaches and showcase that the developed concepts are also applicable for generalizations of CAs, namely mixed-level and constrained CAs.

3.2.1 Background to Artificial Neural Networks and Combinatorial Optimization

Since their first introduction in the 1940s, e.g. by McCulloch and Pitts, artificial neural networks (ANNs) have been studied in several phases by many researchers. They have been successfully applied in various fields of computer science, solving problems of prediction, classification and pattern recognition, while being less successful when applied to optimization problems, as stated in [249]. The author of [250] argues towards the capabilities of ANNs, as they can be used to tackle many different types of combinatorial optimization problems, as many of these approaches result in competitive solutions compared to alternative solution techniques in terms of solution quality. Further, for many years researchers were forced to simulate the behaviour of neural networks on digital computers while awaiting the development of specific hardware advances. According to [250], such simulations can only evaluate the potential of neural networks in terms of generating near-optimal solutions to combinatorial optimization problems and thus result in large CPU times that are uncompetitive with alternative techniques. More recent progress shows the advancement of dedicated processors, like TPUs [251], for implementations of ANNs, or even the development of alternative computing concepts for neural network implementations, see [252]. Recently, significant effort has been spent to replicate the decisions of human experts using artificial intelligence [253]. However, despite recent emphasis on ANNs and ANN computation, existing work [254] shows that also neural networks have their limitations when applied to complex problems.

There exist several works related to the application of ANNs for combinatorial optimization, see [255] or [250] for an overview. Related to *exact* combinatorial optimization problems, we want to mention the work in [256], which presents Boltzmann machines for solving the n-Queen problem for $n = 1000$ as well as Polyomino Puzzles, representing combinatorial optimization problems. More recent work [257], proposes deep learning methods for learning algorithms for *non-exact* combinatorial optimization problems (Minimum Vertex Cover, Maximum Cut and Traveling Salesman Problem) using reinforcement learning and graph embedding. The presented results are very promising, even when considering the generalization of the learned algorithms from smaller to larger graphs, i.e. from 100 vertices up to 1200 vertices. Compared to existing combinatorial optimization works, the current research on ANNs for generating combinatorial designs seems rather limited. We want to mention an application of ANNs for the generation of *exact* combinatorial designs. In [258] the authors present an ANN model for Sudoku-solving that is based on a *Q'tron* model, which represents an extended version of the Hopfield model and also relies on energy minimization. Although generalizable, their model is developed along side $(3, 3)$-*Sudoku latin square*, i.e. the popular 9×9 Sudoku puzzle. Compared to that, in the present paper we target the generation of (mixed-level) covering arrays, a *non-exact* combinatorial design, and generate these for a range of different sizes respectively different instances.

3.2.2 Preliminaries to Artificial Neural Networks and Graphs

In this section we give the definitions needed in this section. Related notions for neural networks are taken from [259] and [260], those for graphs can be also found in [261].

Artificial neural networks are entities of several components. Depending on the type of the network several individual notions are needed to describe the network and its behaviour. Common to ANNs are the notions of *units* or *neurons*, *connections* or *synapses* and *connection weights*.

These underlying structures can be formally understood as an undirected[1] weighted graph $G = (V, E)$ with edge-weights $\omega(E) = \{w_e | e \in E\}$. The units are represented by the vertices V of the graph G, connections between neurons map to the edges E and the connection weights to the edge-weights $\omega(E)$, representing the strength of the respective synaptic connection. Based on these generic definitions, we introduce the considered networks in this section below. The first, being Hopfield networks, are recurrent neural networks, with symmetrical connections, where all units are input and output units, see also [260].

Definition 3.2.1 *A* Hopfield neural network *is a deterministic, recurrent neural network with an underlying undirected weighted complete graph $G = (I, E)$, i.e. $E = I \times I$, with weights $\omega(E) = \{w_{i,j} | \{i, j\} \in E\}$, with $w_{i,i} = 0$. For each unit $i \in I$ a* state $s_i \in \{0, 1\}$, *a so called* bias or threshold $B_i \in \mathbb{Q}$ *and an* activation function g_i *is defined. The state s_i is computed as the function value of g_i, dependent on the weighted sum of all states s_j of neurons j connected to i, together with the biases B_i, more precisely:*

$$s_i = g_i(B_i, \sum_{j \in I} s_j w_{j,i}) := \begin{cases} 0, & B_i > \sum_{j \in I} s_j w_{j,i}, \\ 1, & B_i \leq \sum_{j \in I} s_j w_{j,i}, \end{cases} . \tag{3.1}$$

Further, the family of states $\mathbf{s} := (s_i)_{i \in I}$ is called the state *of the Hopfield network.*

We consider *asynchronous* Hopfield networks, which means any change in the state of the network only affects the state of one specific neuron. Synchronous state changes of neighbouring neurons can lead to undesired phenomena, as the computation of the function value of some activation functions may rely on unadapted information.

Definition 3.2.2 *Using the same notation as in the previous definition, letting $W = (w_{i,j})_{(i,j) \in I \times I}$ denote the matrix of weights and $\mathbf{B} = (B_i)_{i \in I}$ the vector of biases of a Hopfield network with units I, then the* energy *of the Hopfield network in state $\mathbf{s} = (s_i)_{i \in I}$,*

[1] There exist several examples of *asymmetric* ANNs in the literature with an underlying graph that is directed. In this section however, we only consider *symmetric* neural networks. The reason for this is that in our encoding, connections between vertices represent the existence of certain tuples that are common to the adjacent vertices, respectively a non-empty intersection of the two sets corresponding to the adjacent vertices. As these relations are symmetrical, we do not require directed edges.

is defined as the quadratic form

$$\begin{aligned} E(\mathbf{s}) &:= -\frac{1}{2}\mathbf{s}W\mathbf{s}^T + \mathbf{s}\mathbf{B}^T \\ &= -\frac{1}{2}\sum_{i\in I}\sum_{j\in I} w_{i,j}s_i s_j + \sum_{i\in I} s_i B_i. \end{aligned} \tag{3.2}$$

It is well known that Hopfield networks eventually, independent from the initial state, reach stable states at a local minimum of the energy function through asynchronous switches in the unit states, see e.g. [262].

The second kind of neural networks we consider in this section are Boltzmann machines, see also [259]. As Hopfield networks, Boltzmann machines have binary valued states and bidirectional, symmetric connections. However, opposite to Hopfield networks, they use a probabilistic state transition.

Definition 3.2.3 *A Boltzmann machine (BM) is a stochastic, recurrent neural network, with an underlying undirected weighted graph $G = (I, E)$. Formally, a Boltzmann machine is defined as a pair $(G, \omega(E))$ where $\omega(E) = \{w_e | e \in E\} \subseteq \mathbb{R}^E$ is the set of edge weights. Further, the* configuration *of a Boltzmann machine is defined as a binary vector $\kappa = (u_{i_1}, \ldots, u_{i_{|I|}}) \in K = \{0,1\}^{|I|}$, which describes for each neuron $i \in I$ whether it is in state* on *($u_i = 1$), or* off *($u_i = 0$).*

In this section we consider *sequential* Boltzmann machines, this means that as in the case of Hopfield networks, the state transition of neurons are asynchronous[2]. Further, let $\{i, j\}$ be a connection, then it is said to be *activated*, if both units i and j are in state *on*. The weight $w_{i,j}$ reflects the importance of the connection $\{i, j\}$ being activated, i.e. $w_{i,j} > 0$ if it is desired that $\{i, j\}$ is activated, and $w_{i,j} \leq 0$ otherwise. As an overall measure of the desirability of a configuration of the Boltzmann network, the consensus function is defined as follows.

Definition 3.2.4 *The* consensus function $F : K \to \mathbb{R}$ *of a Boltzmann machine in configuration κ maps the configuration κ to its* consensus $F(\kappa) = \sum_{\{i,j\}\in E} w_{\{i,j\}} u_i u_j$.

The aim of a Boltzmann machine is to reach a configuration of maximal consensus of its neurons. To that extent a transition mechanism is used to allow the units to adjust their states according to their neighbours. The transition of states is determined by a stochastic function based on the states of the units and the connection weights of the network. Whether a new state is accepted is based on a stochastic acceptance criterion,

[2]For both, Hopfield networks and Boltzmann machines, there exist also synchronous models, where multiple neurons can change their state *at the same time*. Since multiple state changes can always be reached by a sequence of individual state changes, we opted for the asynchronous models.

allowing the Boltzmann machine to escape from local maxima. The state transition in Boltzmann machines therefore resembles a simulated annealing search for the state maximizing the consensus function, see also [259].

Definition 3.2.5 *For a graph $G = (V, E)$ (in this section we only consider undirected graphs) with vertex set V and edges $E \subseteq V \times V$, a* vertex cover *is a subset C of V, such that each edge in E is incident to at least one vertex of C. An* independent set *of G is a set of vertices $I \subseteq V$, such that no two vertices in I are adjacent, i.e. $I \times I \cap E = \emptyset$.*

3.2.3 Boltzmann Neural Networks for Covering Array Generation

In this section we describe how Boltzmann neural networks can be used for CA generation. As it will serve us as a point of reference, we first recapitulate the work of [247], where Boltzmann machines were successfully used to compute solutions to set cover problems. Our aim is to combine this work with the mapping of the CA generation problem to the minimal set cover problem, see Subsection 2.3.1, in order to use Boltzmann machines to compute CAs.

Boltzmann Machines for Set Cover Problems

The work presented in [247], relies on a chain of mappings, reducing the set cover problem to a vertex cover problem on a graph, which again appears as the complement of an independent set problem on the same graph. The latter is finally solved using a Boltzmann neural network. For reasons of clarity, Figure 3.1 gives a high-level view of the procedure we are following to generate CAs using Boltzmann machines.

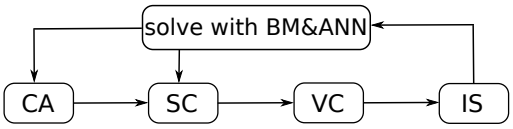


Figure 3.1: An overview of the presented procedures connecting covering arrays (CAs), set covers (SCs), vertex covers (VCs) and independent sets (IS), which will be solved with Boltzmann machines (BMs). See [181] and [247].

In the following paragraphs we discuss the connections between set covers, vertex covers and independent sets, as detailed in [247] and [141]. For a given SC instance $(U, \mathcal{S})$, we recall the construction of an edge labelled graph $G_{\mathcal{S}} = (V, E, \ell)$, used to represent this instance. The construction can be summarized as follows. The vertex set V is defined as the set of blocks $\mathcal{S}$, such that each block is represented by a vertex of the graph. The set of (undirected) edges E of the graph is defined by $E := \{\{S_i, S_j\} | S_i \cap S_j \neq \emptyset, S_i, S_j \in \mathcal{S}\}$. Edge labels are defined by the labelling function $\ell : E \to \mathcal{P}(U) : \{S_i, S_j\} \mapsto S_i \cap S_j$, that assigns as labels to an edge the set of elements of U its adjacent vertices cover in common. We call these labels also *label sets*.

Remark 3.2.6 *At this point we would like to remark, that we can assume without loss of generality, that each element of the universe U appears in the label set of at least one edge of the graph $G_{\mathcal{S}}$. Assume this was not the case, then some element of U would only be covered by a single block $S \in \mathcal{S}$. Then this block has to be in every set cover of U. Hence it is equivalent to consider the reduced SC instance $(U', \mathcal{S}') = (U \setminus S, \mathcal{S} \setminus \{S\})$, where every set cover $\mathcal{C}' \subseteq \mathcal{S}'$ of U' corresponds to a set cover $\mathcal{C} = \mathcal{C}' \cup \{S\}$ of U. Therefore, from now on we assume that each element of U appears in the label of at least one edge of the graph $G_{\mathcal{S}}$.*

Assume we are given a vertex cover $V = \{S_1, \ldots, S_r\}$ of $G_{\mathcal{S}}$, then V already represents a set cover of U. This holds, as the vertices (i.e. subsets of U) $S_1, \ldots, S_r$ cover all edges of $G_{\mathcal{S}}$, the labels of which are again subsets of S_i. Due to Remark 3.2.6, the union of all labels already cover the whole universe U, hence also $S_1, \ldots, S_r$ constitutes a cover of U. The authors of [247] consider reduced graphs $G'_{\mathcal{S}}$, where each element $u \in U$ appears in the label set of *exactly one* edge of $E(G_{\mathcal{S}})$. For all other edges u is removed from their label set, with the result that some label sets become empty. The reduced graph is then defined by containing only those edges with non-empty label set. We refer to such graphs as reduced graphs of *type I* in the following. The result is that the reduced graph contains less edges, while a vertex cover of the reduced graph still constitutes a set cover of $(U, \mathcal{S})$, since still each $u \in U$ appears as a label, see also Proposition 1 of [247]. In [141] we generalized this approach, where we considered reduced graphs $G'_{\mathcal{S}} = (V, E(G'_{\mathcal{S}}))$, in which each $u \in U$ appears in the label set of *at least* one edge of $E(G'_{\mathcal{S}})$. More precisely, to obtain the reduced graph $G'_{\mathcal{S}} = (V, E(G'_{\mathcal{S}}))$, for each element $u \in U$ we select one edge $e \in E(G_{\mathcal{S}})$ with $u \in \ell(e)$. All selected edges keep their entire label sets and appear in the reduced graph; edges not selected are removed from the edge set. We refer to such graphs as reduced graphs of *type II* in the following. This reduction made it possible to maintain the property that a vertex cover of a reduced graph $G'_{\mathcal{S}}$ constitutes a set cover of $(U, \mathcal{S})$ and to preserve more information about the original SC instance in $G'_{\mathcal{S}}$, while the number of edges is not increased when compared to reduced graphs of *type I*. We exemplify the difference between the two just described types of reduced graphs in Example 2.3.2 and Figure 3.3.

As shown in [141] we are able to formulate the following proposition for reduced graphs of *type II*, analogue to Proposition 1 of [247] for reduced graphs of *type I*.

Proposition 3.2.7 *Let $(U, \mathcal{S})$ be a set cover instance, represented by the edge labelled graph $G_{\mathcal{S}}$, then the complement of a maximal independent set in a reduced graph $G'_{\mathcal{S}}$ of* type II *(where each element of U appears in the label of at least one edge) is a set cover for $(U, \mathcal{S})$.*

In the following we use the notation $G_{t,k,v}$ for the graph that corresponds to the set cover instance $(U, \mathcal{S})$, which corresponds again to the CA instance for given t, k and $\mathbf{v}$ (see Subsection 2.3.1) and call $G_{t,k,v}$ the *underlying graph* of the CA instance. We rely on

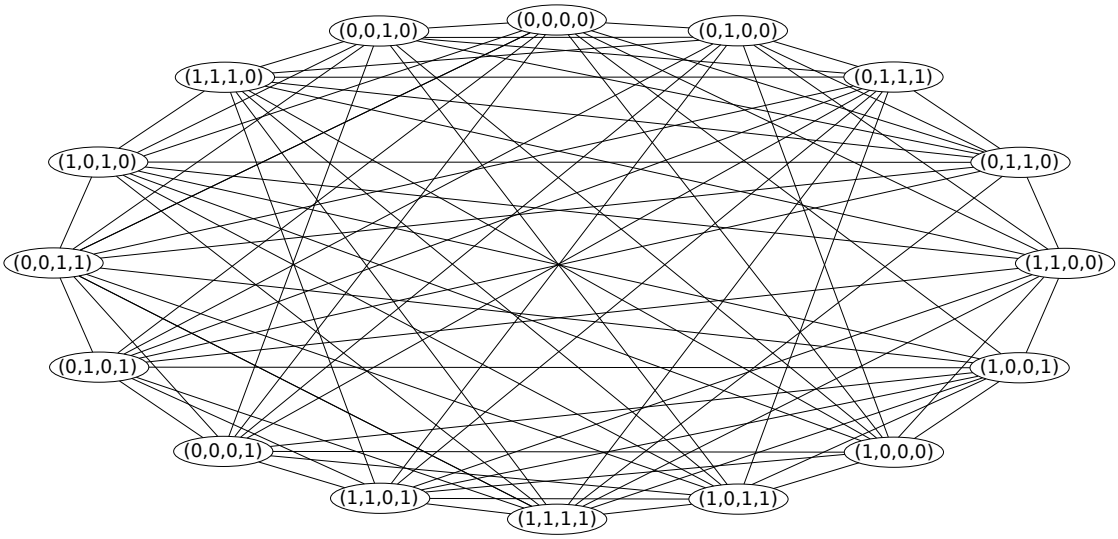


Figure 3.2: $G_{2,4,2}$ underlying the CA instance $t = 2$, $k = 4$, $v = 2$.

this notation also in the following examples, where we further exemplify the connection between covering arrays, their set cover interpretation and their underlying graph.

Example 2.3.2 (continuing from p. 64) *Continuing Example 2.3.2 the edge labelled graph $G_{2,4,2} = (V, E, \ell)$ corresponding to the set cover $(U, \mathcal{S})$ is depicted in Figure 3.2. The set of vertices V is given by the binary vector of length four, two vertices are connected by an edge if there exists a 2-way interaction that is covered by both vertices, and labels of edges are given by the set of 2-way interactions their adjacent vertices cover in common. E.g. the edges incident to the vertex $(1,1,1,1)$ have the following label sets:*

$$
\begin{aligned}
\ell(\{(1,1,1,1),(1,1,1,0)\}) &= \{(1,1,-,-),(1,-,1,-),(-,1,1,-)\}\\
\ell(\{(1,1,1,1),(1,1,0,1)\}) &= \{(1,1,-,-),(1,-,-,1),(-,1,-,1)\}\\
\ell(\{(1,1,1,1),(1,0,1,1)\}) &= \{(1,-,1,-),(1,-,-,1),(-,-,1,1)\}\\
\ell(\{(1,1,1,1),(0,1,1,1)\}) &= \{(-,1,1,-),(-,1,-,1),(-,-,1,1)\}\\
\ell(\{(1,1,1,1),(1,1,0,0)\}) &= \{(1,1,-,-)\}\\
\ell(\{(1,1,1,1),(1,0,1,0)\}) &= \{(1,-,1,-)\}\\
\ell(\{(1,1,1,1),(1,0,0,1)\}) &= \{(1,-,-,1)\}\\
\ell(\{(1,1,1,1),(0,1,1,0)\}) &= \{(-,1,1,-)\}\\
\ell(\{(1,1,1,1),(0,1,0,1)\}) &= \{(-,1,-,1)\}\\
\ell(\{(1,1,1,1),(0,0,1,1)\}) &= \{(-,-,1,1)\}
\end{aligned}
$$

Further, to exemplify the difference between reduced graphs of type I *and* type II*, in Figure 3.3 we consider the subgraph of $G_{2,4,2}$ consisting of all edges incident to $(1,1,1,1)$. In this subgraph, the labels occur right next to the edges they belong to.*

In Figure 3.3a we highlight in green the 2*-way interactions that are selected to reside in the label set of the edges. Figure 3.3b shows the subgraph of the reduced graph* $G'_{2,4,2}$ *of* type I *as it is described in [247] and Figure 3.3c shows the subgraph of the reduced graph* $G'_{2,4,2}$ *of* type II *as we consider it in this section. Notice that once a label of a label set is selected, the whole label set is selected, instead of individual elements of it. This results in larger label sets as compared to the reduced graphs of* type I.

Computing Covering Arrays with Boltzmann Machines

In [247] neural networks are constructed with an underlying graph that is isomorphic to $G'_{\mathcal{S}}$, where neurons correspond to vertices and synapses correspond to edges. Each neuron S_i is in state u_i, which can be either *on* or *off*, represented by $u_i \in \{0, 1\}$. Synapses, i.e. edges of $G'_{\mathcal{S}}$, are assigned weights based on the cardinality of their label sets, see equation (3.3). Assigning positive weights to loops $e_{i,i}$ and negative weights to other edges, such that $w_{i,i} < -w_{i,j}$ holds for all $i, j \in I$, assures that local maxima of the consensus function F correspond to maximal (not necessary *maximum*) independent sets of the graph $G'_{\mathcal{S}}$, which in turn yield set covers, considering the complement on $G_{\mathcal{S}}$. The interested reader is referred to [247] for more details.

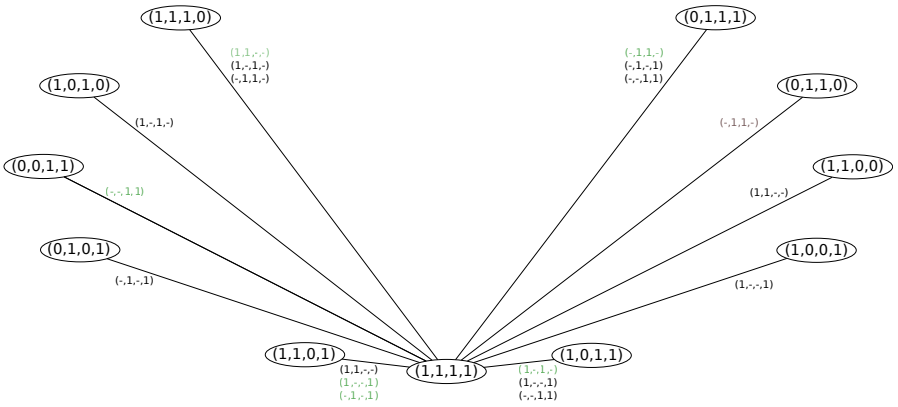
In [141] we combined the reductions of CAs to SCs and SCs to independent sets on reduced graphs $G'_{\mathcal{S}}$ (see Figure 3.1 for an overview) and accordingly modified the algorithm developed in [247] to be applied for CA generation. To summarize once more how the connections between the different structures translate to the introduced concepts and notions we want to give the following overview:

1. Rows of CAs correspond to blocks S_i of SCs, which are further mapped to vertices of BMs. These serve as neurons for the devised neural network.
2. Analogue, t-way interactions correspond to elements of the universe in terms of SCs. These serve as labels of edges that define the weight of the synapses of the devised neural network.

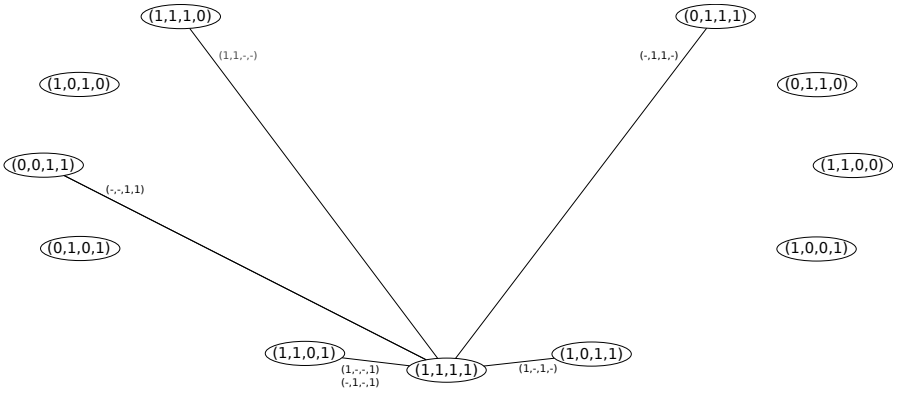
The detailed connections between covering arrays, set covers and independent sets made it possible to formulate the following corollary of Theorem 1 of [247], which also proves the correctness of the adopted algorithm, which we describe next.

Corollary 3.2.8 *Maxima of the consensus function* F *(Definition 3.2.4) induce configurations of the BM-network corresponding to Covering Arrays.*

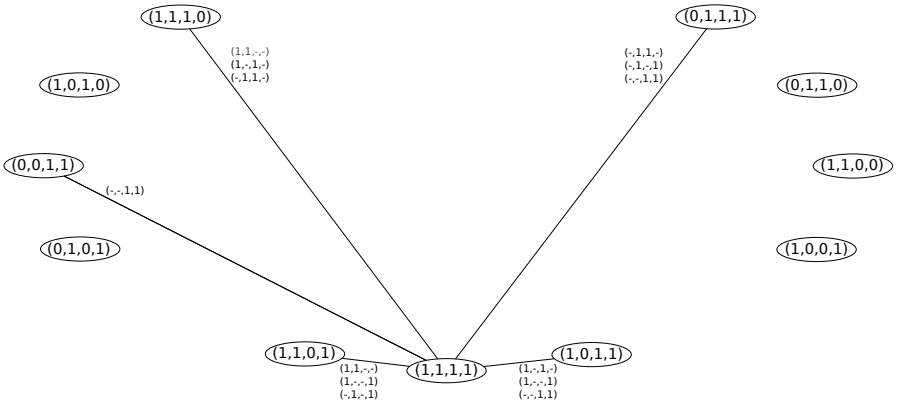
The adopted algorithm served as a base line for the development of our own learning algorithms in [141], which we will revisit in the next section. In light of these algorithmic modifications, we described Algorithm 4 in terms of our own algorithmic design using a variety of building blocks, in order to provide the necessary flexibility to formulate these different versions. A high-level description of the resulting algorithm is given in Algorithm 4, which takes as input the strength t and the number of columns k of the CA to be generated.



(a) Subgraph of $G_{2,4,2}$, where selected 2-way interactions are colored in green.



(b) The resulting reduced graph $G'_{2,4,2}$ of *type I*, when the subgraph is constructed based on the selection of 2-way interactions given in (a).



(c) The resulting reduced graph $G'_{2,4,2}$ of *type II*, when the subgraph is constructed based on the selection of 2-way interactions given in (a).

Figure 3.3: Subgraph of $G_{2,4,2}$, the underlying graph of the CA instance $t = 2$, $k = 4$, $v = 2$ and resulting reduced graphs applying a graph reduction of *type I* and *type II*.

The building block InitialGraph is a procedure that transforms the underlying graph $G_{t,k,v}$ of the CA instance to a subgraph $G'_{t,k,v}$, to which the Boltzmann machine is reduced. In this section we investigate two different procedures that perform a graph reduction of *type II* to instantiate the building block InitialGraph, one random heuristic and one randomized greedy heuristic:

- RandomGraph: This procedure selects for each t-way interaction a random edge of $E(G_{t,k,v})$, such that the t-way interaction resides in the label set of the edge. Edges selected this way reside in $G'_{t,k,v}$ and keep all their labels, where edges that get not selected are deleted. Our initial experiments with this fully randomized underlying graph were not satisfactory, which lead to the introduction of the second InitialGraph, that is more tailored towards the problem of CA generation.
- HeavyGraph: This procedure incrementally generates an edge that contains a maximized number of t-way interactions in its label set, that do not occur in the label set of any previously generated edge. This is done by connecting two rows that differ in only one position and share a maximized number of uncovered t-way interactions.

The second building block we used to devise our algorithms is that of InitialWeight, which is a procedure that assigns a weight to each edge of $G'_{t,k,v}$. Instantiating this building block as described in [247] yields a weight assignment in the following referred to as BMweight and for two blocks S_i, S_j defined by:

$$w_{ij} = \begin{cases} -(\max\{1/|S_i|, 1/|S_j|\} + \epsilon), & i \neq j \\ 1/|S_i|, & i = j \end{cases}. \tag{3.3}$$

It is worth mentioning that for CA instances these weights reduce to uniform weights, as $|S_i| = |S_j|$ for all i, j.

With these algorithmic building blocks, we were able to describe the algorithm of [247] adopted to CA instances as an instance of Algorithm 4, where InitialGraph is instantiated with RandomGraph and InitialWeights with BMweight. Finally, a simulated annealing algorithm SA is applied, to find a maximal independent set $\mathcal{I}$ on $G'_{t,k,v}$, the complement of which is returned and constitutes a CA. Note, that the weight assignment given by equation (3.3) fulfills the property $\omega_{i,i} < -\omega_{i,j},\ \forall i, j \in I$ and thus, when these weights are used for InitialWeight and the consensus function is in a minimum at the end of Algorithm 4, we have the guarantee that it returns a CA.

The simulated annealing algorithm SA takes as input a graph G with edge weights $\omega(E)$ and further requires a starting temperature T_0, a final temperature T_f and a factor λ for the cooling schedule. A pseudocode is given in Algorithm 5, and can be briefly described as follows. In each step a random neuron is selected to change its state. In case the change in the consensus function $\Delta F(\kappa) = (1-2u_i)(w_{ii}+\sum_j w_{ij}u_j)$ is positive the change in state u_i is accepted, otherwise it is refuted with probability $(1 - 1/(1 + \exp(-\Delta F/T))$. The

Algorithm 4 BMforCA

```
1: INPUT: t, k, v
Require: ϵ
2: G'_{t,k,v} ← InitialGraph(G_{t,k,v})
3: ω(G'_{t,k,v}) ← InitialWeight(G'_{t,k,v}, ϵ)                ▷ Assign weights
4: I ← SA(G'_{t,k,v}, ω(G'_{t,k,v}))
5: return CA(|V| − |I|; t, k, (v_1, ..., v_k)) = V \ I
```

cooling schedule of the SA algorithm is based on the schedule developed by Lundy and Mees [263], where it is required that only one iteration is executed at each temperature. In particular, we implemented the cooling schedule (line 12) according to the recursive function $T_{n+1} = T_n/(1 + \lambda T_n)$, where n is the iteration number and λ a small positive value close to zero depending on the instance, that allows for fast convergence.

Algorithm 5 SA

```
1: INPUT: G, ω(E)
Require: T_0, T_f, λ
2: T ← T_0
3: κ ← 0⃗                                  ▷ initially each neuron is off
4: while T > T_f do
5:     randomly choose neuron S_i
6:     change state u_i in κ: u_i ← u_i + 1(mod 2)
7:     if ΔF(κ) > 0 then
8:         keep κ
9:     else
10:        with probability 1/(1 + exp(−ΔF/T)) keep κ
11:    end if
12:    T ← T/(1 + λT)
13: end while
14: return κ
```

3.2.4 Learning Methods for Boltzmann Neural Networks for Covering Array Generation

In this section, we improve the previously described Algorithm 4, enhancing it with new capabilities in several steps. In detail, first we consider different assignments for the connection weights and introduce a feedback loop, called *epoch* enabling updates of these weights. In a second extension we introduce a notion of graph update acting on the underlying graph of the Boltzmann machine.

Before we describe the algorithmic extensions we make to Algorithm 4, we would like to mention that our initial experiments with Algorithm 4 were not satisfactory. Due to the good experimental results in [247], reporting to find smaller SCs than other heuristics, we expected that Algorithm 4 would produce CAs with a small number of rows. However, also our objective of having a learning algorithm capable of further reducing the number of rows in a CA, without specifying a target number of rows is still not achieved. We believe that this is the case, since the approach of finding small set covers as the complements

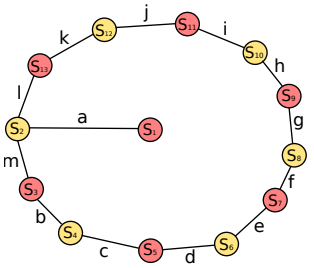


Figure 3.4: A graph corresponding to the set cover $(U, \mathcal{S})$ in Example 3.2.9.

of large independent sets of vertices on corresponding graphs is badly suited for graphs that have a relative high *density*, i.e. on average, vertices are highly connected. It seems that the condition of finding an independent set of nodes on the reduced graph $G'_{t,k,v}$, is too strong of a sufficient condition to actually find small SCs for such instances. To illustrate this we give the following example of a general SC instance, which is highly connected and the algorithm described in [247] is badly suited for.

Example 3.2.9 *Consider the following set cover instance* $(U, \mathcal{S})$, $U = \{a, b, c, \ldots, k, l, m\}$ *and* $\mathcal{S} = \{S_1, \ldots, S_{13}\}$ *where* $a \in S_i$ *for all* $i = 1, \ldots, 13$ *and* $S_1 = \{a, m\}, S_2 = \{a, m, b, c\}, S_3 = \{a, b, c, d\}, S_4 = \{a, c, d, e\}, S_5 = \{a, d, e, f\}, S_6 = \{a, e, f, g\}, S_7 = \{a, f, g, h\}, S_8 = \{a, g, h, i\}, S_9 = \{a, h, i, j\}, S_{10} = \{a, i, j, k\}, S_{11} = \{a, j, k, l\}, S_{12} = \{a, k, l, m\}, S_{13} = \{a, l, m, b\}$. *The graph representing the set cover instance* $(U, \mathcal{S})$ *is the complete graph with* 13 *vertices. In Figure 3.4 we give an example of a reduced graph of this set cover instance, in the sense of [247], i.e. each element of the universe appears as a label of exactly one edge. A maximal (even maximum) independent set in this reduced graph nodes can be identified as* $\mathcal{I} = \{S_1, S_3, S_5, S_7, S_9, S_{11}, S_{13}\}$. *Then the complement* $\mathcal{C} = \{S_2, S_4, S_6, S_8, S_{10}, S_{12}\}$ *constitutes a minimal vertex cover of this graph, and hence* $\mathcal{C}$ *is a cover of the universe* U. *Though,* $\mathcal{C}$ *is not a minimal set cover, since* $\mathcal{C}' = \{S_2, S_5, S_8, S_{11}\}$ *also constitutes a cover of* U *of smaller size. In fact it is not hard to see that* $\mathcal{C}'$ *is a minimal set cover of* $(U, \mathcal{S})$.

We take this example as a further motivation to modify the approach presented in [247], towards relaxing the target of finding an independent set on the reduced graph $G'_{t,k,v}$. Finding independent sets is encoded in the consensus function F (as introduced in Definition 3.2.4), that characterizes independent sets through local maxima, as long as weights of vertices are positive, and the weights of the edges are smaller than the negative vertex weights. Using the same consensus function, our approach is to increase the edge weights, such that local maxima of F can originate also from vertex sets containing adjacent vertices. From this we gain, that when maximizing consensus F more neurons are in state *on*. Hence the complement, the neurons in state *off*, can be less than in the original approach of [247], leading eventually to smaller set covers, respectively CAs in our case. On the downside, we lose the property that the neurons in state *off* always translate to a set cover, respectively a CA in our case, as it was guaranteed by Corollary 3.2.8. We address this issue by evaluating the returned solution, and updating the weights of the edges, realized in the building block WeightUpdate. Then we maximize the

consensus F for the updated instance. The key idea behind this approach is, that the neural network decreases the weights of those edges that carry elements as labels that were not covered in the previous iteration. This modifies the network, such that in the next iteration it is less likely that all neurons connected by such edges are turned *on* and hence some will remain turned *off*, which means they will be part of the suggested solution of the set cover. We detail our edge updates and learning features in the next section. The experimental results provided in Subsection 3.2.6 fully justify this approach.

Weight Updates: A First Step Towards Learning

New Initial Weights. One change we made to the algorithm as it is presented in [247] is that we changed the computation of the edge weights. This is done by assigning the weights as a function of $|S_i \cap S_j|$ instead of $\max(|S_i|, |S_j|)$. The number of t-way interactions two rows S_i and S_j cover in common depends on the number of positions in which these rows are equal, we hence can compute $|S_i \cap S_j| = \binom{k-d_{ij}}{t}$, where d_{ij} denotes the *hamming distance*[3] of the two rows S_i and S_j. We consider an additional instantiation of the building block InitialWeights:

- HDWeight: $w_{ij} = -\binom{k-d_{ij}}{t} \cdot 1/\binom{k}{t}$

for $i \neq j$, and in both cases and $w_{ii} = 1$ for the loops. In Subsection 3.2.6 we will also compare the results when the initial edge weighting BMWeight and HDWeight are used in Algorithm 4.

Weight Updates: Learning in Epochs. The next enhancement to the algorithm presented in [247] was achieved by extending it by means of epochs in which the weights of the edges connecting neurons get updated. This algorithmic extension was implemented for two reasons: First and foremost we wanted the neural network to be able to adapt to given problem instances. Second, since we gave the neural network more freedom by weakening the consensus function F by assigning larger weights to edges using our newly introduced version of InitialWeights, we are not guaranteed anymore that the output of the SA algorithm constitutes an independent set and hence its complement must not constitute a CA. In short, we lose the guarantee of a feasible solution as it was guaranteed by Corollary 3.2.8.

Therefore we extend the capabilities of the neural network, allowing it to increase or decrease the weight of edges, depending on whether the elements in their label sets were covered in the solution returned in the previous epoch. This new algorithmic building block WeightUpdate can be described as procedure that modifies the weight of the edges of the underlying graph $G_{t,k,v}$, in the following way. Whenever a t-way interaction is covered more than twice in the solution of the previous epoch, all edges that have this interaction in its label set get an increment of $1/cov$ in weight (recall that edge weights are initialized negative), where cov is the total number of covered t-way interactions.

[3]The hamming distance of two vectors is defined as the number of positions they disagree.

Opposite, every edge carrying an interaction that was not covered by the solution returned in the previous epoch gets a proportional decrement in weight. The weights of some edges get smaller and in the next epoch it is less likely that both vertices adjacent to such an edge will be in state *on*, i.e. in the independent set to be constructed. This in turn means, that at least one of the vertices will be in the complement, i.e. the return of the next epoch. In terms of CAs, which appear as the complement of the generated independent set, this means that it is more likely that one of the rows covering a previously uncovered t-way interaction is selected to be part of the array generated in the next epoch. In this way the importance of covering specific t-way interactions in the next epoch is encoded in the edge weights.

We present Algorithm 6 in terms of a pseudocode. First, a reduced graph $G'_{t,k,v}$ is constructed and initial weights are assigned. Further, a global best solution is recorded in $\mathcal{I}_{max}$, which is initially set empty. Then a number e of epochs is run, where in each epoch x runs of SA are executed, where we keep the solution $\mathcal{I}$ maximizing the consensus F over these x runs. The procedure WeightUpdate is based on this solution $\mathcal{I}$. If $\mathcal{I}$ is larger than $\mathcal{I}_{max}$ and $V \setminus \mathcal{I}$ covers all t-way interactions $\mathbb{T}_{v,t}$, we store it accordingly before entering the next epoch. Finally if $V \setminus \mathcal{I}$ covers all t-way interactions, a CA is found and returned.

Algorithm 6 BMforCAlearning

```
1: INPUT: t, k, v
Require: e, x
2: G'_{t,k,v} ← InitialGraph(G_{t,k,v}), ω(G'_{t,k,v}) ← InitialWeights(G'_{t,k,v})   ▷ Initialization
3: I_max ← ∅
4: while epoch count ≤ e do
5:     run x times SA on G'_{t,k,v}, store I maximizing consensus
6:     ω(G'_{t,k,v}) ← WeightUpdate(G'_{t,k,v}, I)
7:     if |I_max| < |I| and V \ I covers all t-way interactions then
8:         I_max ← I
9:     end if
10: end while
11: if V \ I covers all t-way interactions then return CA(|V| − |I_max|; t, k, (v_1, ..., v_k)) = V \ I_max
12: else return V \ I_max with additional coverage information
13: end if
```

Graph Updates: An Additional Layer for Learning

In our experiments we recognized that the quality of the solutions produced by Algorithm 6 depends on the graph that is chosen in the initialization step.

Thus, we strived to enhance the learning rate of the neural network by augmenting it with the capability of modifying the reduced graph that the Boltzmann machine runs on. These modifications are realized in a building block called GraphUpdate and happen in an additional layer of learning phases built around Algorithm 6. The GraphUpdate gives the ANN the capability to explore different underlying graphs, while taking advantage of the optimizations in previously examined graphs. A pseudo code description can be seen in Algorithm 7. The initialization is the same as in Algorithm 6, also the structure of the learning in epochs is the same as in Algorithm 6 (lines 6 - 12 of Algorithm 7 coincide

with lines 4 - 10 of Algorithm 6, where variable $\mathcal{I}_{max}$ gets renamed to $\mathcal{I}_{learn}$). Around these epochs n learning phases are run, where at the beginning of each learning phase the $\mathcal{I}_{learn}$ parameter is reset to the empty set. At the end of each learning phase a graph update based on $\mathcal{I}_{max}$ and a bias update based to the best solution $\mathcal{I}_{learn}$ found during this learning phase occurs. Both procedures act on the underlying graph $G_{t,k,v}$ and are explained more detailed as follows.

For the key procedure GraphUpdate we introduce the following instances:

- BestEdges: In each learning phase a subset L of the nodes (respectively rows) of $V \setminus \mathcal{I}_{max}$ is randomly selected. For each row in L we flip a random position to create a second row, to which we draw an edge in the graph. By only flipping one position we generate a row that shares the maximal number of t-way interactions with the original row. The edge thus constructed has a large label set. Thereafter, for each t-way interaction that is not present in any of these label sets, we generate a random edge having it as a label, just as in InitialGraph. With this strategy the neural network can reduce the number of edges in the new reduced graph.

- HeavyEdges: In each learning phase a subset L of the nodes (respectively rows) of $V \setminus \mathcal{I}_{max}$ is randomly selected. For each row in L we flip a random position to create a second row, to which we draw an edge in the graph. For the remaining t-way interactions not represented in any label set yet, we generate additional edges, following the same strategy as in the HeavyGraph initialization, i.e. by incrementally generating edges that contain a maximized number of uncovered t-way interactions.

To guide the neural network and enable it to learn from solutions previously found, we added the additional functionality of BiasUpdate. The bias update acts on the neurons, rather than on the synapses of the neural network. In our encoding it can be realized as a weight update, acting exclusively on the loops, by adding a certain, relatively small, δ to the weight of the loops. The bias update is a way to reward vertices that were part of previous solutions, so that the Boltzmann network has a larger tendency to include them in future solutions. This is due to the structure of the consensus function F (Definition 3.2.4), which value increases whenever a vertex with an increased weight $w_{ii} + \delta$ is activated, instead of a vertex with edge weight w_{ii}. Vertices being part of $\mathcal{I}_{learn}$ in several learning phases are incrementally rewarded through this bias update. Note that due to bias updates, and also updates of edge weights the cumulative weight in the whole network is not constant over several learning phases.

Extension to Mixed-level and Constrained Covering Arrays

The presented methods and algorithms can be generalized also to generate MCAs, since the first step of our approach, the mapping of CA instances to SC instances can also be generalized to MCAs. The interested reader is referred to [144] for more details, where an analogue to Example 2.3.2 formulated for MCAs is given in Example 11 therein.

Algorithm 7 BMforCAlearningGraph

INPUT: t, k, v
Require: e, x, n, δ
$G'_{t,k,v} \leftarrow$ InitialGraph$(G_{t,k,v})$, $\omega(G'_{t,k,v}) \leftarrow$ InitialWeights$(G'_{t,k,v})$ ▷ Initialization
$\mathcal{I}_{max} \leftarrow \emptyset$
while learning phases $\leq n$ **do**
    $\mathcal{I}_{learn} \leftarrow \emptyset$
    while epoch count $\leq$ e **do**
        run x times SA on $G'_{t,k,v}$, store $\mathcal{I}$ maximizing consensus
        $\omega(G'_{t,k,v}) \leftarrow$ WeightUpdate$(G'_{t,k,v}, \mathcal{I})$
        if $|\mathcal{I}_{learn}| < |\mathcal{I}|$ and $V \setminus \mathcal{I}$ covers all t-way interactions **then**
            $\mathcal{I}_{learn} \leftarrow \mathcal{I}$
        end if
    end while
    if $|\mathcal{I}_{max}| < |\mathcal{I}_{learn}|$ **then**
        $\mathcal{I}_{max} \leftarrow \mathcal{I}_{learn}$
    end if
    $G'_{t,k,v} \leftarrow$ GraphUpdate$(G_{t,k,v}, \mathcal{I}_{max})$
    $G_{t,k,v} \leftarrow$ BiasUpdate$(\mathcal{I}_{learn}, \delta)$
end while
return $V \setminus \mathcal{I}_{max}$ with coverage information

Some applications of MCAs, for example when they are applied in real-world testing scenarios involving complex systems with (hard) constraints between their input parameters, it is required that specific t-way interactions do not appear covered by the rows of the MCA. This leads to the notion of *constrained mixed-level covering arrays* (CMCAs), see also [96]. In practice the constraints are mostly formulated via some first-order logic expressions, which can be translated to a set of *forbidden interactions*, that are not allowed to be covered. We assume that the constraints are already given by means of such a set of forbidden interactions.

Definition 3.2.10 *For positive integers t, k and $v_1, \ldots, v_k$ with $t \leq k$ and a set $\mathcal{F}$, where each element of $\mathcal{F}$ is a $(v_1, \ldots, v_k)$-ary s-way interaction for some positive integer s, a constrained mixed-level covering array* $\mathsf{CMCA}(N; t, k, (v_1, \ldots, v_k), \mathcal{F})$ *is defined as an $N \times k$ array $(\mathbf{c}_1, \ldots, \mathbf{c}_k)$, where the entries in the i-th column $\mathbf{c}_i$ arise from the integer interval $\{0, 1, \ldots, v_i - 1\}$ and it holds that*

1. *none of the elements in $\mathcal{F}$ are covered by the rows of the CMCA and*
2. *all t-way interactions in $\mathbb{T}_{(v_1,\ldots,v_k),t}$ that do not contain any of the s-way interactions given by the elements of $\mathcal{F}$ are covered by the rows of the CMCA.*

The existence of CMCAs is not trivially given, as there exist sets of forbidden tuples that exclude the existence of the related CMCA and deciding the existence of a CMCA is not trivial either as it can be an NP-hard problem, see [119].

When we want to generate also CMCAs, with our described algorithms we only have to apply a slight modification to the mapping of MCA instances to SC problems. Assume we are given parameters $t, k, (v_1, \ldots, v_k)$ and a set $\mathcal{F}$ of forbidden interactions and we want to construct a $\mathsf{CMCA}(N; t, k, (v_1, \ldots, v_k), \mathcal{F})$. In order to map the CMCA instance correctly to a set cover instance $(\mathcal{S}, U)$ (the first step in the overview provided by Figure 3.1), we merely need to ensure that all rows that cover any of the s-way interactions given by $\mathcal{F}$ are not mapped to blocks and thus are not represented in the set of blocks $\mathcal{S}$. Similarly, the universe U consists only of the t-way interactions of $\mathbb{T}_{(v_1,\ldots,v_k),t}$ that do not contain any s-way interaction in $\mathcal{F}$. Provided this slight modification of the first mapping, we are guaranteed to have created an SC instance that is consistent with the constraints represented by $\mathcal{F}$. All other steps of the problem mapping follow those for CA generation in an analogue manner. The SC instance is again encoded as a graph analogue to the construction of $G_{t,k,v}$, with the sole difference, that there are less vertices, edge-weights and potentially also less edges, due to the excluded blocks and elements. Once a solution to the resulting minimal SC problem is found, we can translate it to a $\mathsf{CMCA}(N; t, k, (v_1, \ldots, v_k), \mathcal{F})$.

3.2.5 Hopfield Neural Networks for Covering Arrays

As a second ANN we describe how Hopfield neural networks can be used for CA generation. The starting point of our algorithm is the work presented in [248], where Hopfield networks are used to solve set cover problems. Together with the mapping of the problem of generating optimal CAs to the minimal set cover problem presented in Subsection 2.3.1, we again rely on a two-step approach, first interpreting the CA generation problem as a set cover problem, which can then be solved via the Hopfield neural network. We briefly review the work from [248] with a focus on the parts being essential for our purpose.

Hopfield Neural Networks for Set Cover Problems

The authors of [248] consider set cover instances with a given upper bound on the number of blocks covering each element of the universe. For such an instance $(U, \mathcal{S})$ the Hopfield ANN is designed consisting of $|\mathcal{S}|$ neurons and the underlying graph $G = (I, E)$ can be defined as follows. Each block $S_i \in \mathcal{S}$ is represented by a vertex $i \in I$ of the underlying graph.

Neurons can be in states *on* ($s_i = 1$) or *off* ($s_i = 0$), where the state *on* means that the corresponding block S_i is being selected to be part of the set cover, and state *off* means it is not. Thus, the goal of finding a minimal set cover can be expressed by means of minimizing the number of neurons being in active state, i.e. minimizing $\sum_{i \in I} s_i$, subject to the constraint that each element of the universe U appears in at least one of the sets

corresponding to the active neurons, i.e. $\sum_{i\in I}\chi(u,S_i)s_i \geq 1$ for each $u \in U$, where

$$\chi(u,S_i) = \begin{cases} 1, u \in S_i, \\ 0, u \notin S_i. \end{cases}$$

Combining these two objectives, as presented in [248], we can express the target of finding a minimal set cover by minimizing the energy function,

$$E = \lambda \sum_{i\in I} s_i + \gamma \sum_{u\in U} \Big(d - \sum_{i\in I}\chi(u,S_i)s_i\Big)^2, \tag{3.4}$$

where λ and γ are constants and d is a parameter that needs to be specified properly. The second term in (3.4) represents quadratic constraints. When d is properly selected, then for given $u \in U$ the term $(d - \sum_{i\in I}\chi(u,S_i)s_i)$ is maximized, if u is not covered at all, and is minimized if u is covered by exactly d blocks. Finding an appropriate value for d is subject to optimization itself and depends on the given set cover instance. Consider for example a given set cover instance, with an upper bound m of blocks covering each element of the universe. Setting $d = 0$ is too small, as it does not penalize uncovered elements, whereas $d = m + 1$ is too large, even penalizing elements that are covered the maximal number of m times and especially punishing elements that are covered exactly once. Thus, too large of a value for d encodes the purpose of covering each element a maximal number of times, which is not desired for a minimal set cover instance. We will see in Algorithm 8 that the bounds 0 and $m+1$ are used in a binary search to determine an appropriate value for d.

From (3.4) we can now derive the weights $w_{i,j}$ and the biases B_i for the energy of the Hopfield network defined in 3.2 as follows:

$$\begin{aligned} E &= \lambda \sum_{i\in I} s_i + \gamma \sum_{u\in U} \Big(d - \sum_{i\in I}\chi(u,S_i)s_i\Big)^2 \\ &= \lambda \sum_{i\in I} s_i + \gamma \sum_{u\in U} d^2 - 2d\gamma \sum_{u\in U}\sum_{i\in I}\chi(u,S_i)s_i + \gamma \sum_{u\in U}(\sum_{i\in I}\chi(u,S_i)s_i)^2 \\ &= \sum_{i\in I} s_i \underbrace{(\lambda - 2d\gamma \sum_{u\in U}\chi(u,S_i))}_{B_i} + \underbrace{\gamma \sum_{u\in U} d^2}_{const.} + \sum_{i\in I}\sum_{j\in I} s_i s_j \underbrace{\gamma \sum_{u\in U}\chi(u,S_i)\chi(u,S_j)}_{-\frac{1}{2}w_{i,j}}. \end{aligned}$$

The constant summand does not affect the minimization of the energy function E and can thus safely be ignored. As $\sum_{u\in U}\chi(u,S_i)) = |S_i|$ and $\sum_{u\in U}\chi(u,S_i)\chi(u,S_j) = |S_i \cap S_j|$, we get the following weights and biases, where loops, i.e. edges of the form $\{i,i\}$ get assigned the weight 0 according to the definition of Hopfield networks:

$$\begin{aligned} w_{i,i} &= 0, \text{ for all } i \in I, \\ w_{i,j} &= -2\gamma|S_i \cap S_j|, \text{ for } i \neq j \in I, \\ B_i &= \lambda - 2d\gamma|S_i|, \text{ for all } i \in I, \end{aligned} \tag{3.5}$$

with λ and γ being constants, required by the algorithm. Recalling that the energy of Hopfield networks (3.4) is to be minimized, these weights and biases can be interpreted as follows. The weights $w_{i,j}$ are large, when the two blocks S_i and S_j cover many common elements and thus it is likely to be undesired to have both of them in a minimal set cover of the universe. Similarly, the B_i reflect a prioritization of large blocks, as this criterion is also used in a radical way as unique selection criterion by greedy approaches to the set cover problem (see e.g. [182] and references therein).

Using the weights as specified in equation (3.6), asynchronous switches are used to find a stable state of the Hopfield network with (locally) minimal energy. Representing a greedy heuristic, a steepest descent approach of serial unit state switches is used to maximize the energy given in equation (3.4) of the network.

Computing Covering Arrays with Hopfield Neural Networks

Adopting Algorithm HsCN-D as presented in [248] to set cover instances that are derived from a given CA instance, i.e. adopting it to the CA generation problem, we can formulate Algorithm 8. This algorithm takes as input the parameters t, k and v of the desired CA. The algorithm can be outlined as a binary search for the parameter d of the energy function (3.4) (step 6) wrapped around a minimization of the energy of the Hopfield network (step 14). The bounds for the binary search for d are set to be $low = 0$ and $high = v^{k-t}$. This is because $high$ is the number of rows covering each t-way interaction and thus the (maximal) number of blocks covering each element of the universe in the set cover instance corresponding to the CA instance with k columns and strength t. Further, we have $|S_i| = \binom{k}{t}$, as every row covers exactly that many t-way interactions. The binary search terminates when the difference between d and its previous value falls below a given threshold δ. The actual minimization of the Hopfield energy (3.4) is realized by a steepest descent algorithm that sequentially activates the neuron, which change of state yields the largest decrease in energy. In contrast to the algorithm proposed in [248], in our algorithm, at the beginning of each iteration the initial states of the neurons are all set to zero. Recall that asynchronous Hopfield networks always reach a stable state of (local) minimal energy independent from the initial state, see e.g. [262] or [264].

3.2.6 Experimental Evaluation

In this section we report experimental results for different algorithms and different configurations thereof, to investigate and demonstrate the feasibility of ANN based approaches for covering array generation. In order to do so, we first experiment with different configurations of our algorithms using Boltzmann machines introduced in Subsection 3.2.4, evaluating also the effects of the introduced learning capabilities via weight and graph updates, with regards to the number of rows of generated CAs. Second, we compare the performance of the different algorithms using Boltzmann neural networks with the algorithm using Hopfield neural networks, described in Subsection 3.2.5. We implemented all algorithms in Rust and performed the experiments on a server equipped with 64 GB of RAM and an Intel Xeon E3 processor. In the experiments conducted,

Algorithm 8 HFforCA

INPUT: t, k, v
Require: λ, y, δ
$low = 0$
$high = v^{k-t}$
$d = v^{k-t}/2, d_b = 0, d_p = 0$
$\mathbf{S} = (S_i)_{i \in I} \leftarrow \mathbf{0}$ ▷ Initially no neurons are activated.
while $|d - d_p| \geq \delta$ and S is not CA **do**
  for $i \in I$ **do** ▷ Assign weights according to (3.6)
    $B_i \leftarrow 2dy|S_i| - \lambda$
    $w_{i,i} \leftarrow 0$
    for $j \in I \setminus \{i\}$ **do**
      $w_{i,j} \leftarrow -2y|S_i \cap S_j|$
    end for
  end for
  $S' \leftarrow$ [Apply steepest descent to initial state S until stable state is reached]
  if S' is a CA **then**
    $high \leftarrow d$
    $d_b \leftarrow d$
  else
    $low \leftarrow d$
  end if
  $d_p \leftarrow d$
  $d \leftarrow (low + high)/2$
  $S \leftarrow S'$
end while

for the algorithms using Boltzmann machines we used the following settings regarding the simulated annealing algorithm. Temperatures and the factor λ were set to $T_0 = 1$, $T_f = 0.001$ and $\lambda = (1 - T_f)/(T_f \cdot iter)$, where $iter$ is a linear function of the number of vertices in the the reduced graph $G'_{t,k,v}$. The number of inner SA cycles x for configurations of Algorithm 4, 6 and 7 was set to $x = 3$.

In our evaluations, documented in the following subsections, we conducted experiments for CA instances of strengths $t \in \{2, 3, 4\}$, $5 \leq k \leq 50$ and $v \in \{2, 4\}$. We would like to remark that although the numbers specifying the CA instances seem very small, the respective optimization problems are not so. That is, given a CA instance (t, k, v), the underlying graph $G_{t,k,v}$ has v^k vertices, each having $\sum_{i=t}^{k-1}(v-1)^{k-i}\binom{k}{i}$ adjacent vertices. For example, for the CA instance $(t = 2, k = 10, v = 2)$ the underlying graph $G_{2,10,2}$ has already 1024 vertices, each having 1012 edges to other vertices.

Experimental Evaluation of Algorithm BMforCA, Algorithm 4

In the following, we compare different *configurations* of Algorithm 4, using the different instantiations for the occurring building blocks. We pair the two versions of InitialGraph with the two versions of InitialWeight. The column headings of Table 3.1

reflect the respective pairing as InitialGraph + InitialWeight. The first column of the table specifies the CA instance and the second column represents the state of the art, as it shows the currently best known upper bound for the respective $\mathsf{CAN}(t, k, v)$, which is maintained at [97]; the values for $\mathsf{CAN}(2, k, 2)$ are known to be precise. Since we deal with randomized algorithms, we executed 100 individual runs for each configuration. The column headed by *min* shows the smallest number of rows of generated *covering arrays*. Further, in the columns headed by *avg, avg cov* and *time* we document the average number of rows, the average percentage of covered t-way interactions and the average run time in milliseconds over *all* generated arrays. Note that the configurations of Algorithm 4 using BMWeight always return CAs. This is guaranteed by Corollary 3.2.8, which ensures that *all* algorithms examined in this section that use BMWeight as InitialWeight are guaranteed the return of a CA.

In our experiments using RandomGraph with the initial weights with BMWeight, we could not reproduce results of similar quality as documented in [247] achieved for general set cover instances. The version using HDWeight, on the other hand, abandons the concept of low edge weights and a guaranteed return of a CA in order to be able to generate smaller CAs. The experiments however showcase that this concept is only successful for some instances with a small number of columns ($k \leq 15$), as the algorithm fails to construct CAs for larger instances when using HDWeight.

This phenomenon can be explained by the number of vertices of the underlying graph $G_{t,k,v}$ growing exponentially as v^k and the number of edges being upper bound by $\binom{k}{t}v^t$. Hence the resulting reduced graph of *type II* becomes more sparse, the higher k or v grow. In such sparse graphs, with high edge weights, too many neurons in the Boltzmann machine will end up in state *on*, which results in too small arrays that do not cover all t-way interactions.

The configuration using BMWeight is also affected by the sparsity of the reduced graph. We can see that for instances with larger k the size of the majority of returned CAs is exactly $\binom{k}{t}v^t$. This can be explained by the randomized construction of the reduced graph yielding a pairing of vertices for each t-way interaction (the maximal edge degree in the graph is 1), where from each pair exactly one vertex ends up in state *off* and is selected into the CA.

For the experiments documented in Table 3.1, the configuration using HeavyGraph with BMWeight outperforms those using RandomGraph, in terms of generating smaller CAs, in all instances and stays close below the optimal value $\mathsf{CAN}(2, k, 2)$ for the binary CA instances. This difference in quality of output arrays indicates the importance of the reduced graph the Boltzmann machine is acting on. The configuration using HDWeight again rarely returns a CAs, which again is explained by the sparsity of the reduced graph and the initial weights being too large. Regarding the (average) run time we can see that the configurations using HeavyGraph are faster than those using randomGraph, which is also explained by the reduced graph having less vertices and thus the optimization of the Boltzmann machine being faster. A visual representation of selected results from

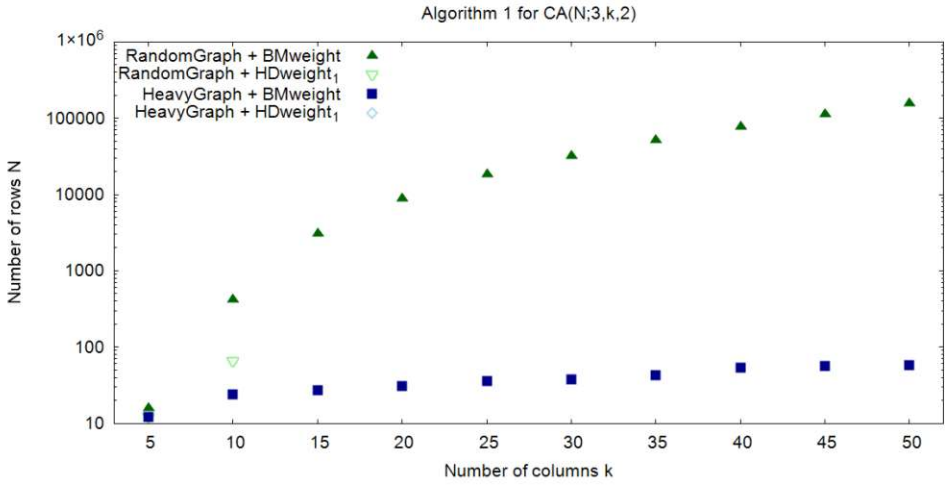


Figure 3.5: Minimal number of rows (smaller is better) of generated CAs by configurations of Algorithm 4 applied to the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$. Not all configurations can find CAs. Note the vertical axis is in $\log_{10}$-scale.

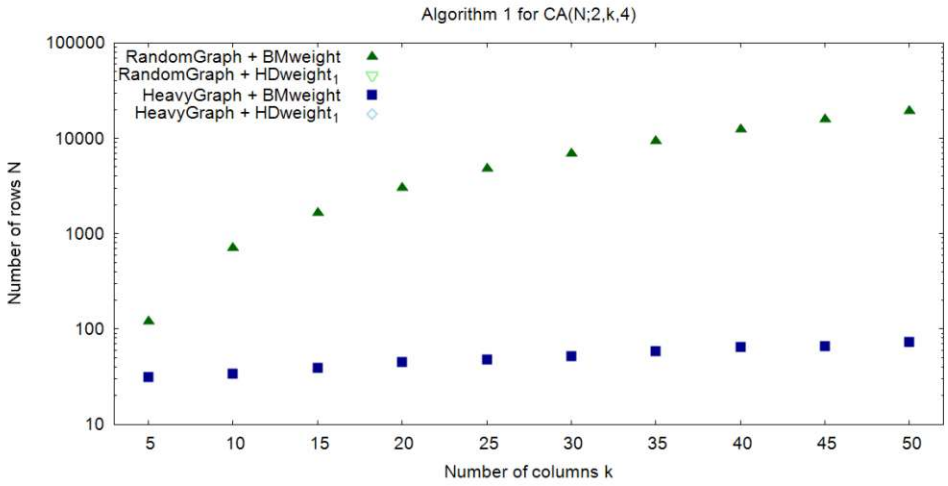


Figure 3.6: Minimal number of rows of generated CAs by configurations of Algorithm 4 applied to the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. Only two configurations can find CAs. The vertical axis depicts the number of rows (smaller is better) of CAs found by the different configurations. Note the vertical axis is in $\log_{10}$-scale.

Table 3.1 is given in Figure 3.5 and 3.6, where the minimum number of rows, for which the four configurations successfully found CAs is depicted.

Summarizing, we can see that the initial weighting of edges in the graph, respectively of synapses in the neural network, and the selection of the reduced graph is crucial for the quality of the output of the tested algorithms.

Table 3.1: Results of the experiments with configurations of Algorithm 4, using no learning mechanisms.

(t, k, v)	[97]	RandomGraph + BMweight				RandomGraph + HDweight				HeavyGraph + BMweight				HeavyGraph + HDweight			
		min	*avg*	*avg cov*	*time*	*min*	*avg*	*avg cov*	*time*	*min*	*avg*	*avg cov*	*time*	*min*	*avg*	*avg cov*	*time*
(2,5,2)	6	12	14.53	100 %	9.99	6	6.83	93.45 %	10.06	7	8.07	100 %	15.14	-	1.21	28.98 %	9.97
(2,10,2)	6	127	139.19	100 %	10.99	15	7.04	84 %	10.04	9	9.66	100 %	15.9	-	0.01	0 %	10.29
(2,15,2)	7	403	411.91	100 %	12.14	-	0.6	13.81 %	11.11	10	10.5	100 %	16.93	-	0.01	0.25 %	11.6
(2,20,2)	8	756	759.32	100 %	16.05	-	0.05	1.25 %	14.62	10	11.4	100 %	19.52	-	0.01	0.25 %	14.72
(2,25,2)	8	1198	1199.92	100 %	21.54	-	0	0 %	19.26	11	12.17	100 %	25.24	-	0.04	1 %	19.11
(2,30,2)	8	1739	1739.99	100 %	27.51	-	0	0 %	24.62	12	12.65	100 %	34.28	-	0.02	0.50 %	26.14
(2,35,2)	8	2380	2380	100 %	39.04	-	0	0 %	34.36	12	13.21	100 %	47.16	-	0.03	0.75 %	36.39
(2,40,2)	9	3120	3120	100 %	50.64	-	0	0 %	45.02	13	13.86	100 %	64.85	-	0.06	1.50 %	50.42
(2,45,2)	9	3960	3960	100 %	69.89	-	0	0 %	60.78	13	14.24	100 %	87.67	-	0.04	1 %	68.8
(2,50,2)	9	4900	4900	100 %	95.54	-	0	0 %	83.12	13	14.5	100 %	115.76	-	0.07	1.75 %	92.29
(3,5,2)	10	16	18.5	100 %	12.56	12	10.29	90.50 %	12.81	12	14.5	100 %	17.8	-	1.6	19.07 %	13.6
(3,10,2)	12	418	433.13	100 %	16.06	66	81.75	100 %	15.07	24	27.63	100 %	30.77	-	0.38	4.50 %	24.72
(3,15,2)	17	3136	3175.57	100 %	45.75	-	6.92	58.23 %	41.59	27	32.92	100 %	83.4	-	0.6	6.32 %	66.32
(3,20,2)	18	8993	9015.33	100 %	176.73	-	0.08	1 %	159.09	31	41.01	100 %	190.43	-	1.02	8.45 %	171.93
(3,25,2)	21	18375	18386.1	100 %	518.85	-	0.13	1.62 %	460.67	36	46.15	100 %	417.79	-	1.83	11.70 %	437.92
(3,30,2)	23	32476	32478.81	100 %	2240	-	0.59	7.11 %	1443.95	38	58.66	100 %	890.21	-	3.35	14.58 %	844.48
(3,35,2)	24	52358	52359.83	100 %	5655.04	-	1.8	19.97 %	3675.48	43	66.68	100 %	1840.36	-	3.67	15.37 %	2256.05
(3,40,2)	25	79040	79040	100 %	12877.57	-	3.71	37.44 %	7753.23	53	77.91	100 %	3964.34	-	5.75	17.08 %	4464.97
(3,45,2)	26	113520	113520	100 %	21066.21	-	6.49	55.73 %	14769.39	57	89.7	100 %	6036.69	-	7.98	18.39 %	8025.81
(3,50,2)	28	156800	156800	100 %	27991.27	-	10.4	73.31 %	20511.06	58	102.18	100 %	9081.32	-	10.59	19.99 %	11604.85
(4,5,2)	16	16	16.06	100 %	12.58	-	7.78	48.62 %	14.04	16	16.12	100 %	42.79	-	7.48	46.75 %	31.67
(4,10,2)	24	683	693.75	100 %	32.58	295	309.69	100 %	33.81	66	72.47	100 %	234.82	-	3.25	16.90 %	193.21
(4,15,2)	35	11828	11932.32	100 %	424.01	192	226.11	100 %	380.76	81	108.84	100 %	989.4	-	6.3	16.57 %	891.06
(4,20,2)	39	70427	70630.34	100 %	7975.26	-	7.42	37.43 %	4344.41	127	178.55	100 %	3609.39	-	20.72	27.80 %	5326.14
(4,25,2)	50	200695	200787.83	100 %	41184.61	-	14.22	58.57 %	30182.05	206	282.13	100 %	13751.5	-	47.07	34.26 %	20131.28
(2,5,4)	16	122	129.28	100 %	10.81	-	1.58	9.61 %	10.02	31	33.68	100 %	18.94	-	0.4	2.49 %	12.98
(2,10,4)	24	716	719.36	100 %	14.13	-	0	0 %	13.1	34	37.14	100 %	25.44	-	0.02	0 %	18.09
(2,15,4)	26	1680	1680	100 %	20.6	-	0	0 %	19.37	39	43.37	100 %	38.33	-	0.03	0.19 %	28.51
(2,20,2)	28	3040	3040	100 %	33.08	-	0	0 %	30.17	45	49.06	100 %	63.42	-	0.02	0.12 %	46.81
(2,25,4)	29	4800	4800	100 %	56.52	-	0	0 %	51.76	48	54.52	100 %	91.46	-	0.03	0.19 %	76.2
(2,30,4)	30	6960	6960	100 %	88.25	-	0	0 %	80.45	52	59.17	100 %	137.83	-	0.03	0.19 %	121.21
(2,35,4)	31	9520	9520	100 %	145.11	-	0	0 %	132.45	58	64.31	100 %	230.66	-	0.19	1.08 %	203.92
(2,40,4)	32	12480	12480	100 %	208.62	-	0.03	0.19 %	186.45	65	68.95	100 %	429.77	-	0.2	1.25 %	347.63
(2,45,4)	32	15840	15840	100 %	299.18	-	0.01	0.06 %	270.83	66	73.15	100 %	636.45	-	0.61	3.19 %	564.27
(2,50,4)	34	19600	19600	100 %	405.47	-	0.15	0.94 %	361.93	73	77.74	100 %	795.45	-	0.91	4.54 %	871.32
(3,5,4)	64	337	355.74	100 %	13.11	-	17.93	25.42 %	14.77	167	176	100 %	152.98	-	0.43	0.67 %	177.44
(3,10,4)	100	7581	7605.03	100 %	82.12	-	0	0 %	79.59	215	230.46	100 %	691.48	-	0.4	0.57 %	852.02
(3,15,4)	124	29117	29118.79	100 %	471.23	-	0.6	0.93 %	438.64	284	310.71	100 %	1744.85	-	1.54	1.56 %	2424.33
(3,20,4)	127	72959	72959.99	100 %	1859.71	-	3.03	4.61 %	1510.5	374	420.7	100 %	3999.48	-	4.74	2.72 %	5357.55
(3,25,4)	152	147200	147200	100 %	6457.17	-	9.51	13.84 %	5027.45	474	541.57	100 %	8442.67	-	8.47	3.32 %	11283.04
(4,5,4)	256	524	542.77	100 %	16.06	-	14.55	5.68 %	14.03	531	544.19	100 %	3690.22	-	15.97	6.23 %	3765.39
(4,10,4)	496	50243	50357.71	100 %	847.43	-	1.85	0.72 %	783.32	1234	1317.55	100 %	41309.44	-	14.13	3.45 %	45823.46
(4,15,4)	508	349263	349290.61	100 %	12760.52	-	27.62	10.22 %	10226.22	2078	2359.66	100 %	174386.99	-	72.86	6.60 %	169873.67

Table 3.2: Results of experiments with Configurations of Algorithm 6, using weight updates.

(t, k, v)	[97]	RandomGraph + BMweight				RandomGraph + HDweight				HeavyGraph + BMweight				HeavyGraph + HDweight			
		min	*avg*	*avg cov*	*time*	*min*	*avg*	*avg cov*	*time*	*min*	*avg*	*avg cov*	*time*	*min*	*avg*	*avg cov*	*time*
(2,5,2)	6	6	6.6	100 %	4633.4	6	6.4	100 %	4040.8	6	6.43	100 %	4178.33	6	6.51	100 %	4001.32
(2,10,2)	6	10	11.7	100 %	5053.3	9	11.9	100 %	5797.8	7	8.2	100 %	4471.93	7	8.1	100 %	4125.39
(2,15,2)	7	13	15.2	100 %	6225.8	14	14.8	100 %	5848.6	8	9.31	100 %	4005.75	8	9.28	100 %	4264.84
(2,20,2)	8	14	16.7	100 %	5998.3	15	17.5	100 %	6015.1	9	10.33	100 %	3796.28	10	10.37	100 %	4008.39
(2,25,2)	8	17	19.2	100 %	8332.3	17	19	100 %	8327.4	10	10.79	100 %	4184.55	10	10.8	100 %	4348.13
(2,30,2)	8	18	19.9	100 %	11910.9	18	19.5	100 %	10229.3	10	11.56	100 %	3834.46	10	11.32	100 %	3970.65
(2,35,2)	8	19	21.6	100 %	14872.2	20	23.7	100 %	15699	10	12.01	100 %	3898.28	10	11.87	100 %	3708.65
(2,40,2)	9	20	23.4	100 %	17679.7	20	21.9	100 %	23085.6	12	12.49	100 %	3773.32	11	12.55	100 %	3832.58
(2,45,2)	9	23	24.5	100 %	25971.5	22	23.6	100 %	41323.7	12	13.02	100 %	4378.52	12	12.93	99.25 %	4114.76
(2,50,2)	9	22	24.7	100 %	33196.8	23	27.5	100 %	31496.8	12	13.24	100 %	4263.93	12	13.29	99.25 %	3809.29
(3,5,2)	10	13	14	100 %	6684.8	12	13	100 %	4407.2	11	12.83	100 %	4411.91	10	12.43	100 %	4634.01
(3,10,2)	12	30	33.6	100 %	9837.3	29	34.2	100 %	11303.6	20	23.25	100 %	5287.62	20	23.13	99.12 %	5959.37
(3,15,2)	17	48	50.5	100 %	21131.8	43	48.6	100 %	20608.8	25	31.06	100 %	3746.05	25	14.73	67.95 %	3841.56
(3,20,2)	18	52	58.6	100 %	74868.4	50	54.6	100 %	89159.2	28	39.93	100 %	3619.01	29	16.62	52.06 %	3485.61
(3,25,2)	21	58	67.3	100 %	182276.1	55	69.5	100 %	179734.8	33	47.8	100 %	3830.79	33	12.5	38.31 %	3835.88
(3,30,2)	23	73	91	100 %	222202.8	63	97.9	100 %	306923.4	41	55.99	100 %	4339.64	39	7.96	25.82 %	4711.01
(3,35,2)	24	74	93	100 %	342334.2	64	113.6	100 %	683419.1	43	65.28	100 %	5684.17	42	8.19	23.45 %	5928.71
(3,40,2)	25	91	96.4	100 %	560322.2	72	118.3	100 %	783339.3	46	74.24	100 %	7615.68	44	8.42	22.52 %	7408.09
(3,45,2)	26	75	86.3	100 %	822433.4	71	150.6	100 %	903760.4	50	83.97	100 %	10230.35	49	9.76	22.81 %	10444.14
(3,50,2)	28	78	96.2	100 %	1186650.7	77	171	100 %	1332173.8	55	96.73	100 %	14512.67	56	14.33	27.91 %	15252.87
(4,5,2)	16	16	16	100 %	3688	16	16	100 %	3927	16	16	100 %	4943.98	16	16	100 %	4985.24
(4,10,2)	24	74	90.9	100 %	30696.7	74	90.2	100 %	15913.7	54	71.68	100 %	3979.42	68	9.45	25.13 %	3703.68
(4,15,2)	35	107	115.4	100 %	219030.7	135	183.6	100 %	52291.5	73	107.74	100 %	4615.68	85	11.51	22.90 %	4734.98
(4,20,2)	39	138	153.6	100 %	2200402.2	195	227.6	100 %	464624.8	99	173.03	100 %	8308.19	-	22.36	28.75 %	8269.93
(4,25,2)	50	196	216.5	100 %	4010665.3	198	239.7	100 %	1588540.9	185	282.32	100 %	22218.11	-	46.34	34.36 %	20637.51
(2,5,4)	16	42	45.7	100 %	4148.2	39	42.9	100 %	4724.1	29	30.65	100 %	4445.55	28	30.8	100 %	5212.62
(2,10,4)	24	74	83.6	100 %	8293.8	63	76.9	100 %	6646.5	32	34.72	100 %	5340.54	32	34.55	100 %	5339.83
(2,15,4)	26	92	102.5	100 %	9354.3	93	97.6	100 %	12136.8	37	42.69	100 %	3855.09	37	42.01	98.12 %	4221.35
(2,20,2)	28	108	123.5	100 %	13060.9	101	109.7	100 %	18092	45	49.38	100 %	3279.07	44	48.94	99.06 %	3260.6
(2,25,4)	29	114	122.5	100 %	39982.5	111	121.9	100 %	36189.6	49	54.17	100 %	3278.72	48	53	97.19 %	3137.63
(2,30,4)	30	116	139.9	100 %	43517.8	123	133.4	100 %	46914.3	54	59.76	100 %	3333.11	54	56.98	95.31 %	3541.37
(2,35,4)	31	147	159.1	100 %	56594.2	127	142.7	100 %	80056.3	58	65.04	100 %	3825.17	63	53.7	88.81 %	3670.79
(2,40,4)	32	145	167.1	100 %	68871.5	135	164.8	100 %	74721.9	63	68.35	100 %	3841.98	62	50.28	75.65 %	3914.73
(2,45,4)	32	133	174.6	100 %	83542.7	145	182	100 %	92931.4	66	73	100 %	3981.13	65	37.74	56.05 %	3975.47
(2,50,4)	34	175	188.2	100 %	100347.6	138	203.6	100 %	108361.8	71	76.62	100 %	4224.94	69	18.37	28.96 %	4348.84
(3,5,4)	64	198	203.2	100 %	10394.3	156	29.7	30.44 %	3853.4	148	159.74	100 %	6573.35	148	23.64	24.37 %	4564.58
(3,10,4)	100	1000	1057.7	100 %	105703	-	5	7.31 %	23866	218	230.46	100 %	4548.27	226	12.2	8.48 %	4754.21
(3,15,4)	124	4507	6609.4	100 %	281754.8	-	1.1	1.72 %	101754.7	283	312.4	100 %	6381.09	-	24.74	15.07 %	6321.32
(3,20,4)	127	11809	11945.1	100 %	1715645.5	-	2.4	3.69 %	311706.4	382	420.78	100 %	8877.50	-	5.18	3.83 %	8883.95
(3,25,4)	152	8880	20882.9	100 %	4428809.2	-	8.2	12.11 %	713349.5	474	543.45	100 %	14216.36	-	8.07	3.25 %	14071.41
(4,5,4)	256	492	499.6	100 %	16143	-	15.2	5.94 %	4144.5	487	500.88	100 %	24586.93	-	16.03	6.26 %	8799.66
(4,10,4)	496	11784	25330	100 %	1217777	-	1.5	0.59 %	163020	1234	1316.84	100 %	50113.58	-	16.18	3.73 %	47691.89
(4,15,4)	508	timeout	timeout	timeout	timeout	-	23.5	8.77 %	1676888.1	2103	2335.28	100 %	189176.41	-	79.66	6.86 %	182037.10

Experimental Evaluation of Algorithm BMforCAlearning, Algorithm 6

To evaluate the efficiency of the introduced weight update in combination with the different initial graphs and weightings, we compare the same configurations as presented in Table 3.1, now using weight updates, i.e. as configurations of Algorithm 6. The results can be found in Table 3.2, which headings present information in the same way as Table 3.1. We have conducted again 100 individual runs for the configurations using HeavyGraph, but reduced the number to 10 runs for the configurations using RandomGraph due to their excessive run-time compared to the configurations using HeavyGraph.

First and foremost it is remarkable that for almost all instances the deployed learning in form of weight updates almost nullified the severe difference in the number of rows of generated CAs when comparing the results to those given in Table 3.1. The configurations using HDweight are capable of finding CAs for many instances compared to almost non without the learning mechanisms via weight updates. For the configurations using BMweight we can see that the one using RandomGraph significantly reduces the number of rows of the smallest generated CA and the same holds for the average number of rows of all generated arrays which all cover 100 % of the t-way interactions. The configuration HeavyGraph + BMweight again performs best, generating the smallest CAs amongst all four configurations for almost all CA instances. Further we can see that the configuration HeavyGraph + BMweight with weight updates (Algorithm 6) improves upon the results when using no weight updates (Algorithm 4) for many instances. However, it is surprising that for some instances it performs slightly worse with regards to smallest CAs found when compared to the same configuration of Algorithm 4. This phenomenon can be also recognized in a weakened form for the average number of rows of the returned arrays (which are all CAs for both Algorithms), see also Figures 3.9 and 3.10. One possible explanation can be given in the form of statistical outliers, since Algorithm 6 represents an algorithmic refinement of Algorithm 4.

A visual representation of selected results of Table 3.2 is given in Figure 3.7 and 3.8, where the minimum number of rows, for which the four configurations successfully found CAs is depicted.

Experimental Evaluation of Algorithm BMforCAlearningGraph, Algorithm 7

To evaluate the procedure GraphUpdate, we compare three different configurations of Algorithm 7 amongst each other and also with configurations of Algorithm 4 and 6. The three configurations all use BMweight as initial weighting for the edge weights. Further, the first configuration uses the RandomGraph to instanciate InitialGraph and the BestEdges procedure to instantiate GraphUpdate. Analogue, the remainig two configurations use RandomGraph in combination with HeavyEdges and HeavyGraph in combination with HeavyEdges, respectively (see the headings of Table 3.3).

For each configuration, we conducted 10 runs for each CA instance, where we limited the number of learning phases to 20. Each learning phase contained 100 epochs and a bias

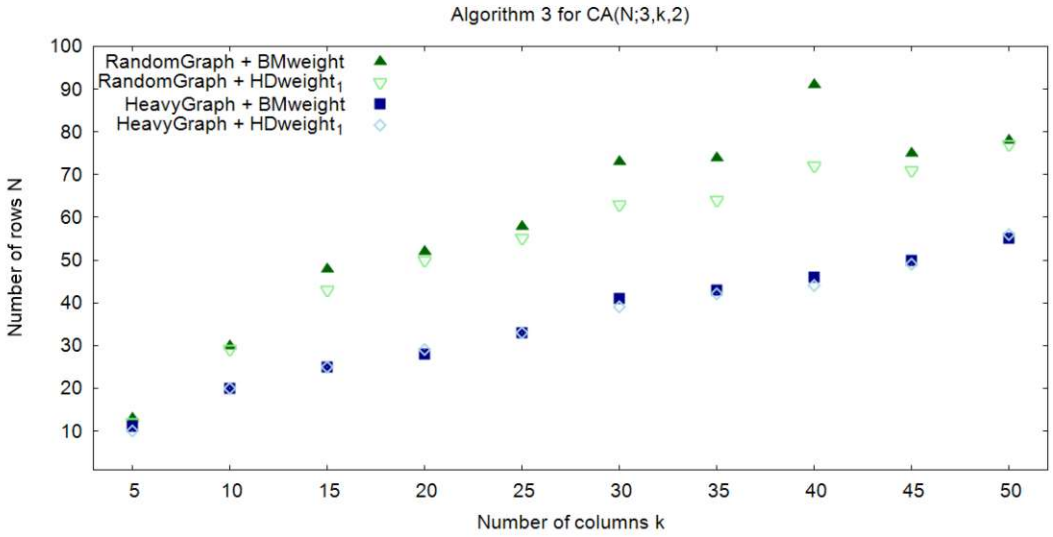


Figure 3.7: Minimal number of rows (smaller is better) of generated CAs by configurations of Algorithm 6 applied to the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$. Not all configurations can produce results in reasonable time. Note the vertical axis is in linear scale.

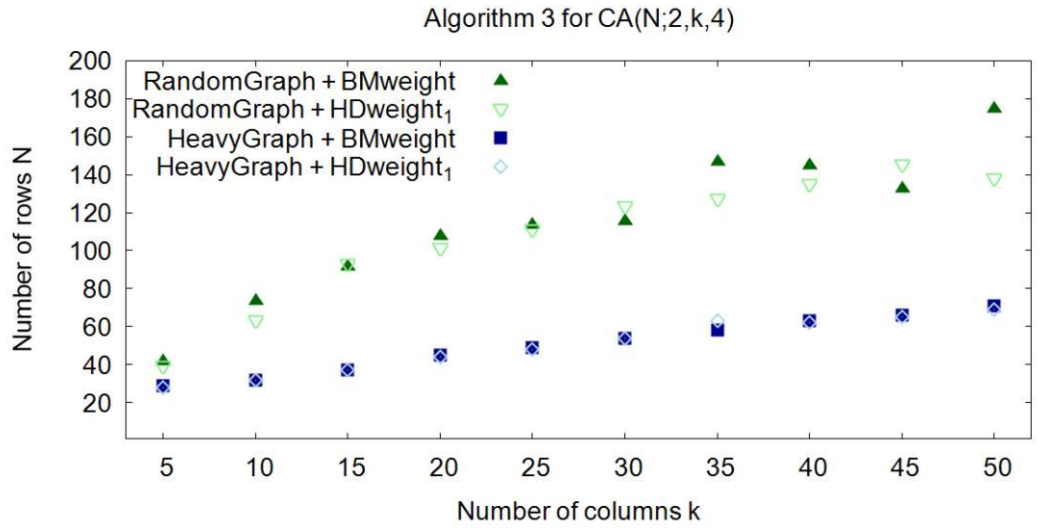


Figure 3.8: Minimal number of rows (smaller is better) of generated CAs by configurations of Algorithm 6 applied to the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. Only two configurations can produce results in reasonable time. Note the vertical axis is in linear scale.

update at the end, where the weight of the vertices (i.e. rows) in $\mathcal{I}_{learn}$ is increased by 0.01. For the BestEdges graph update, 80% of the rows in $V \setminus \mathcal{I}_{max}$ were selected to be included in the new graph, and 50% for the HeavyEdges graph update. The results of these experiments are given by the second to fourth column of Table 3.3.

First, comparing the results attained by Algorithm 6 to those of Algorithm 7 for the configurations using RandomGraph, we can see that by means of the updates of the graph, all results for the average number of rows of generated CAs are improved (comparing the *avg* data of the second and third column of Table 3.3 against the second and third column of Table 3.2). The same holds for the minimal number of rows (the *min* data) of these configurations, with the exception of the CA instance $(3, 45, 2)$, where the RandomGraph] + HeavyEdges configuration of Algorithm 7 performs slightly worse than the RandomGraph + HDweight configuration of Algorithm 6.

Second, the graph update can level out differences in InitialGraph. The results in Table 3.3 demonstrate that via the HeavyEdges graph update the gap in the average number of rows of returned CAs of configurations using RandomGraph compared to configurations using HeavyGraph is reduced. This can be seen in Table 3.3 since the difference in the *avg* data of column three and four for many cases is much smaller than the difference in the *avg* data of columns two and four in Table 3.2.

Third, the configuration using HeavyGraph + HeavyEdges performs overall better then the two configurations using Randomgraph, yielding slightly larger values for *min* in only five CA instances, and the smallest values for *avg* for all but the CA instance $(4, 5, 4)$.

Last, the graph update improves on the best solutions previously found. Comparing the configuration HeavyGraph + HeavyEdges of Algorithm 7 to the best configuration of Algorithm 6 using HeavyGraph + BMweight (fourth column of Table 3.2), it produces the same or smaller values for the average number of rows of returned CAs for all but four CA instances, reducing this value significantly in many cases. Similarly, the values of the minimal number of rows of generated CAs is equal or better in all but three CA instances, namely $(3, 5, 2)$, $(2, 50, 4)$ and $(4, 10, 4)$.

To visualize the comparison of Algorithms 4, 6 and 7, we compare the third columns of the Tables 3.1-3.3 and visualize the results for the minimal number of rows of generated CAs for selected CA instances. Figure 3.9 gives a comparison by means of the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$, showing the improvement of the results by means of the learning mechanisms.

The same claim is not supported by the results presented in Figure 3.10, which gives a comparison by means of the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. We do believe that this is due to the increased complexity of the optimization problem and too little computation time so that Algorithm 7 can benefit from the learning mechanisms.

Table 3.3: Results of experiments with different configurations of Algorithm 7, using graph updates and Algorithm 8. All configurations of Algorithm 7 use BMWeight as initial weighting and the remaining buidling blocks as they are indicated by the column headings.

(t, k, v)	[97]	RandomGraph + BestEdges			RandomGraph + HeavyEdges			HeavyGraph + HeavyEdges			Hopfield	
		min	*avg*	*time*	*min*	*avg*	*time*	*min*	*avg*	*time*	*min*	*time*
(2,5,2)	6	6	6	118048.6	6	6	100288.9	6	6	93452.9	6	2
(2,10,2)	6	8	8.8	118586.4	7	7.8	114149.9	7	7.7	106647.3	18	1388117
(2,15,2)	7	10	11.2	119926	8	8.9	99048.8	8	8.9	101595.5	OOM	OOM
(2,20,2)	8	12	12.9	121629.7	9	9.6	111081.4	9	9.4	100070.6	OOM	OOM
(2,25,2)	8	13	14.4	117723.4	10	10.8	97233.8	10	10.4	100091.7	OOM	OOM
(2,30,2)	8	14	15.4	113215.7	10	11.1	101894.4	10	11	91630	OOM	OOM
(2,35,2)	8	16	17.5	116205.2	11	11.8	98593.6	11	11.3	86739	OOM	OOM
(2,40,2)	9	16	17.8	130717.2	12	12.4	108843.7	11	11.8	85097.8	OOM	OOM
(2,45,2)	9	18	19	125033.6	12	12.7	115656.8	12	12.5	88437.9	OOM	OOM
(2,50,2)	9	19	20.2	129834.1	12	13.3	125091.1	12	12.9	86875	OOM	OOM
(3,5,2)	10	12	12.6	168484	11	11.9	109999.1	11	11.6	106576.8	12	2
(3,10,2)	12	25	26.7	163939.8	21	21.5	145504.6	21	21.4	130968.5	20	1401025
(3,15,2)	17	37	38.8	141479.4	28	29.5	118089.9	26	28.6	92918	OOM	OOM
(3,20,2)	18	43	49.4	235079.8	31	34.4	159904.3	32	33.2	87346.3	OOM	OOM
(3,25,2)	21	55	56.5	368123.7	35	40.3	295587.8	33	43.9	81936.9	OOM	OOM
(3,30,2)	23	55	61.3	457756	39	55.6	323257.6	41	51	105323.2	OOM	OOM
(3,35,2)	24	59	65	582069.7	58	69	434577	48	58.5	129965.4	OOM	OOM
(3,40,2)	25	63	67.8	731733.5	64	74	643408.5	49	64.4	152410.4	OOM	OOM
(3,45,2)	26	66	68.4	1287654.2	73	83.8	962741.8	48	75.4	177409.2	OOM	OOM
(3,50,2)	28	69	72.8	1819620.1	71	85	1478497.6	58	79.2	239133.1	OOM	OOM
(4,5,2)	16	16	16	104074.7	16	16	96597.6	16	16	95401.8	16	1
(4,10,2)	24	62	65.8	347828	54	64.1	150293.5	54	62.4	86364.7	142	10231824
(4,15,2)	35	103	108.9	993661.5	93	101.3	380625.9	79	97	94670.3	OOM	OOM
(4,20,2)	39	127	142.9	3524728.6	133	147.3	2059153.8	131	148.7	144303.4	OOM	OOM
(4,25,2)	50	timeout	timeout	timeout	timeout	timeout	timeout	172	210.7	305028.3	OOM	OOM
(2,5,4)	16	34	36.6	122896.9	29	30.2	121478.5	29	29.6	117223.9	16	3013585
(2,10,4)	24	60	66.2	98649.5	33	34.7	137437.9	33	34.1	122448.2	OOM	OOM
(2,15,4)	26	76	80.8	98350.5	38	43.1	99137.4	39	41	86292.1	OOM	OOM
(2,20,2)	28	81	88.9	133951.3	46	48	105915.9	44	47.5	78778	OOM	OOM
(2,25,4)	29	94	98.1	147574.3	53	55.5	125413.1	49	52.9	80489.6	OOM	OOM
(2,30,4)	30	95	101.4	143915	54	58.4	151920.9	55	58	90585.5	OOM	OOM
(2,35,4)	31	105	109.3	181628.6	59	65.4	154673	58	61.9	90239.6	OOM	OOM
(2,40,4)	32	107	114.9	221983.5	65	73.3	164472.8	66	70.3	94770.6	OOM	OOM
(2,45,4)	32	114	122.1	281603.8	75	78.8	182032.7	68	74.7	96708.3	OOM	OOM
(2,50,4)	34	123	126.9	362497.1	87	97	213029	76	78.7	99346.8	OOM	OOM
(3,5,4)	64	152	160.8	183354.7	140	144.8	153161	143	147.8	166837	64	497555.6
(3,10,4)	100	411	435.2	270418.6	253	257.3	231793	223	228.3	105902.2	OOM	OOM
(3,15,4)	124	491	516.1	884304.7	343	364.1	562810.7	285	312.9	135037.3	OOM	OOM
(3,20,4)	127	542	565.3	2997659.1	482	494.2	2186128.4	379	405	171550.1	OOM	OOM
(3,25,4)	152	timeout	timeout	timeout	timeout	timeout	timeout	486	541.9	234513.9	OOM	OOM
(4,5,4)	256	473	477.8	487389.2	476	479.4	542601.7	476	478.5	555324	256	81254.4
(4,10,4)	496	1986	2133	2311494.7	1445	1484.8	1736530.3	1262	1343	571346.7	OOM	OOM
(4,15,4)	508	timeout	timeout	timeout	timeout	timeout	timeout	2096	2313.4	1849341.1	OOM	OOM

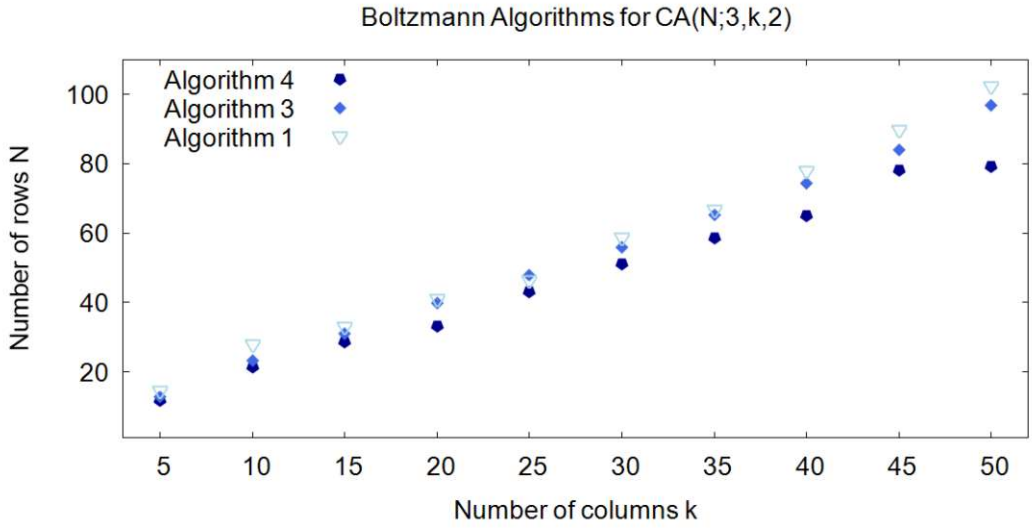


Figure 3.9: A comparison of Algorithms 4, Algorithm 6 and Algorithm 7, each configuration involving HeavyGraph and BMweight and Algorithm 7 using HeavyEdges in addition, by means of the minimal number of rows of generated CAs for the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$. Note the vertical axis is in linear scale.

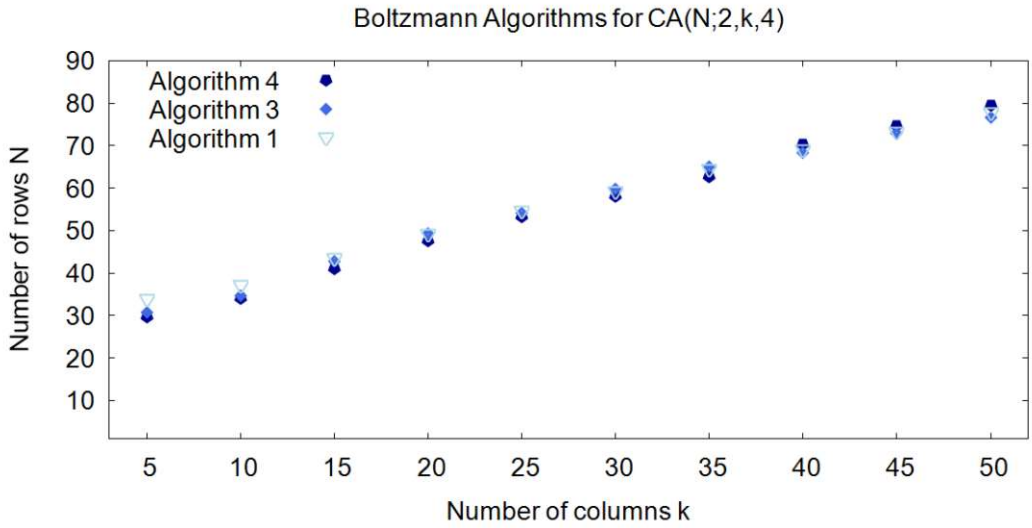


Figure 3.10: A comparison of Algorithms 4, Algorithm 6 and Algorithm 7, each configuration using HeavyGraph and BMweight and Algorithm 7 using HeavyEdges in addition, by means of the minimal number of rows of generated CAs for the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. Note the vertical axis is in linear scale.

Finally, the graphs in Figure 3.11 showcase the learning progress of the networks over multiple learning phases. For two different CA instances, the evaluation of the best found

solution after each learning phase, normalized to the optimal solution, i.e. $\mathsf{CAN}/|V \setminus \mathcal{I}_{max}|$, is depicted. Using this scoring method, larger values are better. We chose this function because it is well suited to visualize the increasing learning curves of the neural networks. Our experiments show that for both instances the configuration using the procedure HeavyEdges finds an optimal CA faster than the configuration using BestEdges.

To conclude this part of the experimental evaluation, we remark that tuning the parameters of neural networks for search problems has been subject to a number of related works (e.g. with *genetic algorithms* [265] or combinatorial approaches [266], [267]). However, a further evaluation in that direction is considered to be beyond the scope of this thesis.

Comparison Between Boltzmann and Hopfield Networks

Now, we compare the results of the Boltzmann machine with the results achieved by the Hopfield network. Despite the algorithm using Hopfield networks producing deterministic results, we conducted 10 runs for each CA instance to calculate the average run time. The results of these experiments are given in the column headed by Hopfield in Table 3.3. For the Hopfield network the values of the constants λ, y and δ were specified as in [248].

The results in Table 3.3 show clearly that the algorithm using Hopfield networks suffers from the combinatorial explosion of the search problem. This can be explained by the underlying graph of the network being the complete graph $G_{t,k,v}$, which shows the necessity of a meaningful graph reduction as used in the algorithms based on Boltzmann machines.

Closing, in Figure 3.12 we compare the run time of the three algorithms on a logarithmic scale in milliseconds, which also depicts the bad scalability of Hopfield networks nicely. The run times of the algorithms using Boltzmann machines scale much better as we consider reduced graphs of *type II*. These reduce heavily the density of the graph underlying the ANN, which reduces the gradient of the run time curve.

The graphs also show that the runtime of experiments with algorithms using learning mechanisms (Algorithm 6 and 7) does not increase as much with the size of the instances compared to the algorithm using no learning mechanisms (Algorithm 4).

Generating Mixed-level and Constrained Covering Arrays with Neural Networks

In this subsection in order to show that there is potential for ANNs to find application in test set generation for applied combinatorial testing, we provide some elementary experiments for the generation of CMCAs. In real-world testing scenarios, systems under test can not always be modeled exclusively by homogeneous parameters and may also involve constraints. Thus, the following described basic experiments demonstrate that ANNs can be applied to generate structures that can be used for combinatorial testing of a broad spectrum of real-world systems.

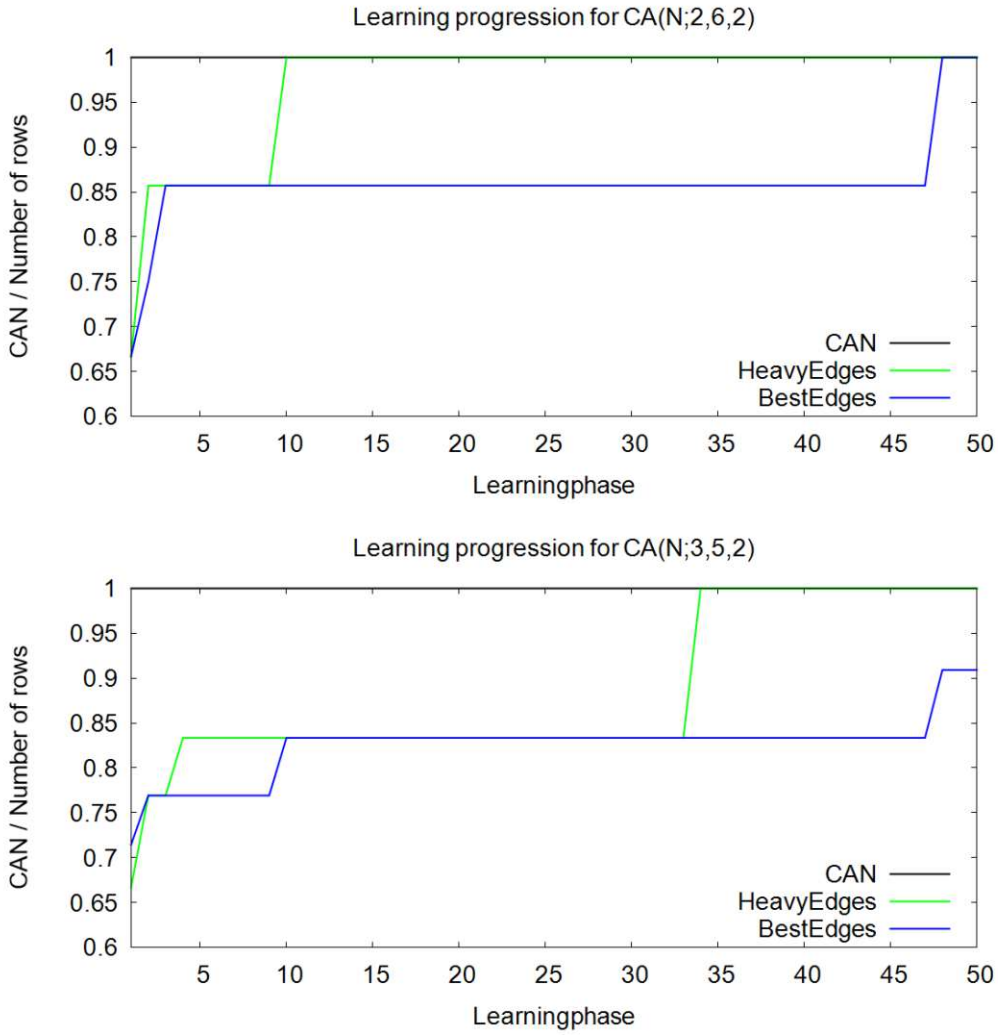


Figure 3.11: The learning progression, evaluated by means of $\mathsf{CAN}/|V \setminus \mathcal{I}_{max}|$, of the two different GraphUpdate procedures for the CA instances $(t = 2, k = 6)$ and $(t = 3, k = 5)$. The growth of $\mathsf{CAN}/|\mathcal{I}_{max}|$ is shown on the vertical axis over the learning phases, on the horizontal axis.

We extracted IPMs for combinatorial testing for some sample systems under test as they are given in the ACTS-tool [117]. These models specify the alphabet sizes of the columns and the constraints that need to be satisfied by a CMCA that can be used for combinatorial testing of the respective system. We generated these CMCAs with Algorithm 7, which was configured the same way as for the experiments documented in the fourth column of Table 3.3, i.e. BMweight + HeavyGraph + HeavyEdges.

The results of our experiments are given in Table 3.4, where we present the smallest number of rows of generated CMCAs over 100 runs for each CMCA instance. In the

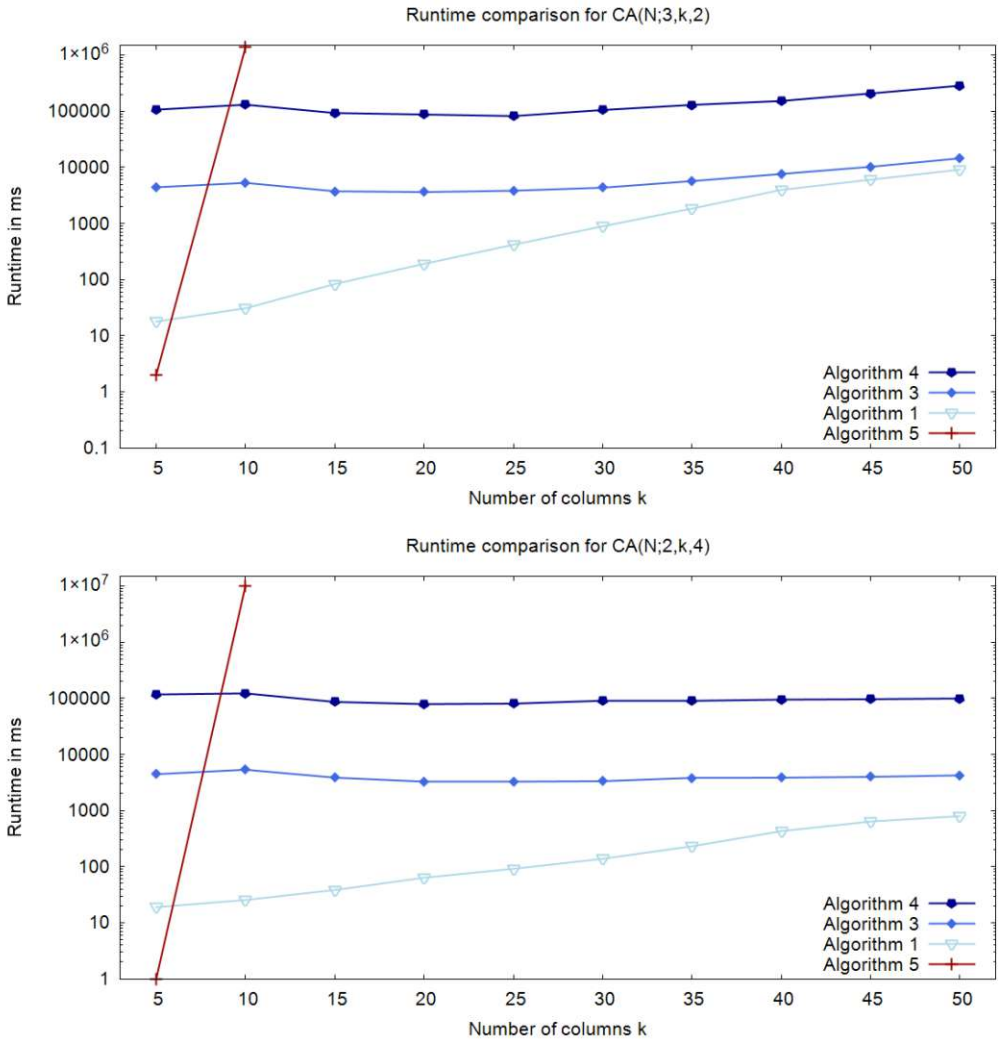


Figure 3.12: The runtime in seconds (vertical axis) of the different algorithms is depicted for different values of k on a logarithmic scale.

first column headed by "SUT" we give the (abbreviated) name of the system under test and in the second column we give the alphabet sizes $(v_1, \ldots, v_k)$ of the columns of the CMCA, where we use an exponential notation, e.g. 10^2 means that there are two columns with alphabet size 10. Further, the column headed by "#Constraints" denotes the number of constraints as they are given in form of first-order logic formulas. The column headed by "#MFT" denotes the number of *minimum forbidden tuples*, which can informally be described as the essential s-way interactions that are forbidden (see [268] for details). We compare the results of Algorithm 7 against CAgen, a state of the art tool for combinatorial test set generation, presented in [78]. On one hand, it is notable that the ANN approach even improves slightly on the CAgen-tool in two instances for

SUT	$(v_1, \ldots, v_k)$	#Constraints	#MFT	Algorithm 7		CAgen [78]	
				$t = 2$	$t = 3$	$t = 2$	$t = 3$
TCAS	$10^2, 3^2, 2^7$	3	6	100	487	101	411
SPIN-S	$4^5, 2^{13}$	13	13	25	136	26	109
bugzilla	$4, 3, 2^{49}$	5	5	19	168	18	65

Table 3.4: Experiments for constrained mixed-level covering arrays for real-world combinatorial testing models.

$t = 2$. On the other hand, considering that with the SA Algorithm 5 a meta-heuristic method is underlying the presented ANN approach, it is less surprising, bethinking that meta-heuristic methods are well suited for small and medium sized CA instances, as is also mentioned in [98]. In Section 3.4 we will hybridize the algorithm underlying the CAgen-tool with a quantum-inspired meta-heuristic. The resulting algorithm is able to improve in several instances on the CAgen-tool, as we will see.

3.3 The sliced AETG Algorithm

This section is based on the publication [138] and documents the investigation of two variants of a well known greedy strategy for CA construction. The underlying idea is to partition a given CA generation problem into smaller parts, which can then be dealt with sequentially or in parallel. We further give upper bounds for the number of rows of the CAs produced by these algorithms, which have the same asymptotic as the greedy strategy they originate from, assuming that the alphabet size v and the strength t are constants. Similar statements for the runtimes of the algorithms are given. Practical applications, such as [269] and [146] highlight the need for methods capable of generating CAs with a large number of columns. This is an additional motivation for the investigation of methods that allow to extend the usage of known CA generation methods, for example by more memory efficient construction methods.

In the following we start by revisiting the greedy AETG algorithm for CA generation, representing the basis for the proposed algorithmic variants. Thereafter we propose our algorithms, describe some of their properties and give an experimental evaluation of them.

Notation. In this section we make use of the following additional notation: the function $\varphi_{v,k,t} : [v]^{N\times k} \mapsto \mathbb{T}_{v,k,t}$ maps an array (or a row in case of $N = 1$) to the set of t-way interactions covered by the array (recall Definition 1.3.10), more formally:

$$\varphi_{v,k,t}(A) := \{\tau \in \mathbb{T}_{v,k,t} : \exists \text{ row } r \text{ in } A, \text{ and } r \text{ covers } \tau\}.$$

3.3.1 Review of Greedy AETG

First we describe a well-known algorithm for CA construction, originally treated in a specialized version in Subsection 3 of [36] and more detailed , interpreted as a greedy

heuristic algorithm for CA generation descending from the connection to minimal set covers in [144], as reviewed in this thesis in Section 2.3. Since this greedy algorithm serves as a starting point for the development of a commercial tool called AETG, we refer to it as greedy AETG (gAETG for short) and describe it by means of a pseudo-code in Algorithm 9. This algorithm initializes the CA A to be constructed as an empty array. It iteratively and greedily selects a row $r \in [v]^k$ which covers the most t-way interactions uncovered by A and adds it to the array.

Algorithm 9 gAETG (for CAs)

```
INPUT: t, k, v
Require: t ≤ k
A ← ∅                                              ▷ Initial array empty
T ← 𝕋_{v,k,t} (the set of all t-way interactions)  ▷ Initial set of tuples
R ← [v]^k                                          ▷ R is set of rows
while T ≠ ∅ do
    determine r = arg max_{r∈R} |φ_{v,k,t}(r) ∩ T|
    A ← A ∪ {r}                                    ▷ Append row r to A
    T ← T \ φ_{v,k,t}(r)
end while
return A
```

Although very simple, Algorithm 9 is of theoretical interest, since it provides a constructive proof that $\mathsf{CAN}(t,k,v)$ is in $O(\log k)$ for fixed t and v, as was shown in [36]. As mentioned in the introduction (Subsection 1.4.2) this result was later shown via a different algorithm in [71]. The argument in 9 was carried out for CAs with homogeneous alphabets of strength two, and generalized for MCAs of higher strength in [144]. We restate the required result in the following corollary of Theorem 2.2.2, which follows immediately from the inequality (2.5), and that $h = v^t$ in the uniform case.

Corollary 3.3.1 *The number of rows N of the returned array in Algorithm 9 is upper bounded by $v^t \log(v^t \binom{k}{t}) + 1$, and hence for fixed strength t and alphabet size v, in $O(\log k)$.*

Lemma 3.3.2 *For a given $r \in [v]^k$ and a set $T \subseteq T_{v,k,t}$ of t-way interactions, the computation of $|\varphi_{v,k,t}(r) \cap T|$ can be performed in $O(t^2 \binom{k}{t})$ time. Under the assumption that elementary arithmetic operations can be performed in $O(1)$ time.*

Proof: We first describe how the set of t-way interactions $\mathbb{T}_{v,k,t}$ can be represented by an incidence structure that can be accessed efficiently. This can be done by means of a ranking function that maps $\tau \in \mathbb{T}_{v,k,t}$ to an integer, which can be used to specify the position of τ in an ordering of the elements of $\mathbb{T}_{v,k,t}$. Such a function can be realized by a combination of a ranking function for the t-tuple of values $(u_1, \ldots, u_t)$ of $\tau = \{(p_1, u_1), \ldots, (p_t, u_t)\}$, together with a ranking function for the combination of column indices $(p_1, \ldots, p_t)$ of τ. The function $f(\tau, t, k, v) := v^t \cdot \sum_{i=1}^{t} (-1)^{t-i} \left(\binom{p_i}{i} - 1\right) + \sum_{i=1}^{t} v^{i-1} u_i$ is such a ranking function onto the integer interval $[0, v^t \binom{k}{t} - 1]$, since

it is known that tSubsetRevDoorRank$((p_1,\ldots,p_t),t) := \sum_{i=1}^{t}(-1)^{t-i}\left(\binom{p_i}{i}-1\right)$ is a ranking function for t-subsets (of a k element set) [165, Alg. 2.11], as well as, that tTupleRank$((u_1,\ldots,u_t)) := \sum_{i=1}^{t} v^{i-1}u_i$ is a bijection from $\{0,\ldots,v-1\}^t$ to $[0,v^t-1]$. From the t appearing binomial coefficients $\binom{p_i}{i}$, $\binom{k}{t}$ would be the most complex to compute, which can be computed as $\prod_{i=1}^{t}(k-j+1)/j$ in $O(t)$, under the assumption that each elementary arithmetic operations can be performed in $O(1)$ time. Thus, $f(\tau,t,k,v)$ can be computed in $O(t^2)$.

Assuming T is initialized already in line 3 of Algorithm 9, e.g. as a binary vector of length $v^t\binom{k}{t}$ representing the incidence vector of the set T. In order to compute $|\varphi_{v,k,t}(r)\cap T|$, we can iterate over the $\binom{k}{t}$ t-way interactions of $\varphi_{v,k,t}(r)$, compute $f(\tau)$ and check in the binary vector representing T if τ is covered or not. In case it is not covered, an incremental counter for $|\varphi_{v,k,t}(r)\cap T|$ can be increased. The comparison and the increase of the counter can be done in constant time $O(1)$. This way the computation of $|\varphi_{v,k,t}(r)\cap T|$ can be performed in $O(t^2\binom{k}{t})$ time. □

Theorem 3.3.3 *The runtime of Algorithm 9 is in* $O\left(v^{k+t}t^2\binom{k}{t}\log(v^t\binom{k}{t})\right)$.

Proof: Provided Lemma 3.3.2, the number of steps required for finding the row r in line 6 of Algorithm 9 is in $O(v^k t^2\binom{k}{t})$, as for each row $r\in[v]^k$ we can check the membership in T of the $\binom{k}{t}$ different t-way interactions covered by r in $O(t^2\binom{k}{t})$ time. The time required by the statements in lines 7 and 8 is dominated by $O(v^k t^2\binom{k}{t})$. From Corollary 3.3.1 we get that the condition $T\neq\emptyset$ in line 5 is satisfied at most $v^t\log(v^t\binom{k}{t})+1$ times. Thus we have a runtime in $O\left(v^{k+t}t^2\binom{k}{t}\log(v^t\binom{k}{t})\right)$. The time for initialization of T as $\mathbb{T}_{v,k,t}$ is in $O(v^t\binom{k}{t})$ and is dominated by other terms. □

3.3.2 Memory Saving Variants of Greedy AETG

In this section we introduce two new algorithms based on Algorithm 9. The underlying idea common to both these algorithms is to partition the set of t-way interactions $\mathbb{T}_{v,k,t}$ into disjoint parts, or *slices*, and treat these slices iteratively or individually. As will be shown, the time complexity respective to the number of columns k of both these algorithms is the same as that of Algorithm 9 and the generated CAs have a number of rows in $O(\log k)$. For the remainder of this paper we denote by $T_u := \{\{(p_1,v_1),\ldots,(p_t,v_t)\}\in\mathbb{T}_{v,k,t}|v_1=u\}$ the set of t-way interactions having the entry $u\in[v]$ as first position. Similarly, we denote by $R_u := \{u\}\times[v]^{k-1}$ the rows of length k having u in first position.

Sliced AETG: A Sequential Variant of Greedy AETG

The slicedAETG algorithm is presented in Algorithm 10 and can be summarized as follows. The core of the algorithm, i.e. lines 7-11, corresponds to the core of the gAETG algorithm. The main difference is that the set of t-way interactions $\mathbb{T}_{v,k,t}$ is processed in disjoint parts, or *slices*, T_u for all $u\in[v]$, where in each iteration of the loop in line 3 one

part T_u is treated. The steps in the **while** loop in lines 7-11 are analogous to those in Algorithm 9, i.e. a row r is selected that covers the maximal number of yet uncovered t-way interactions in T_u. Note however, that Algorithm 10 does not have to quantify over all rows $r \in [v]^k$, instead it only needs to consider elements of R_u. The reason for this is that we have $\varphi_{v,k,t}(r) \cap T_u \subseteq \varphi_{v,k,t}(r_u) \cap T_u$, letting r_u denote the row attained by substituting the first entry of r by u. In the previous relation, equality holds if the first position of r is already u. Otherwise, we have a proper subset. We also draw attention to the update of T_u (line 6), where the t-way interactions in T_u that are covered by the current rows of A are removed. In the next subsection we will treat a variant where this update is not performed.

Algorithm 10 slicedAETG

```
 1: INPUT: t, k, v
Require: t ≤ k
 2: A ← ∅                                              ▷ Initial array empty
 3: for u = 0 to v − 1 do
 4:     T ← T_u                                        ▷ Initialize T
 5:     R_u ← {u} × [v]^{k−1}                          ▷ Initialize T
 6:     T ← T \ φ_{v,k,t}(A)
 7:     while T ≠ ∅ do
 8:         determine r = arg max_{r∈R_u} |φ_{v,k,t}(r) ∩ T|
 9:         A ← (A; r)                                 ▷ Append row r to A
10:         T ← T \ φ_{v,k,t}(r)
11:     end while
12: end for
13: return A
```

The following theorem can be regarded as a derivation of Lemma 2 in [66] or of the theorem in Subsection 3 of [36].

Theorem 3.3.4 *The condition $T \neq \emptyset$ in line 7 ($T \subseteq T_u$) of Algorithm 10 is satisfied at most $v^t \log(v^{t-1}\binom{k}{t}) + 1$ times.*

Proof: To prove this, we first show that in each iteration in line 8 of Algorithm 10, we can find a row that covers at least $|T|/v^t$ of the t-way interactions in T. Let $V := \{(\tau, d) | \tau \in T, d \in R_u$ and d covers $\tau\}$ denote the set of all pairs which have as first position a currently uncovered t-way interaction τ which begins with u and as second position a row u that covers τ and begins with u. Each t-way interaction is covered by at least v^{k-t-1} rows of R_u, hence we have $|T| \cdot v^{k-t-1} \leq |V|$. An upper bound on the cardinality of V can be obtained when m is defined as the maximal number of t-way interactions in T covered by any row of R_u, which gives us $|V| \leq m \cdot |R_u|$. Combining these bounds for $|V|$, considering $|R_u| = v^{k-1}$, we get $m \geq |V|/|R_u| \geq |T| \cdot v^{k-t-1}/v^{k-1} = |T|/v^t$. Provided this, we have that T is covered after s steps, where $|T| \cdot (1 - 1/v^t)^s < 1$, when always

selecting a row that covers the maximal number of t-way interactions in T. Considering $T \subseteq T_u$, it is easily shown that the latter holds for $s \geq v^t \log(v^{t-1}\binom{k}{t})+1$, which concludes the proof. □

Since Algorithm 10 treats T_u for all $u = 0, \ldots, v-1$ sequentially, we get the following.

Corollary 3.3.5 *The number of rows N of the returned array in Algorithm 10 is upper bounded by $v^{t+1}\log(v^{t-1}\binom{k}{t}) + v$, and hence in $O(\log k)$ for fixed strength t and alphabet size v.*

Remark 3.3.6 *Note that for the former corollary we did not consider any reductions of T_u by the rows covering $T_0, \ldots, T_{u-1}$, while such t-way interactions are removed from T_u in line 6 in Algorithm 10. Especially for T_v, the number of previously covered interactions can be significant.*

Theorem 3.3.7 *The runtime of Algorithm 10 is in $O\left(v^{k+t}t^2\binom{k}{t}\log(v^{t-1}\binom{k}{t})\right)$.*

Proof: The number of steps required to find the row r in line 8 of Algorithm 10, is in $O(v^{k-1}t^2\binom{k}{t})$, as again, for each of the v^{k-1} elements of $r \in R_u$, we can check membership in T for the $\binom{k}{t}$ different t-way interactions covered by r in $O(t^2\binom{k}{t})$ time (Lemma 3.3.2). The time required by the statements in lines 9 and 10 is dominated by $O(v^{k-1}t^2\binom{k}{t})$. From Theorem 3.3.5 we get that the condition $T \neq \emptyset$, in the loop in line 7, is satisfied at most $v^t\log(v^{t-1}\binom{k}{t}) + 1$ times. As this holds for each $u \in [v]$, we get a runtime in $O\left(v^{k+t}t^2\binom{k}{t}\log(v^{t-1}\binom{k}{t})\right)$. □

Parasliced AETG: A Parallel Version of slicedAETG

We also consider a variation of Algorithm 10, hereafter referred to as PARASLICEDAETG, that differs insofar that the statement in line 6 is not executed. This algorithm can be considered a *parallelized* version of SLICEDAETG, as once the initial update of $T \leftarrow T \setminus \varphi_{v,k,t}(A)$ is omitted, the individual sets of t-way interactions T_u for all $u = 0, \ldots, v-1$ can be treated separately in a parallelized manner. However, we show that there is no need to execute all of the v branches. In the following, we denote permutations in cycle notation, in particular we denote by (uw) the transposition interchanging u and w, and denote permutations as exponents when they are applied. In particular, for an array $A = (a_{ij})_{i,j}$ we denote by $A^{(uw)}$ the application of the transposition (uw) to the entries of A, i.e. $A^{(uw)} := (a_{ij}^{(uw)})_{i,j}$; further for t-way interactions $\tau = \{(p_1, v_1), (p_2, v_2), \ldots, (p_t, v_t)\}$, $\tau^{(uw)}$ denotes the application of the transposition to the second element of each pair in τ, i.e. $\tau^{(uw)} = \{(p_1, v_1^{(uw)}), (p_2, v_2^{(uw)}), \ldots, (p_t, v_t^{(uw)})\}$.

Lemma 3.3.8 *Let $u, v \in [v]$ with $u \neq v$. Further, let A_u be an array that covers all t-way interactions in T_u, then the array $A_w := A_u^{(uw)}$, obtained by applying the transposition $(uw) \in S_{[v]}$ to all entries of A_u, covers all t-way interactions in T_w.*

Proof: Given an arbitrary t-way interaction $\tau = \{(p_1, w), (p_2, v_2), \ldots, (p_t, v_t)\} \in T_w$, we can apply the transposition (uw) to τ to obtain $\tau^{(uw)} \in T_u$. Since A_u covers all t-way interactions in T_u, $\tau^{(uw)}$ is covered by some row, say r, of A_u; but then the corresponding row $r^{(uw)} \in A_w$ covers τ. □

Provided this Lemma, when aiming for construction of a $\mathsf{CA}(N; t, k, v)$, instead of separately constructing arrays A_u that cover T_u for all $u \in [v]$, we can generate a single array A_0, that covers T_0 and construct a $\mathsf{CA}(N; t, k, v)$ by a product construction, i.e. vertically juxtaposing A_0 and $A_0^{(0u)}$ for all $u \in [v]$. Algorithm 11 makes use of this construction. With the already established results we get the following.

Algorithm 11 paraslicedAETG

1: INPUT: t, k, v
Require: $t \leq k$
2: $A \leftarrow \emptyset$ ▷ Initial array empty
3: $R_0 \leftarrow \{0\} \times [v]^{k-1}$ ▷ R_0 set of rows considered
4: $T \leftarrow T_0$ ▷ Initialize T
5: **while** $T \neq \emptyset$ **do**
6: determine $r = \arg\max_{r \in R_0} |\varphi_{v,k,t}(r) \cap T|$
7: $A_0 \leftarrow A_0 \cup \{r\}$
8: $T \leftarrow T \setminus \varphi_{v,k,t}(r)$
9: **end while**
10: $A \leftarrow A_0$
11: **for** $u = 1$ to $v - 1$ **do**
12: $A \leftarrow \left(\begin{array}{c} A \\ A_0^{(0u)} \end{array}\right)$
13: **end for**
14: **return** A

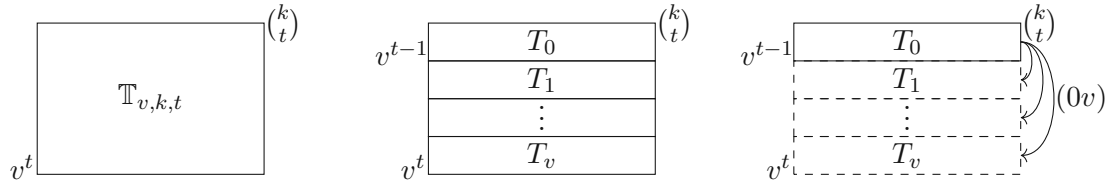
Corollary 3.3.9 *The resulting array in Algorithm 11 is a* $\mathsf{CA}(N; t, k, v)$ *and its size* N *is upper bounded by* $v^{t+1} \log(v^{t-1}\binom{k}{t}) + v$*, and hence for fixed strength* t *and alphabet size* v*, in* $O(\log k)$*.*

Proof: From Theorem 3.3.4 we get that T_0 can be covered with an array A_0 with at most $v^t \log(v^{t-1}\binom{k}{t}) + 1$. Now the assertion follows from Lemma 3.3.8, which shows that the array constructed via juxtaposition in lines 11-13 is a covering array. □

The runtime is reduced when compared to that of Algorithm 10, since we only have to cover a single part T_0 of the t-way interactions $\mathbb{T}_{v,k,t}$.

Theorem 3.3.10 *The runtime of Algorithm 11 is in* $O\left(v^{k+t-1}t^2\binom{k}{t}\log(v^{t-1}\binom{k}{t})\right)$.

Proof: Analogous arguments to those in the proof of Theorem 3.3.7 show that the runtime of Algorithm 11 until line 11 is in $O\left(v^{k+t-1}t^2\binom{k}{t}\log(v^{t-1}\binom{k}{t})\right)$, where the runtime is reduced by a factor of v, as we only have to cover the t-way interactions in T_0. The transpositions and juxtapositions in lines 11 - 13 can be done in $O(v^{t+1}k\log(v^{t-1}\binom{k}{t}))$ time, which is dominated by the other steps. □



(a) gAETG (b) slicedAETG (c) paraslicedAETG

Figure 3.13: Schematics of the incidence structures used in the implementations of the considered algorithms.

	U.B. for Rows	Runtime	Memory for T
gAETG	$v^t \log(v^t\binom{k}{t}) + 1$	$O\left(v^{k+t}t^2\binom{k}{t}\log(v^t\binom{k}{t})\right)$	$\Theta(v^t\binom{k}{t})$
slicedAETG	$v^{t+1}\log(v^{t-1}\binom{k}{t}) + v$	$O\left(v^{k+t}t^2\binom{k}{t}\log(v^{t-1}\binom{k}{t})\right)$	$\Theta(v^{t-1}\binom{k}{t})$
paraslicedAETG	$v^{t+1}\log(v^{t-1}\binom{k}{t}) + v$	$O\left(v^{k+t-1}t^2\binom{k}{t}\log(v^{t-1}\binom{k}{t})\right)$	$\Theta(v^{t-1}\binom{k}{t})$

Table 3.5: Bounds on number of rows of output CAs, runtime and memory usage.

3.3.3 Discussion on Related Data Structures

Before we present our experiments, we want to reason about the differences and similarities in the underlying data structures used to implement Algorithms 9 - 11. For the implementation of the Algorithms discussed in this paper we relied on an incidence structure for the considered subsets $T \subseteq \mathbb{T}_{v,k,t}$ of t-way interactions. This incidence structure can be visualized as an array, each entry representing a t-way interaction $\{(v_1, p_1), \ldots, (v_t, p_t)\}$, labelling rows with the t-tuple $(v_1, \ldots, v_t)$ and columns with the t-tuple describing the column selection $(p_1, \ldots, p_t)$. Figure 3.13 gives a visual representation of the incidence structures underlying the three discussed algorithmic constructions. In our implementations, however, these structures are realized as a vector, concatenating the columns of the respective incidence structure. Algorithm 9 processes the whole set $\mathbb{T}_{v,k,t}$ of t-way interactions at once, using $\Theta(v^t\binom{k}{t})$ memory. Algorithm 10 processes each T_i individually and sequentially for all $i = 0, \ldots, v-1$, which needs $\Theta(v^{t-1}\binom{k}{t})$ memory in each step. Algorithm 11 only treats T_0 individually and covers the other T_i via replication after applying the transpositions $(0i)$ for all $i = 1, \ldots, v-1$, which also requires $\Theta(v^{t-1}\binom{k}{t})$ memory.

Consider that for implementing Algorithms 9 - 11 the set of candidate rows does not need to be stored in memory and hence can be generated on the fly. In contrast, the set T of currently considered t-way interactions to be covered is stored entirely. As the memory required to store T dominates the memory required for the CA itself, this is the significant factor for the memory usage of the implementation. From the above discussion, we get the asymptotics for the memory required by Algorithms 9 - 11. We summarize these together with the upper bounds on the size of generated CAs and the asymptotics for the runtime in Table 3.5. Comparing these theoretical results, gAETG has the

smallest upper bound on the number of rows of the generated CAs, while requiring v times as much memory for the underlying incidence structure as the other two algorithms. slicedAETG and paraslicedAETG have a weaker upper bound on the number of rows for the generated CAs, but paraslicedAETG excels through its asymptotic runtime, which is reduced by a factor of v when compared to that of slicedAETG.

3.3.4 Evaluation

Having derived and discussed theoretical results of the introduced algorithms, we now turn to their experimental evaluation. To evaluate the performance of the algorithms, we focus on the size of generated CAs, runtime and memory usage, which are formulated as research questions (**RQ**). Our experiments were performed on a machine using an Intel i7 CPU clocked at 3.40 GHz and 24 GB of RAM. We computed several different CAs with our implementations of Algorithms 9 - 11, and report the results pertaining to size of generated CAs and runtime in Table 3.6. Columns headed by sliced and para refer to slicedAETG and paraslicedAETG respectively. The first three columns, under **Instance**, specify the parameters of the CA to be constructed. In the next three columns, under **Size of CAs**, the number of rows of the CA generated by the respective algorithm is given. In columns 7 and 8, under **Size Ratio**, we give the ratio of the number of rows of the CA generated by the respective algorithm to the number of rows of the CA constructed by gAETG. In the column headed *Bound Ratio* we report the ratio of the upper bounds on the number of rows of CAs when deriving them with gAETG or slicedAETG and paraslicedAETG, i.e. $(v^{t+1}\log(v^{t-1}\binom{k}{t}) + v)/(v^t \log(v^t\binom{k}{t}) + 1)$. In columns 10 - 12, we give the runtimes in seconds, and finally in the last two columns the speedups, i.e. the relative reduction in runtime compared to gAETG. Note that rounding errors, particularly for instances that could be computed within milliseconds, may influence the values in this column.

Evaluation of the Number of Rows of Generated CAs

RQ: *How does the actual number of rows of CAs compare when constructing them by means of Algorithm 9, Algorithm 10, and Algorithm 11?*

The computational results show that, except for the case of binary CAs of strength $t = 2$, the CAs constructed via Algorithm 10 are generally larger than those constructed with Algorithm 9. This is likely due to the limited influence of local versus global optimization on the size of CAs constructed for very small problem instances. In any case, the class of binary CAs of strength two ($t = 2$ and $v = 2$) is special, as it is the only class for which $\mathsf{CAN}(t, k, v)$ is known for all $k \in \mathbb{N}$, see e.g., [100], [99] or Theorem 2.2.11. For all other conducted experiments the CAs generated by Algorithm 10 are 1.1 to 2 times as large as those generated by Algorithm 9. On one hand, the increase of the number of rows is not surprising, as Algorithm 10 selects rows that are locally optimized for subsets $T_u \subset \mathbb{T}_{v,k,t}$ of t-way interactions (compare $\arg\max_{r\in\{u\}\times[v]^{k-1}}$ in Algorithm 10 line 8 against $\arg\max_{r\in[v]^k}$ in Algorithm 9 line 6). On the other hand, the factor describing the increase in number of rows stays well below the ratio of the upper

Table 3.6: Comparison of sizes of generated CAs and runtimes.

CA Instance			Size of CAs			Size Ratio			Time (in s)			Speedup	
t	v	k	GAETG	SLICED	PARA	SLICED	PARA	*Bound Ratio*	GAETG	SLICED	PARA	SLICED	PARA
$t=2$	$v=2$	10	9	8	10	0.89	1.11	1.75	0.003	0.001	0.001	2.82	2.82
		12	10	8	10	0.80	1.00	1.76	0.032	0.005	0.003	5.79	9.28
		14	9	8	10	0.89	1.11	1.77	0.143	0.028	0.017	5.14	8.18
		16	10	8	10	0.80	1.00	1.78	0.542	0.149	0.086	3.63	6.31
		20	11	10	12	0.91	1.09	1.80	14.615	4.350	2.740	3.36	5.33
		24	11	10	12	0.91	1.09	1.81	335.716	104.053	54.475	3.23	6.16
	$v=3$	8	15	19	21	1.27	1.40	2.42	0.023	0.008	0.003	2.93	8.72
		10	18	20	27	1.11	1.50	2.46	0.350	0.093	0.038	3.75	9.18
		12	19	21	27	1.11	1.42	2.49	5.798	1.209	0.544	4.79	10.65
		14	19	22	27	1.16	1.42	2.52	62.501	15.029	5.500	4.16	11.36
		16	21	23	27	1.10	1.29	2.54	795.641	184.348	63.441	4.32	12.54
	$v=4$	6	22	32	40	1.45	1.82	3.00	0.014	0.006	0.001	2.37	11.91
		8	26	34	40	1.31	1.54	3.10	0.378	0.100	0.025	3.80	14.87
		10	29	39	52	1.34	1.79	3.17	10.129	2.442	0.716	4.15	14.14
		12	31	41	52	1.32	1.68	3.21	293.190	77.282	15.754	3.79	18.61
	$v=5$	6	25	51	65	2.04	2.60	3.65	0.059	0.017	0.004	3.39	13.98
		8	39	53	65	1.36	1.67	3.78	3.347	0.719	0.158	4.66	21.19
		10	43	66	85	1.53	1.98	3.86	137.391	31.736	7.457	4.33	18.43
	$v=6$	6	46	75	96	1.63	2.09	4.30	0.270	0.062	0.013	4.35	21.47
		8	52	79	96	1.52	1.85	4.45	19.021	3.768	0.659	5.05	28.88
		10	60	97	126	1.62	2.10	4.55	1244.536	251.164	46.410	4.96	26.82
	$v=7$	6	63	105	133	1.67	2.11	4.94	0.863	0.183	0.032	4.72	27.36
		8	68	108	133	1.59	1.96	5.12	84.955	15.005	2.341	5.66	36.29
		10	77	125	175	1.62	2.27	5.24	7084.338	1259.878	222.319	5.62	31.87
	$v=8$	6	82	139	176	1.70	2.15	5.58	2.762	0.467	0.073	5.92	38.07
		8	89	142	176	1.60	1.98	5.78	322.537	49.917	6.849	6.46	47.09
		10	120	164	232	1.37	1.93	5.92	41450.148	5438.750	944.232	7.62	43.90
$t=3$	$v=2$	10	16	20	24	1.25	1.50	1.80	0.013	0.007	0.004	1.72	3.09
		12	21	23	28	1.10	1.33	1.82	0.222	0.051	0.031	4.40	7.07
		14	23	26	34	1.13	1.48	1.83	0.861	0.674	0.250	1.28	3.45
		16	22	27	34	1.23	1.55	1.84	5.745	3.767	1.481	1.53	3.88
		20	26	30	38	1.15	1.46	1.85	228.652	96.092	56.863	2.38	4.02
		24	29	33	42	1.14	1.45	1.86	6237.654	3098.439	1574.636	2.01	3.96
	$v=3$	6	43	54	69	1.26	1.60	2.48	0.006	0.002	0.002	2.86	2.86
		8	52	67	84	1.29	1.62	2.55	0.150	0.045	0.020	3.34	7.66
		10	63	79	93	1.25	1.48	2.59	3.363	1.072	0.345	3.14	9.75
		12	66	83	111	1.26	1.68	2.62	50.374	17.774	6.402	2.83	7.87
		14	71	95	129	1.34	1.82	2.64	968.705	313.088	107.762	3.09	8.99
	$v=4$	6	102	131	168	1.28	1.65	3.23	0.076	0.019	0.006	4.08	12.04
		8	125	163	216	1.30	1.73	3.32	3.065	0.785	0.269	3.90	11.38
		10	144	184	252	1.28	1.75	3.38	136.433	31.695	10.728	4.30	12.72
	$v=5$	6	186	263	355	1.41	1.91	3.97	0.915	0.110	0.031	8.30	29.63
		8	237	324	435	1.37	1.84	4.09	45.499	7.290	2.022	6.24	22.50
		10	274	377	510	1.38	1.86	4.16	2414.913	478.319	126.868	5.05	19.03
	$v=6$	6	325	455	612	1.40	1.88	4.72	4.745	0.464	0.108	10.22	43.82
		8	398	564	798	1.42	2.01	4.86	307.547	44.804	10.833	6.86	28.39
		10	459	638	906	1.39	1.97	4.94	24465.040	4116.988	1008.546	5.94	24.26
	$v=7$	6	507	728	1008	1.44	1.99	5.46	11.912	1.588	0.326	7.50	36.55
		8	627	880	1274	1.40	2.03	5.62	1343.968	204.568	43.316	6.57	31.03
	$v=8$	6	756	1098	1496	1.45	1.98	6.20	38.707	4.948	0.927	7.82	41.75
		8	916	1320	1928	1.44	2.10	6.38	6948.383	907.519	164.838	7.66	42.15
$t=4$	$v=2$	8	32	40	46	1.25	1.44	1.80	0.005	0.002	0.001	1.95	3.42
	$v=3$	6	131	164	195	1.25	1.49	2.54	0.015	0.005	0.003	2.85	4.82
		8	175	224	285	1.28	1.63	2.62	0.594	0.204	0.087	2.91	6.82
	$v=4$	6	407	515	616	1.27	1.51	3.33	0.265	0.067	0.020	3.97	13.16
		8	547	699	856	1.28	1.56	3.43	21.325	4.620	1.412	4.62	15.10
	$v=5$	6	962	1285	1720	1.34	1.79	4.12	3.643	0.488	0.133	7.47	27.44
		8	1311	1764	2450	1.35	1.87	4.25	327.216	54.645	15.141	5.99	21.61
	$v=6$	6	1956	2710	3558	1.39	1.82	4.91	16.751	2.524	0.837	6.64	20.01
		8	2670	3712	5280	1.39	1.98	5.06	2953.887	407.093	96.238	7.26	30.69
	$v=7$	6	3576	5110	6762	1.43	1.89	5.70	58.771	10.225	1.960	5.75	29.99
		8	4875	6855	9989	1.41	2.05	5.87	18214.156	2198.659	455.896	8.28	39.95

bounds (*Bound Ratio*) on the number of rows of CAs, derived for these two algorithms (compare columns 8 and 10 of Table 3.6). This observation is still true, when considering the arrays generated by Algorithm 11, which produced CAs being 1 to 2.60 times as large as those generated by Algorithm 9 (columns 9 and 10 of Table 3.6). Finally comparing Algorithm 10 and Algorithm 11, we see that Algorithm 10 generates CAs that are smaller than Algorithm 11, showing that the initial update of T for all $u \in [v]$ (see line 6 in Algorithm 10) significantly influences the size of the resulting CA. See also Remark 3.3.6.

Evaluation of Runtimes

RQ: *How do the runtimes of these algorithms compare?*

The runtimes of Algorithms 10 and 11 are significantly reduced when compared to Algorithm 9. Particularly Algorithm 11 exhibits a speedup greater than v (even $v \log v$) in all documented cases. In addition to the algorithmic improvement in runtime, our implementation benefits greatly from the reduced overhead to set up the incidence structure (see Subsection 3.3.3). We did not measure the time needed for the replications required by Algorithm 11 (lines 11-13), which are computationally trivial, as they merely require copying the existing array and transposing values.

Evaluation of Required Memory

RQ: *Can the reduced memory usage of slicedAETG extend the usability of gAETG?*

t	v	gAETG	slicedAETG
2	10	5971	16221
3	10	228	481
4	10	48	85
5	10	20	32
6	10	13	18

Table 3.7: Column limits with 16 GB of memory.

To evaluate extended usability for practical applications we performed experiments to determine the point at which the memory usage of each implementation reaches 16 GB of RAM. In these experiments we chose a fixed alphabet size $v = 10$, a size occurring in industrial applications [146]. In Table 3.7, we give the maximal number of columns k for strengths t from two to six for which the incidence structures as well as other data required by our implementations fit into 16 GB. The column headed with gAETG relates to our implementation of Algorithm 9 and the column headed by slicedAETG relates to both Algorithm 10. Our results show that a significantly larger number of columns can be treated by utilizing our memory efficient variants of gAETG. While we initially verified the reduction in size of the incidence structure used in our implementations of Algorithms 10 by a factor of the alphabet size v, the theoretically obtainable reduction of $1/v$ (see Table 3.5) can not be realized due to overhead introduced by our implementation. Note that the choice of data type used in the implementations influences the memory usage, albeit only up to a constant factor.

Although the CAs constructed via our algorithms are far from optimal, practical applications, especially industrial applications do require CAs with a large number of columns, see e.g. [269] and [270], and the Sections 4.3 and 5.2 of this thesis. The experiments documented in Table 3.7 show case that the presented method can extend the usability of a CA generation algorithm.

3.4 The IPO-Q Algorithm

This section presents a hybrid algorithm for CA generation, that combines a greedy heuristic dedicated for CA construction with a metaheuristic approach, based on the publication [143]. To be more explicit, we introduce the IPO-Q algorithm that combines the well-known IPO strategy [174] for CA generation with the quantum-inspired metaheuristic algorithm for CA generation introduced in [245]. Different versions of this algorithm are implemented and evaluated in an experimental evaluation, against each other and against an algorithm implementing the IPO strategy.

We start by providing the necessary preliminaries and thereafter we describe the IPO-Q algorithm and the different versions of it considered in the experimental evaluation which concludes this section.

3.4.1 Preliminaries

In this subsection we briefly summarize the main concepts of the IPO strategy and the quantum inspired evolutionary algorithm for CA generation that will serve as starting point for the IPO-Q algorithm.

Review of the IPO Strategy

The In-Parameter-Order (IPO) strategy was first introduced in [174] for the generation of strength $t = 2$ covering arrays, and was later generalized to work also for higher strengths [27, 88] and arbitrary alphabets. However, in this section we are concerned the binary alphabet only. Algorithms implementing the IPO strategy [27, 88] are greedy algorithms that grow CAs in two dimensions, horizontally and vertically. The input to these algorithms is the desired strength t as well as the number of columns k of the desired CA.

Algorithm 12 IPOG(t, k, v)

```
1: A ← {0,1}^t
2: for i ← t + 1, ..., k do
3:     A ← HorizontalExtension(A, i)
4:     if any tuples are uncovered then
5:         A ← VerticalExtension(A, i)
6:     end if
7: end for
8: return A
```

A pseudo-code of the IPOG algorithm is given in Algorithm 12, and it can be summarized as follows. Starting with the $\{0, 1, \ldots, v-1\}^t$ array of all vectors over $[v]$ of length t, algorithms implementing the IPO strategy proceed iteratively in two phases that alternate each other, the horizontal extension (adding a column) and the vertical extension (adding rows), until the desired CA with k columns has been constructed. Thereby it is ensured that the result of the vertical extension phase is always a CA. The HorizontalExtension(A, i) and VerticalExtension(A, i) algorithms are only described informally as follows, for more details the interested reader is kindly referred to [27]. 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 $(v_1, \ldots, v_i, v_{i+1})$-ary t-tuples is maximal under all possible values that can be selected for this position.

In case all t-way interactions can be covered in this way, there is no vertical extension phase, potentially unspecified entries in the newly added column are marked as *don't-care values* and the algorithm proceeds by adding the next column to the array.

Otherwise the remaining uncovered t-way interactions have to be covered, where for each such t-way interaction τ the algorithm tries to find a row of the existing array 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 t-way interactions are covered. Hence the resulting array is a $\mathsf{CA}(N; t, i+1)$ with some unspecified values, representing *don't care* values. The algorithm enters again the horizontal extension phase, unless k columns are already reached. Finally all unspecified values in the array can be set arbitrary.

Review of the QiEA Algorithm for CA Generation

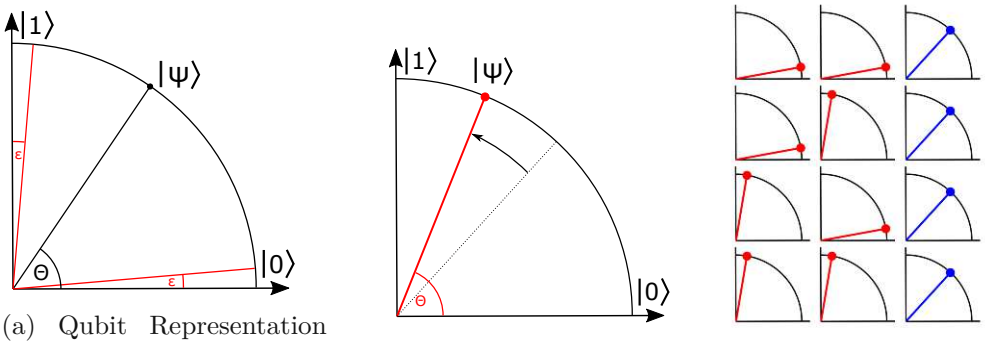
Recently, a quantum-inspired evolutionary algorithm for CA construction (QiEA) was proposed in [245], which we review briefly and gather the notions necessary for the algorithm introduced in Subsection3.4.2. QiEA takes as input the parameters N, t and k of a desired CA. The underlying idea is to consider an $N \times k$ array of qubits, which states $|0\rangle$ and $|1\rangle$ are identified with the numerical values 0 and 1. Thus, when observing the individual qubits, they collapse to either state and an $N \times k$ array over $\{0, 1\}$ is attained. To obtain an actual covering array, the states of the qubits are iteratively evolved.

For *classical* computational realization, in [245], a reduced qubit representation is used,

$$|\Psi\rangle = \cos\Theta\,|0\rangle + \sin\Theta\,|1\rangle\,, \text{ with } \Theta \in \left[0, \frac{\pi}{2}\right],$$

where the state of a qubit is completely specified by its angle Θ, see Figure 3.14a. An angle of 0° corresponds to the state $|0\rangle$ and 90° to $|1\rangle$ respectively. Measurement of a qubit yields state $|0\rangle$ with probability $(\cos\Theta)^2$ and state $|1\rangle$ otherwise. *In the following we use the term qubit synonymous for this reduced qubit representation.*

In [245], an initial array of qubits in the state corresponding to an angle of 45° is generated, representing a uniform distribution of the possible states, hence representing



(a) Qubit Representation and ϵ mutation. (b) Qubit Update. (c) Qubit Bias.

Figure 3.14: Reduced qubit representation (a) and used update mechanics (b) and (c).

a *neutral state*. The modification of qubits is realized in rounds, until either a CA is found, or a set number of iterations passed. In each round, a new candidate solution is generated by measuring every qubit. The resulting array is evaluated in terms of the number of covered t-way interactions. The best array over all rounds is stored as base for modification. In contrast to actual quantum computation, we can preserve the state $|\Psi\rangle = \cos\Theta\,|0\rangle + \sin\Theta\,|1\rangle$ of a qubit after measurement. To update the state of a qubit, it gets rotated by a small angle towards state $|0\rangle$ or $|1\rangle$, see also Figure 3.14b. The direction of the rotation is based on the corresponding entry in the current best array. The qubit rotations serve to guide the search towards a promising subset of the search space, while the probabilistic nature of the qubit measurement serves as an exploration mechanism.

Further, the concept of *mutation* is used as a constraint on the rotation of qubits, by preventing them from completely converging to one of the states $|0\rangle$ or $|1\rangle$, see again Figure 3.14a. It thus offers means for individual qubits to escape local minima. For more details the interested reader is referred to [245].

3.4.2 IPO-Q: A Quantum-inspired IPO Algorithm

In this subsection we introduce the quantum-inspired extension strategy IPO-Q for CA generation. It combines the IPO strategy with ideas of quantum-inspired evolutionary algorithms. Like other algorithms implementing the IPO strategy, IPO-Q consists of horizontal and vertical extension steps. The algorithm acts on an array of qubits that is iteratively extended. Based on this qubit array, in each extension step QIEA is used to generate an array optimizing the number of covered t-way interactions. IPO-Q is given by means of a pseudocode in Algorithm 13 and can be described as follows.

The algorithm starts with a $2^t \times t$ array Q_0 of qubits in the neutral state, from which an initial binary array is generated, using QIEA. Afterwards, when in the i-th step a CA was found by QIEA, based on the array of qubits Q_i, IPO-Q enters the horizontal extension step and adds a new column q_{i+1} of qubits, yielding $Q_{i+1} = (Q_i, q_{i+1})$. The newly added qubits in q_{i+1} are initialized in neutral state, while the qubits of Q_i are

biased towards the corresponding values in the previously computed CA, see Figure 3.14c. Hence, the *new* qubits are left open for exploration, while the *old* qubits are more likely to measure the values of the previous CA with i columns. Then QIEA is used to maximize the number of covered t-way interactions based on the array of qubits Q_{i+1}.

If QIEA does not find a CA based upon Q_{i+1}, IPO-Q performs a vertical extension step. A new row of qubits is added to Q_{i+1}. The new qubits are initialized in the neutral state, while all other qubits get biased towards the corresponding value in the previous best solution. Again, QIEA is used attempting to find a CA based on Q_{i+1} with the increased number of rows. If it fails, additional rows are added to Q_{i+1} one by one, as the process is repeated, until a CA is found.

Once a CA with $(i+1)$ columns is generated, IPO-Q enters the next horizontal extension phase. These steps get repeated until a CA with the desired number of columns is found.

Algorithm 13 IPO-Q

```
1: INPUT: k, t, QIEA settings
2: Generate initial 2^t × t array A using QIEA
3: for i ← t, ..., k do
4:     Add new column of qubits in neutral state to A          ▷ Horizontal Extension
5:     Bias old qubits towards previous solution
6:     Apply QIEA to maximize the number of covered t-way interactions
7:     while any t-way interactions are uncovered do
8:         Add new row of qubits in neutral state to A         ▷ Vertical Extension
9:         Bias old qubits towards previous solution
10:        Apply QIEA to maximize the number of covered t-way interactions in A
11:    end while
12: end for
13: return A
```

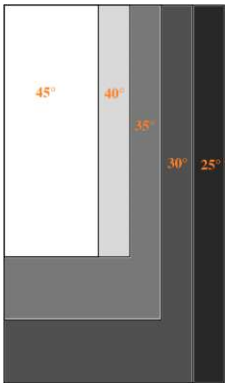
New Concepts for IPO-Q. In the following we describe two new realizations of the concepts of mutation and bias, that adapt to the extension strategy used in IPO-Q.

First, during our experiments (see Subsection3.4.3), we noticed that constant mutation rates negatively affect the solution quality. Further, once the number of columns k and the number of rows N of the desired arrays, and thus the number of qubits, gets too large, mutation rates that work well for less columns make the system too unstable. Therefore, in addition to the mutation types presented in [245], we introduce `variable mutation`, which sets the angle for mutation to

$$\epsilon = \begin{cases} g, & N \times k \leq 100 \\ \frac{100 \times g}{N \times k}, & N \times k > 100 \end{cases} \tag{3.6}$$

where g is an initial parameter called `base mutation`.

Second, we introduce a concept further referred to as `onion` extension. In this version, the bias of all qubits is increased by a set amount at the beginning of every horizontal



(a) Bias Layers.

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

(b) Resulting $\mathsf{CA}(12; 3, 7, 2)$.

Figure 3.15: Visualization of the onion concept, with the different bias layers on the left and an example array on the right.

extension step, until the bias reaches 45°, i.e. the qubits are fixed to state $|0\rangle$ or $|1\rangle$. This results in an onion-like structure of the bias values of qubits in Q_i, where newer qubits are left more open for exploration, while older qubits allow less exploration.

Figure 3.15 provides an example of the different bias layers caused by the `onion` concept for $\mathsf{CA}(12; 3, 7)$. After the initial $\mathsf{CA}(8; 3, 3)$ is generated, IPO-Q performs a horizontal extension. At the beginning of the extension step, the bias of the old qubits is increased by a given angle Δb. Afterwards, QIEA is used to find a CA with 4 columns, resulting in a successful horizontal extension. Hence, the next horizontal extension step starts and the bias of all qubits is increased. Note that the bias of the qubits in Q_0 is increased for the second time, therefore its qubits are biased more towards the previous solution than the qubits in Q_1. If a horizontal extension fails to construct a CA, the rows added by means of vertical extension have the same bias as the added column, see e.g. Q_2 and Q_3.

3.4.3 Evaluation of Different Configurations of the IPO-Q Algorithm

To evaluate the effect of different settings of IPO-Q,we compared the following selected configurations:

1. $g0$ uses no mutation (i.e. $\epsilon = 0°$),
2. $g5$ uses a mutation angle of $\epsilon = 5°$, that is applied to every qubit,
3. $v5$ uses `variable mutation` with a `base mutation` of $g = 5°$ (equation (3.6)),
4. *onion* $v10$ implements the `onion` concept with $\Delta b = 2°$, using `variable mutation` and a `base mutation` of $g = 10°$.

We compared these configurations by means of computing a $\mathsf{CA}(N; 3, k)$ for $k = 2, \ldots, 100$ and comparing the resulting values for N. For each configuration, we conducted 5 runs for

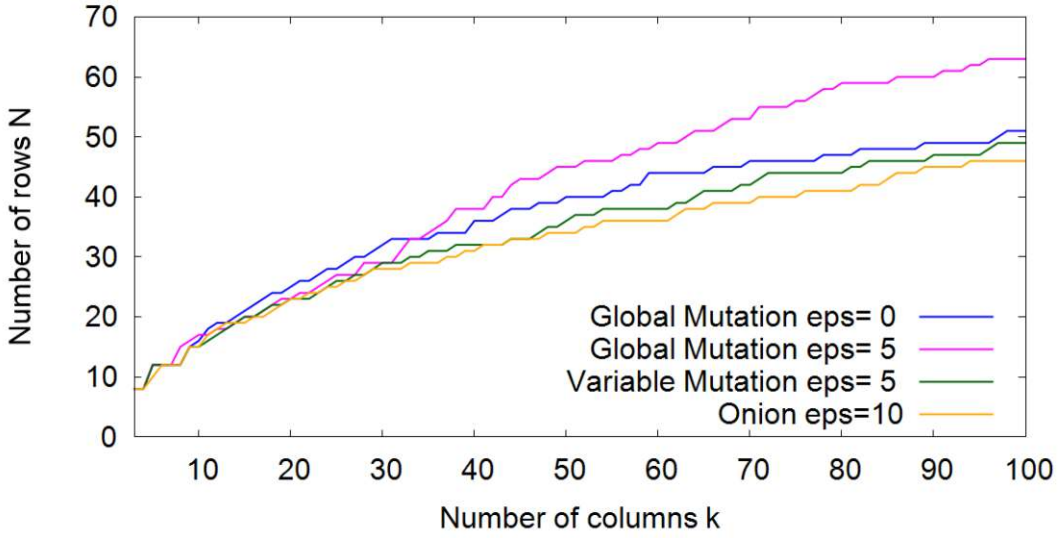


Figure 3.16: Parameter evaluation for IPO-Q on the instance $\mathsf{CA}(N; 3, k)$ (smaller values are better).

every $\mathsf{CA}(N; 3, k)$ instance and recorded the minimal number of rows N of the 5 generated CAs. The results are depicted in Figure 3.16, where the horizontal axis represents the number of columns k and the vertical axis represents the number of rows N. As one aims to minimize the number of rows in a CA, smaller values indicate better results.

We can see that configuration $g0$ performs worse than the other configurations for up to $k = 30$ columns, i.e. it generates CAs with a higher number of rows, but manages to find acceptable solutions for a higher number of columns. Interpreting these results, we believe this is due to $g0$ using no mutation and hence lacking means of exploration to optimize smaller instances.

Configuration $g5$, on the other hand, finds CAs with a smaller number of rows compared to $g0$, for $k \leq 30$, but becomes unstable for higher values of k.

Configuration $v5$ improves on both, $g0$ and $g5$, for all values of k. This shows nicely that the concept of `variable mutation`, decreasing the mutation angles proportional to the number of qubits $N \times k$, combines the advantages of exploration of small instances ($g5$) and exploitation of previous solutions for larger instances ($g0$), respectively.

Finally, *onion* $v10$ further improves the results of $v5$, yielding the best results of the considered IPO-Q configurations. In the general IPO-Q algorithm approach without the `onion` concept, due to iterative horizontal and vertical extension steps, inner parts of the array are optimized multiple times. While, the concept of `variable mutation` can only decrease exploration on a global scale, *onion* $v10$ reflects the core idea of the original IPO strategy, as depicted in [174]. The inner parts of the qubit array, that already experienced multiple optimization rounds, get fixed by means of increasing the bias up to 45°, while the high initial mutation angle of 10° allows for exploration for newly added qubits.

3.4.4 Comparison with IPOG-F and best known upper Bounds for CAN

In a second phase of experimental evaluation, we compared the results of the IPO-Q algorithm against the best known upper bounds for covering array numbers [97] and the numbers provided by NIST [271] using the IPOG-F algorithm. For that purpose we considered the configuration of IPO-Q, that had the best performance in the experiments reported in the previous subsection, i.e. in the following evaluation IPO-Q refers to the *onion* $v10$ configuration described above.

The graphs in Figure 3.17 compare the number of rows N of generated CAs, on the vertical axes, for a given number of columns k, on the horizontal axes. Furthermore, selected numerical values are highlighted in Table 3.8. For strength two, IPO-Q consistently finds CAs with less rows and stays close above the covering array number CAN. For higher strengths t, IPO-Q finds smaller CAs for up to $k = 81$ columns for strength $t = 3$ and for up to $k = 21$ columns for strength $t = 4$, see Figure 3.17. For higher number of columns, IPOG-F produces better results.

We believe these results reflect the probabilistic nature of the Quantum-inspired algorithm very well. For small strengths t and number of columns k, IPO-Q can fully utilize the probabilistic search and can improve on the representative of the classical IPO algorithms. However, for higher strengths and higher numbers of columns IPOG-F produces better results.

CA instance	*IPO-Q*	*IPOG-F*	CAN$\leq$
CA$(N; 2, 100, 2)$	10	13	10
CA$(N; 2, 500, 2)$	14	17	10
CA$(N; 2, 1500, 2)$	16	20	14
CA$(N; 3, 20, 2)$	23	25	18
CA$(N; 3, 50, 2)$	34	36	28
CA$(N; 3, 100, 2)$	46	45	33
CA$(N; 4, 10, 2)$	34	41	24
CA$(N; 4, 20, 2)$	66	65	39
CA$(N; 4, 35, 2)$	91	85	64

Table 3.8: Results for selected CAs CA$(N; t, k)$. Values for "CAN$\leq$" can be found at [97].

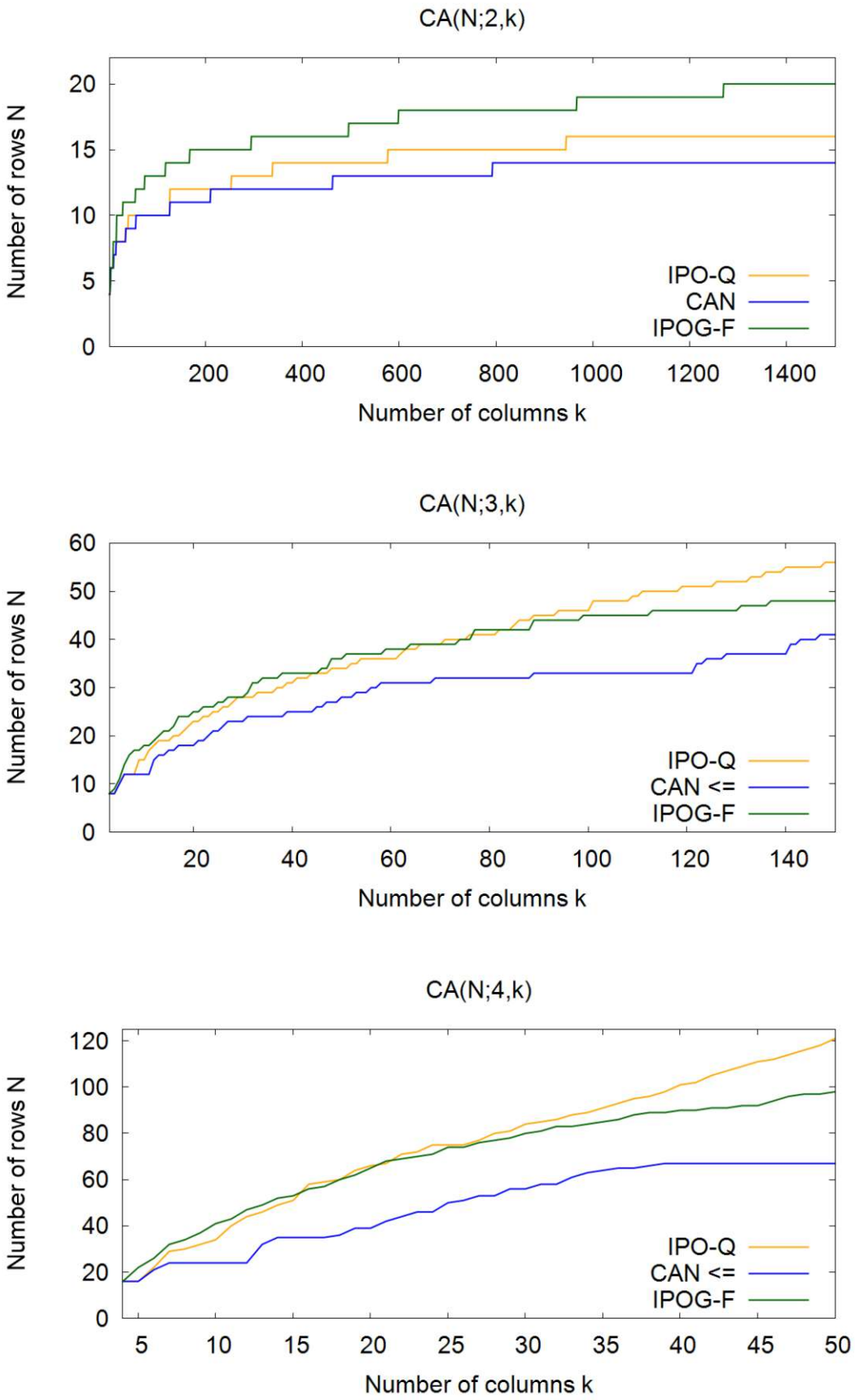


Figure 3.17: Comparison of the number of rows of generated CAs by IPO-Q and IPOG-F with the best known upper bounds for CAN, maintained at [97], denoted as "CAN <=", for strengths $t = 2, 3, 4$ from top to bottom (smaller values are better).

3.5 Summary

To summarize the third chapter of the this thesis, we presented artificial neural network models for the construction of CAs and generalizations thereof. We were able to devise Boltzmann machines and Hopfield networks for the construction of CAs, that *do not depend* on previous specification of a desired number of rows. This was achieved by combining the mapping of CA generation problems to set cover problems, as summarized in Section 2.3 and presented in [144], with the solution strategies for set cover problems based on artificial neural networks described in [247] and [248]. We enhanced the Boltzmann machines with learning capabilities in the form of weight updates and graph updates. Finally, we conducted an in-depth experimental evaluation of the presented approaches. While the CAs generated by the neural networks are not always optimal, our comparison shows that the introduced learning capabilities for Boltzmann machines allow for significant improvements in solution quality. Our results confirm that the application of neural networks to the CA generation problem is possible and can even lead to optimal solutions for small instances. This may constitute the grounds for hopes for future applications in CA generation, such as verification or post-optimization via artificial neural networks.

In Section 3.4 we proposed an quantum-inspired IPO algorithm IPO-Q, merging the ideas of the IPO startegy with a quantum inspired evolutionary algorithm for CA computation. We introduced two new concepts, called `variable mutation` and `onion` extension, that improved the performance of IPO-Q in our evaluation. The experiments further showed, that in some cases IPO-Q can construct CAs with less rows than the well known IPOG-F algorithm. A generalization of this algorithm to higher and mixed alphates remains subject to future work.

Finally, in Section 3.3 we have first presented algorithmic variations of GAETG, a known greedy heuristic strategy for CA construction, that extend the practical applicability by reducing memory consumption and runtime while tolerating increases in the size of generated CAs. The experimental results show that the increase in number of rows stays well below the derived upper bounds. In terms of theoretical interests, this raises the question for better estimates for the upper bound on the size of the returned CAs. In terms of applications, the proposed algorithmic variations may be implemented in other algorithms for CA generation such as DDA [71], since the modifications of the underlying data structure can be transferred to any algorithm that relies on the same, or similar, data structure. In that regard, our work provides the basis for an extension of existing algorithms for CA generation to meet the requirements arising from large problem instances in applied or indurstrial settings, which will be a focus in the second part of this thesis.

Part II

Contributions to Combinatorial Software Testing

CHAPTER 4

Applications of Combinatorial Testing to Test Software Systems

This chapter is focused on applications of combinatorial testing (CT) for *fault detection* in software. This means that in the following applications we are merely interested in *revealing the presence* of a failure caused by a t-way interaction of the input parameters. Knowing about the presence of a failure-inducing t-way interaction (t-FIT) can be, in some cases, the starting point of a combinatorial fault localization process (which will be the subject of the next chapter). In other cases, it may already be the required information for an engineer to take action and resolve the cause of the failure. This is the case in Section 4.3, where the results of combinatorial testing were post-processed by a domain expert who identified the cause for the failure and took the necessary steps. However, we want to emphasize again that the primary focus of CT is not to resolve failures or locate flawed statements in source-code, but to *trigger failures*.

The overarching goal in this chapter is to extend the application domains of CT. In particular, this means

- to either extend the applicability of CT in an already explored domain by going beyond known boarders, or
- to pioneer applications of CT to SUTs of a novel, previously unexplored domain.

In the remainder of this chapter, we aim to achieve this goal by applying CT to test a smart building protocol and large language models. To the best of my knowledge, both are application domains that have previously not been subject to CT. Further, we present an application of CT for testing a web tool, a domain previously explored; however, the subject SUT is modeled with more than 2,000 input parameters, representing – at the time of writing – the largest real-world SUT where CT has been applied, and the results have been documented in a research paper. Before we detail these works, we provide some

context and present related work on similar applications of CT. Additional background information for each application domain will be given in the respective section.

In previous works, the CT process (Figure 1.3) has been adapted to other domains than software testing, for example hardware Trojan testing [68], explainable AI [272], or the testing of automated driving functions [149, 273].

4.1 Related Work

The authors of [56] conducted an empirical study, evaluating the performance of CT when applied for real-world, industrial applications. For five industrial SUTs, they compared the number of failures they detected using CT with those detected by the in-house testing teams using other methods. The results of the v case studies show that CT is in fact an effective technique for detecting failures in practice. In relation to the total number of failures, CT detected 89.3% failures triggered by a single parameter, i.e., 1-way interactions and 93.3% of the failures triggered by higher strength t-way interactions. In comparison, the in-house testing teams could detected 71.4% failures triggered by a single parameter and only 6.7% of the failures triggered by higher strength t-way interactions.

In [57] an empirical comparison of combinatorial testing, random testing and adaptive random testing is presented. The comparison is conducted by means of nine real-world programs, for which a total of 1683 different scenarios are considered, i.e. different settings of the programs, where the proportion of available parameters, the proportion of available constraints in the IPM and the failure rate is varied. The authors conclude that their results show significant differences in the failure detection ability of the methods, especially when the failure rates are relatively low. Overall, combinatorial testing performs best, achieving results, being second to none in 98% of the documented scenarios. Adaptive random testing performs better than random testing and is comparable to CT in 96% of the scenarios, but its computational cost can be up to 3.5 times higher than those of CT when the program is highly constrained. Only for the case of a highly constrained SUT, when there is no information about the constraints, a large random test suite is as effective as CT, while having significantly lower computational cost for test generation, according to [57].

A contemporary work to the one we will present in Section 4.3, is the one presented in [274], which reports on the combinatorial coverage measurements to evaluate the validation framework as part of the reporting engine of the Adobe analytics product. Their evaluation shows that combinatorial coverage measurements are an effective way to supplement existing validation strategies.

Similarly, a contemporary work to the one we will present in Section 4.2, is given by [275], where the combinatorial coverage measurements of various test suites designed for a building automation product of Siemens Building Technologies. Further, the use variable-strength covering arrays for reducing the number of the required tests is considered.

CT for Web Application Testing. The authors of [276] use CT in a two step approach to web service testing. In a first step *combinatorial group testing* is used to rate different compositions of a web service consisting of multiple component web services, where the same input is used for all thus composed web services. In a second step, after the number of options for the composite web service was reduced, the remaining options are tested against undesired interactions of the component web services. For this, combinatorial testing based on biased covering arrays is suggested, in order to handle the possible large number of combinations of web services.

In [277] a CT based approach for the generation of cross-site scripting (XSS) attack vectors is examined. The authors define an attack grammar, which serves as an IPM and derive attack vectors based on CAs. The attack vectors are executed against a web application's fire wall, which is considered the SUT. Their results show that their attack vectors perform equal or better than conventional ones in almost all documented cases.

The work in [278] presents a combinatorial testing approach for detecting SQL injection vulnerabilities in web applications. The authors describe a tool designed for automated SQL injection vulnerability testing of web applications, and apply it to real-world applications, where they could successful detect SQL injection vulnerabilities. Their evaluation shows the suitability of their approach compared against a vulnerability scanner and real world applications with known flaws.

CT for Protocol Testing. Simos *et al.* [279] use CT for testing the Transport Layer Security (TLS) protocol, where they focus on the TLS handshake as one of the most important components of TLS. They devise IPMs for three different messages that can appear on the client-side of the TLS protocol, namely `ClientHello`, `ClientKeyExchange` and `ClientFinished`. Based on these IPMs they generate pairwise combinatorial test sets, that are used to specify the TLS messages. The derived tests are executed using a test execution framework. In a follow up work [280], a weighted, sequential combinatorial testing approach is followed. For the considered TLS implementation, weights for the handshake messages are created by analyzing bug reports. Using a multi-set of ten handshake messages, including duplications, an SCA of strength three with ten sybmols is constructed, where the weights of the events are taken into account. The sequence of messages given by a row of the SCA are then used to obtain an IPM for which a pairwise combinatorial test set, i.e. a CA of strength two, is generated and used for testing.

CT for Testing Machine Learning and Artificial Intelligence Applications. The authors of [281] apply CT methods for active learning, i.e. the labeling of unlabeled data points to optimize the learning phase of a Machine Learning (ML) model. Such additionally labeled data points, typically are beneficial for a particular ML model, but maybe less so for a different model. Additionally, such data points may come from a small sub-space of the feature space creating a sampling bias. The proposed method is based on finding those unlabeled data points which contain t-way interactions that are not included in the already labeled data points. The authors find that their proposed method for active learning performs similarly when used for the original model and that it outperforms them, when the data is transferred to new models.

Ma et al. propose a t-way coverage based test generation technique targeting specifically the neuron layers of deep learning systems [267]. Their idea is to benefit from combinatorial coverage for exploring the large state-space of deep neural networks, testing for neuron interactions. They apply the notion of combinatorial coverage to the neuron of a layer of a deep neural network, and consider the binary t-way interactions of activated or non-activated neurons. Based on this coverage criterion they can search for input that maximizes the coverage of t-way interactions between the neurons of a layer. An experimental evaluation demonstrates the potential usefulness of CT for deep neural network testing.

A different approach is followed, by the authors of [282] who apply CT methods to the input of ML models, rather than to their internals. They use an extended notion of combinatorial coverage for the construction of ML test sets, training sets, and for directing labeling efforts. Demonstrating their method by means of the MNIST digits data set [283], their results show that model performance varies greatly between test sets with covered and non-covered value combinations. Further, they show that training sets constructed to satisfy certain coverage properties lead to more robust models than identically distributed training sets, and finally, that they can improve label and sample efficiency.

Most recently, a combinatorial approach for identifying individual fairness violations in pre-trained ML models was presented [284]. The authors investigate whether "*the key insight that has allowed combinatorial testing to be effective for general software testing could also apply to fairness testing*". Their approach consists of two phases. First, based on existing training data an IPM for combinatorial testing is generated, by identifying parameters and their respective values in the training data. Second, individual fairness violations are identified using a counterfactual approach. Therefore, based on a previously generated t-way test set, perturbations are generated that are similar to the t-way test instance, but that differ by modifications of some protected attributes (e.g. age, sex, or ethnicity). The results of an experimental evaluation of the proposed approach using three datasets, suggest that the combinatorial approach can successfully identify fairness violations in ML models. In some cases, more than 40% of the generated t-way test cases resulted in a fairness violation. Further, the large number of fairness violations is across different types of ML classifiers, which leads the authors to hypothesize that their approach may be model-agnostic and can be adopted for different ML models.

4.2 Applying Combinatorial Testing to BACnet Protocol Conformance Testing

In this section, we report an application of combinatorial testing for testing communication protocols used in buildings. As a proof-of-concept we apply CT through a widely used Siemens product, the APOGEE Insight® workstation, which can be used to centrally manage multiple devices in a smart building using the BACnet protocol for communication. To the best of my knowledge, this is the first application of CT in the realm of smart building automation protocols.

The motivation for this work is to devise a testing methodology for thoroughly testing smart building protocols, which could increase the quality of smart buildings in smart cities.

This section is based on the publication [145], which was the result of a collaboration with a partner from the industry. My contribution to the work presented in [145] was to aid the input space modeling performed by the partner from the industry and to construct the used combinatorial test set.

Smart Buildings as Part of Life in Modern Society.

A report conducted by the Centre of Regional Science at the Vienna University of Technology identified six main axes (or dimensions) along which a ranking of 70 European middle size cities can be made [285]. These axes are: a smart economy, smart mobility, a smart environment, smart people, *smart living*, and, finally, smart governance. Switching the focus to human beings, a study documented in [286] shows that Americans spend 87% of their time indoors, that makes almost 21 hours per day on average. This observation is underpinned by considering the same phenomenon from the contrary point of view. A study [287] conducted in 2011 showed that for several western countries the pooled estimates for the times spent out doors during work weeks on weekdays and weekends were 1:43h and 2:38h respectively, and about 5-6h during holidays. Considering that people spend the majority of their time inside, strongly suggests to consider smart buildings a main factor of smart cities. Finally, another paper [288] focusing on cyber-security threats for smart cities explicitly names four key components for smart cities: smart grids, *building automation systems*, unmanned aerial vehicles and smart vehicle.

4.2.1 Introduction and Background to BACnet, Building Automation and the Siemens APOGEE Insight® Product

In modern smart buildings there exist various hardware pieces and processes, such as thermostats, lights, air ventilators, air condition, fire alarms, water controls et cetera. Building automation systems provide the means for properly coordinate these, e.g. by a central control unit. To be able to facilitate the communication between the interacting parts and devices from different vendors, they also have to be compliant to a common communication protocol. Such is represented by the Building Automation and Control Networks Protocol, called *BACnet* (ISO 16484-5, [289]) which is an interoperable communication protocol for building automation and control networks. When a piece of hardware or software is BACnet compliant, this means it can communicate with the hardware and software of any other BACnet compliant vendor. According to a senior engineer at Siemens [290], BACnet compliance is an essential aspect of every research and development project at major corporations like Siemens Building Technologies, Honeywell, Johnson Controls, Schneider Electric and others [145]. Further, according to [291] it is used by products of over 800 vendors.

According to a Siemens engineer [290], the standard processes for BACnet testing has evolved around exhaustive and was done in user-defined specification testing scenarios, conducted in contractual work with third parties, or company consortia. For example, there exists a BACnet Manufacturers Association which conducted interoperability tests in a NIST laboratory, according to [292]. To the best of my knowledge, our first steps for proposing a CT methodology for the BACnet protocol is novel and could aid the practitioners by utilizing the advantages of CT.

As also mentioned in [293], in order to capture, and transmit, the properties and the state that a specific hardware piece or process is in, the BACnet protocol uses *standard objects*. The way the standard objects are used to represent the underlying data and processes is left to the vendors. The BACnet protocol defines 60 standard object types, one of which are Event Enrollment Objects (EEOs). A *BACnet Event* can be characterized as any change in the value of any property of any object that meets a particular criteria. The purpose of an Event Enrollment Object is to define an event and offer the engineer an association with the occurring event and the transmission of notification messages.

APOGEE Insight®. The APOGEE Insight® [294] is a Siemens legacy building control product and is designed to provide building managers with the means to control a building from a central point - the *APOGEE Insight® workstation*. There a building manager can graphically monitor and control the building equipment and environment. Vendors and building control products of those vendors may have varying implementations for EEO configurations in the user interface of the APOGEE Insight® workstation. Through a graphical UI a building engineer can configure an EEO at the APOGEE Insight® workstation. Upon deployment, the EEO gets sent via the BACnet protocol to a so called *field panel*, which is connected to the actual hardware pieces, like thermostats or ventilators. If the EEO was configured correctly, the field panel signals `OK` and can translate the digital information carried in the EEO to (potentially) analog signals sent to the hardware pieces. However, in case an EEO with an invalid configuration gets transmitted and downloaded to the field panel, the field panel would signal a configuration `Error` for the EEO. In case of such an invalid EEO, the EEO can be modified at the workstation and downloaded again to the field panel. This process is clearly more time consuming when compared to that of valid EEO configurations.

To elucidate the work flows, lets consider the following example: A building manager commands a thermostat to heat to 30 degree Celsius in the graphical UI at the APOGEE Insight® workstation. An EEO gets created, specifying the change of state, the type of the event, the target value and whether the output of the field panel, i.e., the input to the thermostat has to be analog or digital. In the described case, the EEO would result in an alarm at the field panel, i.e., the configuration results in an `Error`, because the target temperature of 30 degrees is set too high and is out-of-range. The EEO would require to be re-configured by the building manager.

Due to its complicated nature, the EEO configuration was previously left open for the user, however, many of the configurations could be invalid. According to a senior

engineer at Siemens [145], there were above 5 million ways to configure an EEO via the graphical UI on a Siemens APOGEE Insight® workstation. In order to avoid unnecessary re-configurations of invalid EEOs, an EEO can be checked at the APOGEE Insight® workstation for validity with regards to certain constraints. An enhancement of APOGEE Insight® enforced constraints onto configurations of EEOs through the graphical UI of the workstation. Considering that via the graphical UI one EEO at a time can be configured and later be downloaded to the field panel, and that one configuration takes one second to execute, which is a conservative estimate by the engineer, the total effort spent towards exhaustive testing makes apparent the need for a more sophisticated testing methodology. Especially, when considering that the APOGEE Insight® workstation and its connection via the BACnet Protocol has to be tested newly for every vendor and the test setup needs a lot of time and resources.

More recent enhancements introduced the so-called *Harmonization Tool* to APOGEE Insight®, which provides the ability to mass configure EEOs and to send them to a field panel via the BACnet protocol [290]. The Harmonization Tool can significantly ease the testing process, as it allows a tester to specify hundreds of EEOs in a comma-separated values format (csv-format) and deploy the tests. These enhancements created the need to test the correctness of mass configuration of EEOs via the Harmonization Tool and their conformance with the BACnet protocol. In the following we present a CT approach to test the BACnet conformance of EEOs the Harmonization Tool.

4.2.2 Combinatorial Testing of the BACnet Conformance of the Harmonization Tool of the APOGEE Insight® Workstation

The Harmonization Tool provides already the essential means to automate a CT process: a generated test set, derived from a constrained CA which serves as a basis for specifications of EEOs, can be stored in csv-format and be executed through the Harmonization Tool. It configures the EEOs according to the test set and sends them to the field panel using the BACnet protocol. We make this testing process visible in Figure 4.1. The important last step in this testing process is the inspection of the EEOs status `OK/Error` at the field panel, which can be made visible through the APOGEE Insight® workstation, i.e., the EEOs status is uploaded from the field panel to the central APOGEE Insight® workstation. This last step closes a cycle resulting in a *round-trip testing strategy*, where the EEOs serve as a message in the BACnet protocol, and allows to test the BACnet conformance of the Harmonization Tool against the field panels [290]. This also allows us to use the information about the EEOs configuration status at the workstation as a *testing oracle* [295]. In particular, if an EEO results in a configuration `OK` at the field panel, the test represented by this EEO is `passing`; otherwise the test is `failing`. An overview of the workflows in the testing process is given in Figure 4.1.

This provides the essential means to automate a testing process for conformance testing of the BACnet Protocol implementation in the Harmonization Tool of the APOGEE Insight® workstation:

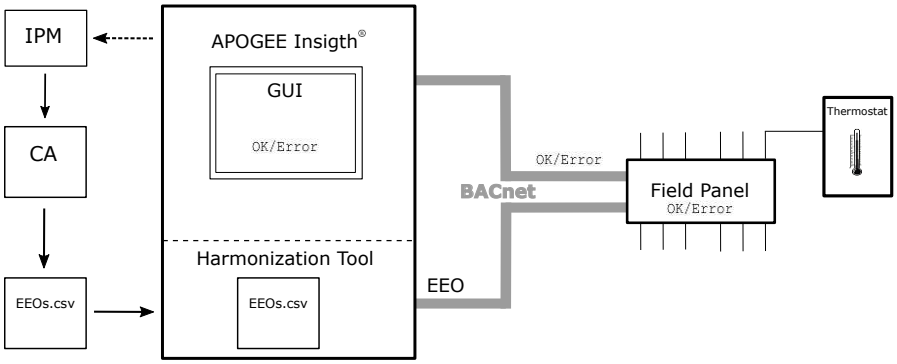


Figure 4.1: Testing process for testing the BACnet protocol implementation of the Harmonization Tool of the APOGEE Insight®.

1. Devise an IPM for the EEO configurations including the constraints as realized by APOGEE Insight®.

2. Generate a t-way test set for this IPM.

3. Input the t-way test set in csv-format to the Harmonization Tool.

4. The EEO configurations are downloaded to the field panels, where the field panels should show configuration "`OK`" if the EEOs were configured correctly, and "`Error`" otherwise

5. Finally, the status of the EEO configuration can be uploaded from the field panel to the APOGEE Insight® workstation, where it can be used for assessment:

   - EEOs should result in a signal configuration "`OK`" if they were configured correctly, representing a `passing` test case.
   - EEOs should result in a signal configuration "`Error`" if they were configured incorrectly, representing a `failing` test.

Due to the industrial nature of the tested real-world product, we do not provide explicitly the derived IPM. We can merely give an abstract description of the IPM, which consists of 7 parameters, each taking values from a domain of a size ranging from 4 to 37. Further, there are 5 constraints formulated between the parameters, which are logical connections of 2 to 8 parameter-value assignments. We used the ACTS tool [117], provided by the U.S. National Institute of Standards and Technology (NIST) [296], which is implementing the IPO strategy [174] for CA generation, to generate a 2-way test set for the devised IPM, which yielded 988 tests, each representing an EEO configuration. Exporting the test set in a csv-format it was ready to be used as input in the testing process depicted in Figure 4.1.

c_1	c_2	c_2	c_2
0	0	0	1
0	1	1	1
1	0	1	1
1	1	0	0
0	0	1	0

OT	ET	SPT	SPP
AI	FL	S	PV
AI	OoR	TL	PV
AO	FL	TL	PV
AO	OoR	S	ST
AI	FL	TL	ST

Figure 4.2: On the left hand side a covering array (of strength 2), and on the right hand side the derived 2-way test set for the IPM of the reduced EEO.

Example 4.2.1 *In order to still provide some motivation of the conducted testing process, we briefly sketch below how an IPM for a* reduced EEO *configuration looks like together with a combinatorial test set of strength two, also called a* 2*-way test set. A (reduced) EEO can be modeled consisting of the following parameters that can be configured assigning the respective subsequent values to them:*

- *ObjectType (OT): Analog-Input (AI), Analog-Output (AO)*
- *EventType (ET): Floating-Limit (FL), Out-of-Range (OoR)*
- *SetPointType (SPT): Schedule (S), Trend-Log (TL)*
- *SetPointProperty (SPP): Start-Time (ST), Present-Value (PV)*

A 2*-way test set for this IPM of the reduced EEO can be attained by computing a (binary) covering array (of* strength 2*) with four columns and replacing the entries in the columns with the values of the corresponding parameter. The resulting test set covers all* 2*-way combinations of parameter-value assignments, see Figure 4.2.*

4.2.3 Testing Results and Remarks

We have conducted our experiments with a derived 2-way combinatorial test set, based on the derived (full) IPM. The ACTS tool [117] returned a 2-way combinatorial test set with 988 abstract tests. The corresponding EEO configurations have been executed via the Harmonization Tool of the APOGEE Insight® workstation. During the entire testing process no failures have been revealed, i.e., all EEOs were configured correctly at the Harmonization Tool. Hence, unfortunately it was not possible for us to improve the quality of the BACnet implementation in the Harmonization tool of the APOGEE Insight® workstation. Nevertheless, we have contributed by performing 2-way combinatorial testing, providing the guarantee that all 2-way parameter-value combinations respecting the constraints have been tested.

This case study presents how CT can be applied for testing conformance with the BACnet protocol and thus it shows a way how CT can be used within the realms of smart building protocol testing. The absence of any positive testing results, i.e., the revelation of any

failures, may be due to the legacy of the SUT, which has been examined, tested and used many times, by different engineers, around the globe. However, in the next section we will see an application of CT that lead to an immediate benefit for the applicants, by revealing previously unknown failures in a more novel software product.

4.3 Applying Combinatorial Testing to Adobe Analytics

In this section, we report the practical application of CT to the data collection, compression and processing components of the Adobe analytics product, an SUT modeled with over 2000 input parameters.

The work presented in this section is based on the publication [146], which represents the documentation of the application of CT to the largest SUT in an industrial setting – where largest relates to the number of parameters in the IPM. *I contributed to the combinatorial test set generation, making possible this practical application beyond the limits of applied CT at that time. In particular, this pertains the use of the combinatorial doubling construction for the generation of the combinatorial test sets. As well as the generation of smaller combinatorial test sets used as* seed arrays *in this construction.* The input space modeling, test execution and also the evaluation was performed by other contributors, especially our industrial partners. Nevertheless, having set out clearly my contribution, in the following subsections we still consider the application of CT as far as detailed in [146], in order to give a more holistic impression of the application of CT, including the hurdles in such applied settings.

Therefore, in the following we will first provide some background on the Adobe Analytics tool, a product of Adobe Inc., in the following briefly referred to as Adobe. We then report some details of the input parameter modeling process and test value selection conducted by coauthors of [146], to provide more context for the testing problem faced at Adobe and how CT provides the structure to improve the validation of Adobe Analytics. Thereafter we describe how the combinatorial test sets were created. Finally, we report on the execution of the created test sets at Adobe, which revealed previously unknown failures and in addition had some unexpected, positive, side effects.

4.3.1 Introduction and Background to Adobe Analytics

We start by giving some background information for the system under test (SUT), i.e. Adobe Analytics, reflecting its development, previous testing efforts, and outline the faced challenges, also with regards to the application of CT.

Originating from web analytics, the Adobe Analytics product has evolved into a customer marketing platform allowing users to instrument data collection across many digital platforms for real time reporting. For the remainder of this section, we consider the Adobe Analytics as composed of two components: the *data collection and compression* component and the *data processing* component. The collected data are the main input for the data collection and compression component, which output is then submitted to

the processing component. Users of Adobe Analytics configure the amount and type of data to track. Hence a user can largely influence the input to the Adobe Analytics components, potentially leading to a high dimensionality of collected and processed data.

Data Collection and Compression. For the analytics product, Adobe provides customers with a software development kit (SDK). Customers use the SDK to instrument their own sites and applications. Depending on implementation, the instrumented applications then send data to the Adobe Analytics data collection pipeline. For a large customer, this can exceed 2000 input parameters. Being largely user-defined, the input parameter values approach practical limits. Eventually, this pipeline converts the data into a columnar format where it waits to meet certain conditions. Once meeting the conditions, the collection system exports the data to the compression algorithm. This algorithm transforms the data to another format for long term storage.

Data Processing. The processing system reads the compressed files and transforms the data for reporting. Being similar to the collection and compression system, this system uses the same CAs for over 2000 parameters but has different constraints.

Previous Testing of Adobe Analytics. As the evolution of the Adobe Analytics product, the number of configurable elements has increased to at least a few thousand just for these components. Given this domain knowledge, traditional validation of these components relied on randomly generated values for the data input parameters. This approach was generally seen as a practical solution to exercise the input space based on the assumption that the input space was too broad to be systematically covered. However, over time detected faults in these components exposed interactions not covered by the traditional approach. These faults revealed the insufficiency of this existing validation method.

Combinatorial Testing as a Supplementary Method. A key observation of CT maintains that software failures are generally caused by the interactions between a limited (small) number of input parameters [25]. Due to increasing attention from both industry and academia [28], CT was recognized as a viable alternative to random testing at Adobe and investigated for improving existing validation. Existing reports on industrial applications, such as [297], prompted the internal tools team at Adobe Analytics to apply CT to provide better values for the data collection input parameters.

In the following we report an industry application of CT to the data collection, compression, and processing components of Adobe Analytics, intending to improve existing validation. *The effectiveness of CT is thereby measured at Adobe in terms of new failures found, in comparison to the traditional random approach, rather than detecting known defects in previous faulty versions.*

Collaboration Setup and Approach. Applying CT to these subject systems heavily depends on the ability to generate CAs for the large number of input parameters and input parameter values. It is important to stress once more, that the subject systems can vary in size in terms of lines of code, but all have the characteristics of having a large number of input parameters with complex constraints and many possible values. As mentioned, the subject systems can exceed 2000 input parameters which can take over 300 potential unique values. Given that the number of tests in a combinatorial test set is lower bound by v^t and by $\log(k)$, see also [25], the number of rows in the CAs quickly approach impracticality. The high dimensionality of the processed data has two main implications, representing challenges for the application of CT

1. No tool existed that supported the generation of CAs for so many input parameters.
2. The number of values for the input parameters needed to be minimized.

Motivated by the practical findings of CT [298] [25], the aim of the engineers at Adobe was to perform 6-way testing. Where the idea was to start with 2-way testing, continue with 3-way testing, iteratively increasing the strength until strength six was reached. However, existing tools would not succeeded in generating the desired CAs of strength t for such a large number of input parameters and input parameter values. The following Subsection describes how these challenges were addressed.

4.3.2 IPM Creation and CA Construction

To overcome these challenges, Adobe engineers contributed their domain knowledge of the subject systems and minimized the number of parameters and their possible values, while the researchers from NIST and SBA Research generated the CAs for the created IPMs.

Input Parameter Modeling. Due to the large number of parameters and potential values, the input space for these subject systems is largely unknown. Fortunately, the data collection system temporarily stores the data in database tables, so that it was possible to used the description of these tables to infer the input space and parameters. Although critical to the modeling efforts, the database table descriptions could not encapsulate all the constraints of the subject systems and the respective input parameters. Consequently, the input space model was refined over several iterations. Furthermore, the table descriptions provided the data types and sizes which proved useful for selecting input parameter values.

Parameter Value Selection. Being defined by the user, the actual input parameter values can become intractable especially when considering combinations of these values. Consequently, the aim was to minimize the IPM by limiting the possible values for each input parameter, while maintaining a representation of the input space. Again, using the database table descriptions, we minimized the input parameter values by limiting the possible values for each input parameter. The data types and sizes from the table

descriptions provided an intuitive way to accomplish this by using the boundary values of the input parameters [299]. This way, over the course of this application, several IPMs of different complexity have been generated. To make one of the followed modeling approaches explicit, for each parameter of the IPM, i.e. each column of the desired CAs, three values were selected from the database table descriptions:

- *Minimum value*: The minimum value for number-based database primitives was easy to determine.For character-based primitives, we minimized both the length and byte sizes.
- *Maximum value*: The maximum for number-based primitives was also easy to determine. For character-based primitives, we inversely maximized the length and byte sizes. This maximization does not come without consequences, as it limits this application of CT to 3-way interactions. Any t-way interactions above this contained maximum value that exhausted the memory resources of the machine to execute the test cases.
- *Unset*: In addition to minimum and maximum values, we also used unset values where not prohibited by the database table description.

This approach resulted in an IPM with 2127 parameters, where each can take three different values. Also IPMs reflecting a more fine grained input space have been derived, in particular two IPMs both having again 2127 input parameters, which can all take seven and ten values respectively.

Covering Array Generation. Aiming to perform 2-way up to 6-way combinatorial testing for the SUT, the greatest challenge was to create CAs of the desired strengths for the 2127 parameters, with alphabet sizes $v = 3$, $v = 7$ respectively $v = 10$ in this setup. In order to do so, we deployed a combination of a greedy algorithm together with theoretical constructions of CAs, which we briefly describe as follows. Due to implementation details, available tools at the time of the study [146] were only capable of generating CAs for a maximum of 255 parameters. Aside from this, memory issues were expected to arise for such a large number of parameters during execution of the used FIPOG implementation when computing of higher strength CAs for the 2127 parameters. This is due to the fact that in later column-extension steps of FIPOG. for example, more than $\binom{2000}{4} > 6 \cdot 10^{11}$ 5-way interaction need to be stored in memory when computing a CA of strength 5. Thus, for the construction of the t-way CAs with 2127 parameters (and 571 parameters), we devised an approach that combines the greedy FIPO algorithm for CA generation, described in [180], with a combinatorial doubling construction, referred to as *Roux-type* construction described in [300]. To elaborate, we briefly describe such a *Roux-type* construction in its most general form for arbitrary strength t, and refer the interested reader to [300] for its proof, and more specialized versions for strengths $t \leq 5$. The *Roux-type* construction underlies the proof of the following Theorem.

Theorem 4.3.1 *[Theorem 4.13. in [300]] For any integers $t \geq 4$ and $v \geq 2$ we have* $\mathsf{CAN}(t, 2k, v) \geq \mathsf{CAN}(t, k, v) + (v-1)\mathsf{CAN}(t-1, k, v) + \sum_{i=2}^{t-2} \mathsf{CAN}(i, k, v)\mathsf{CAN}(t-i, k, v)$

Carrying characteristics of a combinatorial-multiplication construction, the *Roux-type* can be described by a combination of horizontal and vertical juxtaposition of a CA of different strengths, leading to an array with $2k$ columns. To illustrate the idea of the construction, we give a graphical representation of it, parallel to the first step in the proof [300], where we refer for more details. Assume for $i = 2, \dots t$ let

- A_i denote a CA of strength i, in particular a $\mathsf{CA}(N_i; i, k, v)$,
- define B_i to be the $(N_i N_j) \times k$ array obtained by repeating each row of A_i exactly N_j times,
- define C_j to be the $(N_j N_i) \times k$ array obtained by vertically concatenating exactly N_i copies of A_j.

Further define

- $R_t = (A_t, A_t)$ as the horizontal juxtaposition of two copies of A_t, and
- $R_{t-1} = \begin{pmatrix} A_{t-1}, A_{t-1}^{\pi} \\ A_{t-1}, A_{t-1}^{\pi^2} \\ \vdots \\ A_{t-1}, A_{t-1}^{\pi^{v-1}} \end{pmatrix}$, where π is a cyclic permutation on the set $[v]$,
- and for $i = 2, \dots t-2$: $R_i = (B_i, C_{t-2})$

Then $R = (R_t, R_{t-1}, R_{t-2}, \dots, R_2)^T$ constitutes a $\mathsf{CA}(N; t, 2k, v)$ with $N = N_t + (v-1)N_{t-1} \sum_{i=2}^{t-2} N_i N_{t-i}$. The CAs A_i for $i = 2, \dots, t$ that are required to perform the Roux-type construction described above are referred to as *seed arrays* in the following.

CAs generated with the Roux-type construction are not necessarily optimal, however, in cases where available methods reach the limits of usability, it provides a viable way to generate CAs with a larger number of columns. Especially, when considering that the IPMs consisting 2127 parameters required not only one, but multiple Roux-type constructions doubling the number of columns in the resulting CA in each step. The required seed arrays were generated using the FIPO algorithm as described in [180].

To facilitate the automated construction of CAs using the Roux-type construction in combination with the FIPO algorithm, we implemented a script that takes as an input the parameters of the target CA (i.e. its interaction strength, number of input parameters and values), the maximum number of doublings, and parameters limiting the maximum number of input parameters in constructed seed arrays. The program first finds appropriate parameters for seed arrays. It then repeatedly executes the doubling construction described in [300] until the desired number of input parameters is reached. In each step, it either reuses existing CAs or generates them on the fly using FIPO, and finally stores the generated target CA into a CSV file. We also note that the CAs generated with this approach have $k \geq 2127$ columns, however, any columns go beyond 2017 can simply be omitted.

Combining the flexibility of the efficient generation of CAs of strength t, for a comparatively small number of parameters and the capabilities of theoretical constructions to construct CAs for a large number of parameters, it was possible to construct CAs for the 2127 input parameters of interaction strength $t = 2$ up to $t = 5$ where the input parameters have uniform alphabet sizes $v = 3$, $v = 7$ or $v = 10$ respectively. The analog was done for the IPMs with 751 input parameters of alphabet sizes $v = 6$ up to $v = 25$ each. Last, we want to mention that we also constructed CAs for the IPM with 20 parameters with alphabet size $v = 10$, as a direct computation of the FIPO algorithm [180], so to provide the means for testing the SUT using smaller input models. All generated CAs are available publicly online [301]. To the best of my knowledge this is the first time that such large t-way CAs have ever been used in terms of an industrial application.

4.3.3 Application and Results

In the following we report on testing two components of the Adobe Analytics product, based on the CAs generated for the IPMs consisting of 2127 parameters and alphabet size $v = 3$, as described in the previous subsection. In addition to the tests specified by the rows of the CAs, two additional test cases were added to the CAs, to ensure their appearance. First, a test where all input parameters are set to minimum values, and second a test where all input parameters are set to maximum values.

We used the two main parts of the Adobe Analytics data collection pipeline as SUTs: First, the data collection and compression component, and second, the data processing component. For that, the test cases are first executed against the data collection and compression component. After successfully executing these test cases, the resulting compressed files, i.e. the outputs of this component, were used as tests against the data processing system. As mentioned above, we thereby aimed at starting with combinatorial test sets based on CAs of strength 2 and then move to CAs of strength 3, and so on, until either a fatal failure is detected or the testing at the current strength does not detect any failures that were not detected while testing at lower strengths. However, the combinatorial test set based on the CAs of strength 4 exceeded the memory resources of the machine used to execute the test cases, even after minimizing the input space, i.e. for the IPM with the lowest alphabet size $v = 3$. Engineers at Adobe suspect that this exceed of memory may be due to the 4-way interaction of maximum boundary values. However, investigations have not yet concluded whether the system should behave in this manner. *Regardless, the successful use of CT with 2-way and 3-way test sets revealed the existence of previously undetected faults. The remainder of this section details these findings.*

Testing of the Data Collection and Compression Component

As stated, we began with combinatorial 2-way testing. This did not work immediately as we discovered additional, undocumented constraints while attempting to submit the test cases to the data collection system. Although simple, finding these undocumented

constraints *marks unplanned success*. Adding these missing constraints will prevent future faults caused by incorrect implementations based on bad documentation. After successfully writing the combinatorial 2-way test cases into the database tables, the data collection system exported the columnar data to the compression algorithm. However, the compression algorithm core dumped immediately. Originally thought to be of little significance, further investigation revealed otherwise. The compression algorithm declares a pointer to a bit array used to track the occurrence of input parameter values. This buffer initializes to a predetermined size that facilitates over 2000 active input parameters (i.e. a parameter with a non-null value). However, the 2-way test cases exposed interactions where the number of input parameters exceeded the size of the array thereby causing a buffer overflow. Detecting this fault consequently requires the simultaneous use of more than 2000 input parameters. However, the values for the more than 2000 parameters do not matter. So, many combinations of input parameters could have revealed this fault provided the total number of input parameters exceeded the size of the array. Nevertheless, CT provided the formal validation approach that systematically detected this fault. As shown, the compression system could not even accommodate the interactions of higher-order input parameters from the 2-way test set. This first application of CT detected a significant fault before any users did. Consequently, this initial attempt demonstrated enough success to warrant approval for additional applications of CT f Adobe.

Data Processing After successfully compressing the data which was specified based on the CAs of strength two and three, we submitted the compressed files to the data processing system. Consequently, this system uses the same CAs with 2127 columns. This process took more iterations than the application of CT to the previous component, despite reusing its output. The reason for this is that the application of CT to this system discovered many more, but less significant faults. *It is important to note that these faults were not nor ever would be detected using existing validation for this subject system.* Consequently, the application of CT still proves responsible for the detection of these faults by providing the formal framework for the validation approach that allowed detection. The faults detected in this system largely related to inadequate data input validation. Similar to the initial issue with the previous SUT, there were several undocumented constraints. Violating these constraints caused test cases to prevent successful execution of the program. Some of the undocumented constraints were valid. These constraints were documented to avoid failures from future implementation errors. However, other failures caused by parameter interactions were due to invalid input parameter handling. Code was changed to appropriately accommodate these valid parameter interactions. For example, a set of columns was defined in the database tables as carrying data of the type `varchar`, but the processing system expects specific values that it interprets as `varchar` values. Table 4.1 summarizes the detected failures within this system.

Initial results of this CT application found new faults in each of the subject systems after only a small number of test cases have been executed. For example, a significant fault was detected in the data compression algorithm by the 2-way combinatorial test set after only roughly 150 tests. Therefore, measuring the effectiveness of CT by means of newly

Interaction strength t	*Failure Description*	*Failure cause/Fault*	*Fault resolution*
2	Flag-type fields throw error	Undocumented value constraint	Update input space model
2	Event-type fields throw error	Undocumented format constraint	Update input space model
2	Parser throws error (CDS)	Undocumented value constraint	Update input space model
3	Parser throws error (JSON)	Undocumented format constraint	Add input validation
2	Invalid date fields interaction	Undocumented value constraint	Update input space model

Table 4.1: Process Failures Detected using CT. The data processing component was tested using test cases based on the CA with 2127 columns and with alphabet sizes $v = 3$ [301].

detected defects as mentioned in the introduction to this section (Subsection 4.3.1), the results suggest that combinatorial testing may prove more effective than the traditional random approach.

In the next section we describe an application of CT to a previously unexplored domain, which may open the opportunity for future industrial applications of CT.

4.4 Applying Combinatorial Testing to Test the Consistency of Large Language Models

In this section we present an application of combinatorial testing to the testing Large Language Model (LLM)s and explore its applicability for consistency testing in an experimental evaluation. We thereby lie the focus on the modeling and the creation of an IPM for the input to LLMs, i.e., for natural language sentences. This was also the main contribution of mine to the publication [147] that is underlying this section.

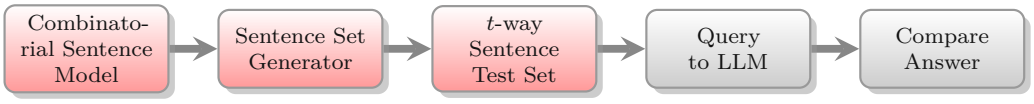


Figure 4.3: An overview of the proposed testing approach, instantiating a simplified generic CT process (see also Figure 1.3) for LLM testing. CT dependent steps are colored in red, LLM specific steps are colored in gray.

To give an overview, in the following we present how the simplified CT process (Figure 4.3) can be instantiated in order to be applied for the testing of LLMs. In particular, we present:

1. An approach to derive an IPM from a sentence which is given as input to an LLM;
2. A method to derive a *combinatorial test set* based on a given sentence and a CA;

3. An initial experimental evaluation, where we apply pairwise CT to two SUTs.

My contributions were in particular: devising the applied modeling methodology for creating IPMs for natural language sentences, supporting the IPM generation process and the test set generation, contributing to the instantiation of the CT process for testing LLMs, and aiding the presentation and interpretation of the experimental evaluation results. Having set out my contribution, in the following we still present the entire application of CT for testing LLMs as in [147], to provide a better impression of this pioneering application of CT.

We will first provide some background on recent development of LLMs and Artificial Intelligence (AI) in general, before we give some related work on testing and consistency testing of LLMs (and AI in general). We then give a general description of our approach for applying CT to consistency testing of LLMs. We offer a proof of concept of our proposed approach in Subsection 4.4.4, instantiating the CT process using one benchmark set of sentences and two LLMs as SUTs in an experimental evaluation. Finally, we discuss threats to validity of this approach and, when applicable, potential ways to address them.

4.4.1 Background on Recent Developments of AI and Large Language Models

Several researchers agree that we are currently experiencing the so-called *third wave of AI* [302]. Recent developments seem to have yielded a breakthrough in the capabilities of large language models (LLMs) to engage in human-like conversations. Such AI systems currently attain broad attention throughout society, being highly discussed in research, frequently covered by the media and subject to political discussions [303, 304]. The recent increase in attention to LLMs is not least due to the popularity of ChatGPT and its Plus variant, which makes GPT-4 available to a broad audience [305].

The recent increased capabilities and improved accuracy of AI systems, generally come at the cost of increased complexity of the utilized models, which also require increased efforts for learning. The resulting financial and environmental costs, in terms of spent money for training and caused CO_2 emissions, are approximated by Strubell et al. [306]. For training of the base version of BERT [306], they estimate a CO_2 emission of 650kg, and an expense of 3751$ to 12571$. *Such numbers emphasize the high costs and large data sets required for training and testing modern AI models.* As also mentioned by Khashabi et al. [307], LLMs require large training sets, which are expansive to create. For example, in order to create the BoolQ data set as described in [308], a number of almost 15 942 sample questions have been gathered and got processed by an *independent (human) annotator* who first evaluated a question for comprehensibility, then paired it with a paragraph from Wikipedia, that was identified as containing enough information to answer the question, and finally annotated with the correct answer.Aside from pure training of AI systems, also their testing becomes more important. As is also stressed by Wotawa [309], with the growing importance of AI systems, there is a need for developing appropriate testing and quality assurance measures.

In contrast to previous generations of AI systems, novel LLMs offer advanced capabilities to conduct human-like conversations. Studies as conducted by Jung, Dong and Lee [310] have shown that AI systems that act more similar to humans elicited greater trust from the interacting participants.

Experiments as conducted in [311] show that human-like behavior is essential for establishing resilient trust, i.e., a higher resistance to losses of trust in case of an automation failure. The authors of [311] thus conclude that the degree to which AI exhibits human characteristics is a critical variable and carefully considered when creating novel AI or when reasoning about the trust of humans in AI systems.

Although the capacity of novel AI systems to have human-like conversations is impressive, a factor that should not be underestimated with regards to trust in and the acceptance of AI systems is *consistency*. There exist studies arguing that consistency is one of the most important factors in establishing trust. In that regard, Dunn [312] performed two experiments evaluating three aspects of trust and concludes that consistency is the most important element in engendering cognitive-based trust in a dyadic relation.

Coming back to LLMs, conducting a consistency analysis of ChatGPT, respectively of GPT-4 [313], Jang and Lukasiewicz [314] state that especially when testing for *semantic consistency* of LLMs, it is extremely challenging to cover all possible variations of formulations of input queries.

Motivation. In light of these developments, the need for data sets and the challenge of consistency testing, we investigate the use of CT for testing the consistency of LLMs. Our goal is to *model* the problem of semantic consistency testing of LLMs, so that we can apply CT methods for generating semantic consistency tests. Where the idea is to benefit of the advantages of CT with regards to the guaranteed degree of input space coverage induced by combinatorial t-way coverage. The work presented can thus be understood as part of a larger effort that aims towards developing and establishing CT based methods for (consistency) testing of LLMs or AI systems in general.

Methodology. Our approach takes as an input a sentence, e.g. a test question from an existing training data set, and allows to diversify the input to a chosen degree of *coverage* by deriving additional test sentences. Thereby the focus is to maintain the semantics of the sentence, such that a potentially existing correct, expected answer is also valid for any of the derived sentences. *For achieving this, we will consider the replacement of words by a synonyms, aiming to maintain the semantics of the given sentence.* Clearly, the validity of this property highly depends on the "quality" of the considered synonyms. Threats to validity of our proposed approach are closely related to this technique and are discussed in Section 4.4.5. However, in order to ease the presentation and comprehension of our proposed approach, we assume for the largest parts of this paper that the replacement of a word with a synonym does not change the semantics of a given sentence.

4.4.2 Related Work on (Consistency) Testing of Large Language Models

Jang and Lukasiewicz [314] present a consistency analysis of ChatGPT. They analyse the LLM's capabilities to generate logically consistent responses with respect to three types of equivalence: semantic equivalence, logical negation, and symmetric consistency. From their experiments the authors conclude that ChatGPT achieves higher amounts of consistency when it comes to negated expressions and the usage of antonyms, compared to other pre-trained LLMs (like BERT [306]). However, they find that with regards to symmetric and semantic equivalence the LLM frequently returns inconsistent responses. Their findings suggest that ChatGPT fails to generate logically correct responses, despite its seemingly high degree of language understanding. As already mentioned, during their studies, the authors of [314] identified the problem that especially when testing for more complex consistency types, such as *semantic complexity* it can be challenging to cover all possible variations input queries.

The approach of mutating input sentences to a LLM while preserving the meaning or sentiment has recently garnered attention. The work by Gardner et al. [315] proposes the generation of *contrast sets* from standard test sets for supervised learning by manually perturbing the test instances, leading to a more accurate and comprehensive assessment of a model's linguistic capabilities. Similarly, Khashabi et al. [307] propose a novel method for generating training datasets by applying *human based natural perturbations* to a small scale *seed dataset*. The authors evaluate their approach utilizing the BoolQ dataset [308], finding that it improves the robustness of LLMs.

Ruane et al. [316] put forward a framework for using *divergent input examples*, generated by altering a textual user utterance while still maintaining the original intent, for testing the quality of conversational agents. In a continuation, Guichard et al. [317] propose and evaluate an approach with regard to the utilization of paraphrases. They generate divergent input examples by processing the input data through lexical substitutions, i.e. replacing words with their synonyms, using the Oxford Thesaurus for the retrieval of synonyms. Liu, Pang, and Fan [318] investigate how employing queries from a federated pool of synonymous questions can improve the capability of LLMs. The authors present two different methods deduced from two common user scenarios, comprised of synonymous questions featuring the same parameters or synonymous questions featuring different parameters.

Existing paraphrase detection approaches often rely on word overlap and syntactic similarities, which may not capture the true inferential properties of sentences. Nighojkar and Licato [319] propose an adversarial method called the *Adversarial Paraphrasing Task* (APT) for creating a paraphrase identification dataset, which aims to train models to identify paraphrases by considering the inferential properties of the sentences rather than relying heavily on lexical and syntactic overlap, leading to an enhancement in performance.

The aim of the authors was to create a high-quality paraphrase resource that can be used to improve the performance of various natural language processing applications. PPDB is the result of a process involving the extraction of lexical, phrasal, and syntactic paraphrases from vast bilingual parallel corpora. They have also computed distributional similarity scores for each paraphrases. Leveraging the vast resources of Google n-grams and the Annotated Giga word corpus, these scores offer a measure of the contextual and semantic similarity between paraphrases.

Incentivized by previous paraphrase collection efforts and work on the use of paraphrases for various natural language applications, Ganitkevitch et al. introduce the paraphrase database (PPDB) [320], which is a large-scale resource containing millions of paraphrase pairs, and describe its construction process via extracting lexical, phrasal and syntactic paraphrases from large bilingual parallel corpora and computing distributional similarity scores. Li et al. [321] explore paraphrase generation with deep reinforcement learning, training a model with rewards from a paraphrase scoring function based on the PPDB. Their work includes a generator-evaluator framework, two evaluator training approaches and developed techniques to enhance generator and evaluator learning, thereby improving paraphrase generation performance and accuracy while preserving meaning and linguistic variations.

Bozic [322] presents a metamorphic testing approach for hotel booking chatbots, in which an ontology is used for representing formalized knowledge of a bot's domain, generating natural language inputs and processing outputs. The results indicate that a metamorphic testing approach can detect unexpected behavior and that the used ontology can provide evidence on what type of information caused issues during test execution.

4.4.3 Instantiating the Combinatorial Testing Process for Testing the Consistency of Large Language Models

In this subsection, we describe the application of the CT process fot testing the consistency of LLMs. To briefly outline our approach, we start from a given sentence, e.g. from a training or evaluation set for LLMs. Interpreting each word of the given sentence as a parameter of an IPM, we generate from it several derived sentences by replacing words with synonyms according to a combinatorial test set. The derived sentences can later be submitted to an LLM. Provided an *annotation* of the given sentence, such as a `true/false` assignment to a statement or a correct answer to a question, we use such an annotation to obtain also annotations for the derived sentences, which can subsequently be used for a test oracle to assess the LLM's responses to the derived sentences. Annotated sentences can be provided within benchmarks, e.g. in the BoolQ benchmark set [308], from which we select one sentence and use it as a running example in this section to demonstrate our approach. An outline of our approach is provided by Figure 4.3. Further, an overview of the correspondence between notions of LLMs and CT that is induced by our proposed approach is given in Table 4.2, and elucidated in this section.

A Combinatorial Sentence Model via an IPM

Bridging concepts from LLMs and CT, our key idea is that a given original sentence (can be a statement or question from a benchmark) gives rise to an IPM, by regarding each word in the sentence as a parameter and synonyms for that word as the corresponding parameter-values. For each word, we create a list of synonyms with at most v_{max} elements to avoid an uncontrolled size of the IPM, where the original word appears as the first element. It is also possible to consider no synonyms for a specific word, which means to consider the word itself as its only synonym, leading to a parameter in the IPM with only one value. Conceptually, our proposed approach is independent from the way *how* and from *where* these synonyms are selected. Note that synonym dictionaries are available from various sources and we refer to Section 4.4.4 for the specific choice taken for the experiments in this work. After collecting the synonyms for replacement for each word in a sentence, the corresponding IPM is determined by a set of parameters, each corresponding to a word in the original sentence, and their values, which are given by the respective list of synonyms for each word of the original sentence.

The derivation of the IPM can be summarized in the following procedural steps:

***Step 1*:** Select an *original sentence* (e.g., from a given benchmark set).

***Step 2*:** Each word of the sentence gives rise to one *parameter* of the IPM.

***Step 3*:** For each word of the sentence, create a list of synonyms:

- each synonym serves as one *parameter value* of the corresponding parameter in the IPM;
- the length of the list of synonyms reflects the parameter domain size and is bounded by v_{max};
- ensure that the original word appears as first element in the list.

LLM	CT
LLM instance	SUT
Sentence	IPM
Word	Parameter
Synonyms	Parameter-values
Querying the LLM instance	Test case execution
Annotation of a sentence	Testing oracle (Correct response)
Correct LLM response to query	Passing test case
Wrong LLM response to query	Failing test case

Table 4.2: Overview of the linkage between concepts from LLMs (on the left) and those of CT (on the right).

Example 4.4.1 *We illustrate the steps of our approach by means of a (running) example, where we consider the following concrete original sentence, the* 23*rd question of the training set in the BoolQ benchmark [308]:*

"can you drink alcohol in public in denmark".

This sentence consists of eight words, and hence the corresponding IPM has eight parameters. We set the maximal number of considered synonyms v_{max} *to three, and further for the words* 'can', 'you' *and* 'in' *we select themselves as their only synonyms. Table 4.3 shows the considered synonyms for each word of this sentence. The corresponding parameters* $p_1, \ldots, p_8$ *can take* $1, 1, 3, 3, 1, 3, 1$ *and* 3 *values, respectively. We emphasize again that the first value of each parameter is equal to the corresponding word in the original sentence.*

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
can	you	drink drinking booze	alcohol alcoholic drink alcoholic beverage	in	public populace world	in	denmark kingdom of denmark danmark

Table 4.3: The selected synonyms in the example are reasonable and largely maintain the question's semantics.

Generation of t-way Sentence Test Sets

We use an IPM, as described above, to generate a combinatorial test set that achieves coverage of all t-way combinations of synonyms, for a specified interaction strength t. For practical applications, there exist dedicated CT tools, such as CAgen [78], where a concrete IPM with specified parameter-values can be given as input, and a combinatorial test set of the desired interaction strength t is generated. Each row of the returned (abstract) combinatorial test set represents then a derived sentence and hence corresponds to one (abstract) test case. The fact that the first element of each synonym list is the original word has, together with the properties of the CAgen tool, the result that the first derived sentence in the combinatorial test set is equal to the original sentence; at least up to slight variations such as replaced separating characters, or capitalization, which may occur due to pre-processing. Collectively, all derived sentences in a generated combinatorial test set have the property that every combination of t synonyms for different t words appears within at least one of them.

Example 4.4.1 (continuing from p. 220) *The IPM depicted in Table 4.3 can now be used to generate the pairwise combinatorial test set given in Table 4.4 derived from the example sentence. The columns headed by* $p_1, \ldots p_8$ *represent the parameters, respectively the words, and each row* $q_1, \ldots, q_9$ *represents one test case, i.e. a derived sentence. Altogether, these derived sentences in the combinatorial test set achieve full* 2*-way coverage,*

#	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
q_1	can	you	drink	alcohol	in	public	in	denmark
q_2	can	you	drink	alcoholic drink	in	populace	in	kingdom of denmark
q_3	can	you	drink	alcoholic beverage	in	world	in	danmark
q_4	can	you	drinking	alcohol	in	populace	in	danmark
q_5	can	you	drinking	alcoholic drink	in	world	in	denmark
q_6	can	you	drinking	alcoholic beverage	in	public	in	kingdom of denmark
q_7	can	you	booze	alcohol	in	world	in	kingdom of denmark
q_8	can	you	booze	alcoholic drink	in	public	in	danmark
q_9	can	you	booze	alcoholic beverage	in	populace	in	denmark

Table 4.4: A pairwise combinatorial test set for the first question *"can you drink alcohol in public in denmark"* and its corresponding IPM with eight parameters. It consists of nine derived sentences, with the original sentence appearing as first row q_1.

i.e. they have the property that any pair of two synonyms for two corresponding different words of the example sentence appear together in at least one of the derived sentences of the combinatorial test set. To illustrate this pairwise coverage property, consider the parameters p_4 and p_8, for which we can verify that each pair in the Cartesian product $\{alcohol, alcoholic\ drink, alcoholic\ beverage\} \times \{denmark, kingdom\ of\ denmark, danmark\}$ appears in at least one of the tests $q_1, \ldots, q_9$. Taking the pair (`booze`, `kingdom of denmark`*), i.e. the synonym* `booze` *for the original word* `drink` *and the synonym* `kingdom of denmark` *for the original word* `denmark`*, we can see that it appears in the seventh derived sentence q_7. Similarly, the remaining pairwise coverage requirements can be verified.*

Sentence Test Set Translation and Execution

The generated combinatorial test sets now serve as a basis for test execution against LLMs. Thereby, each sentence given as a row of a combinatorial test set is translated to one *test case*, i.e. one query to an LLM. In order to obtain executable test cases from the derived sentences that yield a processable response, adequate prompt design is required; in this step, modifications are applied to the prompt in order to evoke a Boolean answer. We refer the interested reader to [323] or [324] for an exploration of this topic and to Section 4.4.4 for the specification of what has been used for the experiments presented in the following. The generated executable test case (i.e., the derived sentence combined with appropriate prompt) is submitted together with potential further configuration values to an LLM.

Example 4.4.1 (continuing from p. 220) *We translate the first derived sentence, i.e. test case q_1 in Table 4.4, to an executable test case. That is, for the sentence "*can you drink alcohol in public in denmark*", we add a prompt specific to the LLM called LLaMA*

(cf. Section 4.4.4) to obtain an executable test case, which yields:

```
can you drink alcohol in public in denmark.
the boolean answer to this question is
```
(4.1)

Test Oracle

Under the assumption that synonym replacement preserves the semantics of a sentence, the meaning of the derived sentences in a combinatorial sentence test set is equal to the meaning of the original sentence. Making use of this assumption, we can create a *test oracle* for the derived sentences based on an annotation for the original sentence, in case it is available.

For the specific case where the original sentence is a Boolean question with its corresponding annotation given by a `true/false` assignment, consider an executable test case obtained from a combinatorial sentence test set derived from this original Boolean question. When submitted against an LLM, we expect the LLM to deal with the truth content of the test sentence, and therefore, it should be possible to categorize its response into one of the three classes `true`, `false` and `undefined`. This categorization can be obtained by textual post-processing using keywords in the returned response. The test oracle then compares the assigned category with the given annotation of the original Boolean question: if the category of the response is equal to the `true/false` annotation of the original Boolean question, the test oracle decides this derived sentence to be a `passing test case`, otherwise it is a `failing test case`.

Example 4.4.1 (continuing from p. 220) *The response of LLaMA to the prompt given in* (4.1)*, is:*

```
can you drink alcohol in public in denmark?
The boolean answer to this question is Ã¢
```
(4.2)

After removing the submitted query from (4.2)*, the remaining part "Ã¢" is transformed to the empty string via the post-processing. Hence, the response is classified as* `undefined`*. To give another example, consider the response obtained after executing the test case obtained from the derived sentence* q_2 *of Table 4.4:*

```
can you drink alcoholic drink in populace in kingdom
of denmark?  The boolean answer to this question is yes
```
(4.3)

After removing from (4.3) *the query input to the LLM, the remaining response "*`yes`*" is classified as* `true`*.*

4.4.4 Experimental Evaluation

This subsection provides technicalities on a proof of concept implementation of the proposed approach together with some details on practical experiments using the BoolQ benchmark set together with two LLMs as SUTs.

Synonym Dictionaries and Combinatorial Test Set Generation of Derived Sentences

For the derivation of the IPM to a given sentence, as described in Section 4.4.3, we follow an automated approach for identifying synonyms for the individual words. We set the maximal number of synonyms to be considered to $v_{max} = 3$, in order to avoid unnecessary long lists or such with potentially sub-optimal synonyms in them. Further, to guard against the use of deficient synonyms, we do not consider *all* words for synonym replacement. In particular, we exclude words belonging to classes such as conjunctions, numerals, adverbs, pronouns, adpositions and determiners from synonym replacement. That is, for any words belonging to one of the aforementioned classes, we only consider themselves as their only synonyms. In terms of CT, they correspond to parameters which take only a single value.

To identify the different word classes in the original sentence, we analyze it via natural language processing (NLP) . This classification of words in the original sentence is performed using the Python library *SpaCy* 3.5.1[1] using the *en_core_web_sm* model version 3.5.0. For those words that are considered for synonym replacement, we first generate a list of potential synonyms using the wordnet module of the Python *nltk* library[2] version 3.8.1. Each word in this list is then converted to lower-case letters and duplicates are removed. Thereafter, all words that do not belong to the same class as the original word are removed from the list of potential synonyms. This additional filtering significantly increases the reasonability of the automatically generated synonym lists. Finally, we use the original word, together with the first two elements (recall that $v_{max} = 3$) from the potential list of synonyms to create the list of synonyms, i.e., in terms of CT, the values for the parameter corresponding to the original word. Performing the above described process for all words of a sentence yields an IPM. Subsequently, we use the devised IPM as input to the tool CAgen [78], [325] to generate a pairwise combinatorial test set, i.e. interaction strength $t = 2$.

SUT Description

We chose two LLMs, LLaMA [326] and T5 [327], based on criteria such as open availability as pre-trained models, prior use in research, and ease of use. As each LLM offers individual settings, we may consider an LLM together with its settings as the SUT. Where possible, we used Hugging Face's transformers Python package,[3] and we utilized the functionality to initialize the LLMs from a pre-trained state using the .from_pretrained() method.[4] In the case of T5, we decided to use the base model trained on the TriviaQA dataset[5] [328]. LLaMA was sourced via the LLaMA Cpp Port[6] of Meta's LLaMA to increase execution speed. Our tests were run with a version[7] dated to the 14^{th} of March 2023.

[1] https://pypi.org/project/spacy/3.5.1/
[2] https://www.nltk.org/
[3] https://huggingface.co/docs/transformers/index, accessed: 2023-05-03
[4] https://huggingface.co/docs/transformers/main_classes/model, 2023-05-03
[5] https://huggingface.co/docs/transformers/model_doc/t5v1.1, accessed: 2023-05-03
[6] https://github.com/ggerganov/llama.cpp, accessed: 2023-05-03
[7] Commit id: 47857e564c218a2c38346d0cdd94314632878fcb

For both LLMs, we utilized a prompt design similar to what is described in the LLaMA documentation [326] by appending the string "`? The boolean answer to this question is `" to the question, with the intent to guide the LLM towards emitting tokens which are easier to classify as a Boolean answer.

LLaMA and its corresponding testing pipeline was executed on an Intel Xeon E3-1275 v6 with 64GB RAM running Ubuntu 20.04.6 LTS, while T5 was run on a machine based on an AMD EPYC 7502P with 128GB RAM running Debian 10 (buster). All experiment data was stored in a PostgreSQL database.

Experimental Evalaution Results

In order to conduct our experimental evaluation, we automated the process of applying CT for testing the consistency of LLMs in an extensible testing pipeline that is open to additional LLMs, benchmark sets as well as test oracles. We applied our approach to all 9,427 questions in the `train.jsonl` file from the BoolQ benchmark set, generating pairwise combinatorial sentence test sets for each of them. Subsequently, we executed all derived questions from all combinatorial sentence test sets (90,976 sentences in total) against the two SUTs T5[8] [327] and LLaMA [326], which was sourced via the LLaMA Cpp Port[9].

The results are documented in Table 4.5. The upper half shows the confusion values for "annotation/response" in percent; cases where the response agrees with the annotation, the oracle decides that the tests are passing. The lower half shows the percentages of `true`, `false`, and `undefined` responses, as well as three different accuracy values:

- the overall accuracy (acc) shows the percentage of responses that agree with the annotation, i.e. the percentage of passing tests in terms of CT,
- the accuracy without undefined responses (accWUR) is computed when disregarding sentences where the response was `undefined`,
- and the average accuracies of combinatorial test set (accCA) is the average over the accuracies computed for each individual set of derived sentences, since different sentences can induce CAs with a different number of rows.

Both table segments depict the results grouped by LLM and set of sentences, in percent.

To outline the results of our experimental evaluation, first, both LLMs are not very accurate, and second, this is independent from the sentence sets. To elaborate, the high percentage of undefined responses by T5 (see column U in the lower part of Table 4.5) shows that it is difficult to steer T5 towards giving Boolean answers reliably. Considering that roughly 62% of the original questions of the BoolQ dataset are annotated as true, and the remaining 38% as false, this explains the low accuracy achieved by T5 in our experimental evaluation. Hence, this LLM exhibits a very low accuracy of only 2.02% for

[8] `https://huggingface.co/docs/transformers/model_doc/t5v1.1`, , accessed:2023-05-03
[9] `https://github.com/ggerganov/llama.cpp`, accessed: 2023-05-03

our set of derived sentences and 2.76% for the set of original sentences. In contrast, the settings of the LLaMA model together with the applied post-fix mostly yielded responses classifiable as Boolean value. Nevertheless, the accuracy is only 47.45% and 45.75% for original and derived sentence sets, respectively, which is not far from to a random fair coin flip for a Boolean classification task.

Overall, it seems that the accuracy (whther acc, accWUR) of the two LLMs, highlighted in **bold** was not meaningfully impacted by our derived sentences, since the accuracy values do not differ by more than 2% in each category. This could suggest that our combinatorial generation approach was successful in generating more sentences from a given set, without reducing its quality. However, the fact that the accuracy of both LLMs for the original sentences was either very low for T5, and close to that of a fair coin flip for LLaMA, relativizes this observation. This issue makes it infeasible to assess the suitability of the proposed method for testing the sentences for testing the consistency of LLMs in general.

SUT	sentences	T/U	T/T	T/F	F/U	F/T	F/F
T5	original	59.44	2.76	0.12	36.20	1.49	0.00
	derived	60.05	1.98	0.10	36.73	1.10	0.04
LLaMA	original	32.88	20.34	9.09	18.37	14.56	4.75
	derived	32.85	19.58	9.71	19.40	12.91	5.56

SUT	sentences	T	F	U	acc	accWUR	accCA
T5	original	4.24	0.12	95.64	**2.76**	**63.26**	-
	derived	3.07	0.14	96.78	**2.02**	**62.68**	2.15
LLaMA	original	51.26	34.90	13.84	**47.45**	**55.07**	-
	derived	52.25	32.48	15.27	**45.75**	**54.00**	45.45

Table 4.5: Confusion values in the top half of the table: showing the percentage of "annotation/response" for the original and derived sentences and for each LLM. Performance results in the bottom half of the table: showing the percentage of T = true, F = false, and U = undefined responses, as well as three measures of accuracies (i.e. percentages of passing tests in terms of CT): "acc" is the accuracy over all sentences; "accWUR" is the accuracy without undefined responses is the accuracy when disregarding sentences where the response was U; "accCA" is the average accuracies for derived sentences.

4.4.5 Threats to Validity

Throughout our experiments, we encountered a variety of issues in connection with the use of synonyms. We first provide details on these pitfalls before moving on to other threats.

Synonyms. A potential issue with the use of synonyms is that they may alter the meaning of the original sentence. This is particularly true for words that have multiple meanings or even syntactical uses (a good example is the word "run"). To give an

example that we encountered during our experiments, take the 26th original sentence from the BoolQ benchmark set:

"does the world cup final go to penalties" (4.4)

The IPM derived for this sentence consists of eight parameters, each of which can take either one or three values, and is visualized in Table 4.6. Amongst other issues, the compound noun "world cup final" is separated into three individual nouns and replacing any of them by a synonym changes the meaning of the sentence. This results in derived sentences of low or no comprehensibility such as *"does the existence cup final go to punishment"*. Further, the use of external synonym databases itself may prove a threat to validity, as assessing their quality is not trivial and beyond the scope of this work.

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
does	the	world universe existence	cup cupful loving cup	final concluding last	go travel move	to	penalties punishment penalty

Table 4.6: An IPM for an original sentence, where some of the synonyms change the meaning of the sentence.

Well-formulated Original Sentence. Some of the original queries may not be well-formulated in the first place. For example, the sentence above (4.4) does not specify which world cup finals are meant, and rules vary wildly across sports.

Small Number of SUTs. Finally, we only considered a small set of sample SUTs in this work and made use of specific pretrained LLMs. The reason for this is that the necessary versions of other LLMs, such as GPT-4 [313], that allowed for sufficient automation were not available. Additionally the achieved accuracies below 50% makes it difficult to evaluate the our method. However, as the primary focus herein is to propose a CT method for testing LLMs, we consider a large-scale evaluation as part of future work.

4.4.6 Future Work and Ideas to Address the Threats to Validity

We believe that large parts of these threats of validity can be addressed by adaptations to the proposed approach. For example, regarding the quality of synonyms, an important first step will be to group together words that compose a single term, like *European union*, or *world cup final* before any synonyms are created. Further, aiming to improve the quality of the synonyms we may query the SUT in a pre-processing step with this task, e.g. "`please list three synonyms for 'world cup final'`". This way should also guarantee to generate the synonyms in the most "SUT-friendly" way, i.e. the LLM is certainly aware of the used synonyms, which is not necessary the case when using external synonym data bases. In terms of CT, this way of finding the synonyms

comes equal to *asking the SUT what its values are* - an opportunity rarely met in testing. Finally, as for many other testing problems, we may rely on domain experts knowledge in order to create or improve the used IPMs. In this case, this could mean to use humanly created lists of synonyms for a given sentence, which would also allow to incorporate constraints in the IPM, e.g. to forbid or enforce certain combinations of words in a sentence to ensure the semantics stay intact.

Even more important than improving the method is to apply it to well or even high performing SUTs. We believe this is important, as this would allow to assess the methods quality in an empirical way, showing whether it is a viable way for enlarging test sets for testing the consistency of LLMs.

4.5 Summary

This chapter documents applications of CT in three domains that are different in nature: testing the implementation of a smart building protocol, testing (components of) a web tool for data tracking, and for testing the consistency of LLMs. The applications of CT for smart building protocol testing and LLM testing, represent applications of CT in novel domains. They have in common that they have *not yielded any advancement of the tested applications themselves*, respectively of the application domains, and should be seen rather as exploratory works, representing proof-of-concept applications of a pioneering stage. Nevertheless, the application of CT for testing the BACnet smart building protocol represents a case study of an industrial application. As such, it provides a proof-of-concept and shows a way how CT methods can be used in such settings and makes them available for future developments in that domain.

In Section 4.4 we have demonstrated the capabilities of the CT based method for enlargement of data sets for the use case of consistency testing of LLMs. A first experimental evaluation, at the very least, does not allow for a negative conclusion regarding the applicability of this method for the intended purpose: Applying our CT method led to a significant increase in available test data while leaving the accuracy of the responses roughly on par with those returned for the original test set. The proposed method bears potential for future exploitation also for data set enlargement for testing or learning of LLMs, which may be of interest for several reasons. First, some data sets may be effortful to create, e.g. due to relying on humans in the loop [308]. Second, existing data sets may not be trust-worthy, considering that the threat of data poisoning attacks is given [329], [330], [331]. Third, legal boundaries regarding what data can be used to train LLMs or artificial intelligence in general are not settled, as recent developments suggest [332], [333], [334] there may be legal issues when content owned by a third parties is used for training models. Such upcoming challenges, of a relatively novel application domain also represent potential for future contributions of CT in the domain of LLMs. The work presented above pioneers for such applications. We also note that, it has to be part of future investigations in how far the notion of "full t-way coverage" has a meaning or is of advantage, e.g. in comparison to full-Cartesian product constructions. The

exploration of the meaning of t-way interactions in the domain of artificial intelligence is ongoing, as for example the work of Patel et al. [284] shows.

In contrast to the novel application domains, the application of CT for web application testing, i.e. for testing Adobe analytics yielded progress to the application itself. The effectiveness of CT was measured in terms of new defects found rather than detecting known defects. The application of CT to the subject systems, as described in Section 4.3 led to the identification of undocumented constraints and *detected seven previously undetected defects*. By finding previously undetected faults, this application shows the great potential of CT to improve existing validation for the Adobe Analytics product, which traditionally relied on random testing. Furthermore, the application of CT to the subject systems provided by the engineers of the Adobe Analytics challenged ingrained assumptions and debunked long-held beliefs that the subject systems are too complex to practically achieve systematic coverage. Overall, the organization (Adobe) considers this application of CT successful enough to warrant continued augmentation of existing validation with CT. These findings represent a successful application of CT in a large scale use case in an industrial setting.

CHAPTER 5

Applications of Combinatorial Testing Fault Localization

This chapter is focused on applications of combinatorial testing fault localization (CT-FLA) methods. In such applications, we are not only interested in detecting the presence of a failure in a system under test (SUT), but we also want to *identify which t-way interaction triggered the failure*. Our intention is to extend the capabilities and to show the potential of CT-FLA methods in practical applications. We further demonstrate the versatility of CT-FLA methods by making them applicable for usages that go beyond software testing in a narrow sense, and explore how the concept of failure inducing t-way interactions (t-FITs), or more informally failure inducing tuples (FITs), can be utilized in other domains. In other words, the goals of this chapter are to

- devise new methods for CT-FLA,
- explore novel application domains of CT-FLA methods, and
- demonstrate the applicability of CT-FLA methods in these domains.

In the following, we will present two case studies of CT-FLA methods that go beyond software testing in a narrow sense: hardware testing, and testing automated driving functions. Further, we will explore the capabilities of CT-FLA methods in the more recently pioneered domain of *explainable artificial intelligence* where FITs appear no longer as the trigger of a failure, but as explanations of the output of an AI/ML system.

5.1 Related Work

The applications of CT-FLA reviewed in the following are structured in four paragraphs. The first pertains to *general* applications of CT-FLA. It is followed by one paragraph for each of the three application domains considered in the sequel of this chapter.

Applications of CT-FLA. Over the past decades a significant number of publications pertain the *application of CT* in real world scenarios, see e.g. [28], [56] or [57], however, the number of works on the *application of CT-FLA* methods is only rising in recent years. In the following we first review some works on applications of CT-FLA methods in general, whether in experimental evaluations or in industrial settings. Thereafter, we review applications of CT and CT-FLA specifically with regards to the application domains appearing in this chapter. We start by reviewing more closely two works we briefly mentioned in the introduction as part of the state of the art in Section 1.4.

With regards to adaptive CT-FLA, one of the most prominent contributions is represented by the BEN tool [130]. This approach consists of two phases, where the second phase can be performed in a white-box testing scenario, when the source code of the SUT is available. The first phase of BEN consists of analyzing an initially executed and annotated combinatorial test set based on statistical ranking of *suspicious* t-way interactions, i.e. t-way interactions that are likely to be failure inducing. The analysis also includes the identification of parameter-value assignments that are likely to be not involved in any FIT. These values are then used together with the ranking of suspicious t-way interactions to derive new tests, aiming to concretize and eventually identify the set of FITs. In the second phase, BEN selects a top ranked potentially failure inducing t-way interaction, a failing test containing the FIT and a small number of passing tests are generated in an iterated test generation and test execution process. Further, in case the source code of the SUT is available, in the second phase the execution traces of the generated passing tests are analyzed and compared to that of the failing test, and a final ranking of faulty statements in the code is generated. In another study [335], the BEN tool is applied in an experimental setup to real world programs, such as the Siemens suite and four real world programs, flex, grep, gzip and sed, from the software infrastructure repository [336], [337]. The authors conclude from their experimental results that the BEN approach can effectively and efficiently localize the faulty statements in these programs. Later, the BEN tool has been used for evolving an input model for security testing for XSS vulnerabilities [338], where the suspicious t-way interactions returned from BEN represent input combinations suspected to enable XSS vulnerabilities. Hence, the applied CT-FLA method is not particularly used for *combinatorial fault localization*, but rather for identifying input combinations that trigger a certain behaviour of the SUT.

The authors of [83] propose an interleaving approach to CT-FLA, where a failing test of an initial combinatorial test suite is immediately examined by executing new tests directly after it, aiming to identify the FIT causing the original test to fail. The rational behind this strategy is to identified FITs early and guide the remaining test case generation. This *interleaving approach* leads to a high frequent communication between the test

generation and the test execution unit, especially when there are many failing tests, or FITs. The proposed method is also extensively evaluated experimentally by applying it to five subject programs (Tomcat, Hsqldb, GCC, Jflex, TCAS).

The authors of [339] have integrated CT-FLA into an industrial framework, as part of the commercial tool IBM FOCUS [340]. Briefly summarized, the followed approach consists of replacing each position of a failing test (coming from a CA of strength t) with a *safe-value* that does not appear in any failing test. The underlying CT-FLA algorithm further relies on the assumption that the FITs are at most of the same length as the strength of an initial combinatorial test set, and that any two FITs *share* at least one parameter. Under the assumed preconditions the tool always returns a FIT. They report on a promising initial experience in applying it to an operating system where the existence of a bug was known due to previous CT. Thanks to their end to end automation, the FITs were correctly identified and reported to a test engineer without any manual intervention.

With regards to applications of non-adaptive methods of CT-FLA, Compton et al. [341] used locating arrays for screening interaction factors in a wireless network testbed, representing the application of locating arrays for screening a physical system. They utilize the separation properties of locating arrays designed for pairwise interactions to determine which factors play a significant role in wireless network performance with multiple performance metrics (response variables). For the analysis of the experiments, they adopt a signal recovery method via orthogonal matching pursuit together with statistical methods, to determine which factors are significant. They find that the results of the tests based on the locating array are able to rule out most insignificant factors and levels, which can reduce the number of experiments required in subsequent experimental testing.

Similarly, the use of locating arrays for level-wise screening in experiments is proposed in [342]. The proposed method uses locating arrays as screening design and the analysis focuses on the identification of level-wise main effects and two-way interactions. The validity of the approach is demonstrated by means of well-studied and synthetic data sets, including a wireless network test bed experiment.

CT for Hardwarwe Trojan Detection. Combinatorial testing has been previously used for hardware Trojan (HT) detection, where the circuit under test is treated as black box and CAs are used as test vectors for the circuits. The combinatorial characteristics of CAs then induce that the test vectors have the property that all input combinations of a fixed number of input gates are *covered* at least a predefined number of times in order to guarantee the excitement of all possible combinational HTs up to a certain length. In [269], the applicability of combinatorial testing to HT testing is demonstrated by means of testing a hardware implementation of the AES cryptographic algorithm. Since there are no constraints amongst the 128 bit vectors that represent the primary inputs to an AES module, the authors use methods of unconstrained combinatorial interaction testing to generate the test set. Their work highlights the efficiency of combinatorial testing, as it

provides test sets that are smaller in size by several orders of magnitude when compared to other approaches, while guaranteeing the excitement of specific combinational HTs up to a certain *length*. Testing focuses exclusively on the input gates of the circuit under test, as they represent the primary interface for testing and triggering a potential attack. This line of research was continued in [343], where the *completeness* of combinatorial testing in terms of excitement of combinational HT up to a certain length was compared against random testing. A series of experiments with different HT designs underpins the efficiency of combinational Trojan detection through combinatorial testing. A similar study on the effectiveness of CT for this task is presented in [344].

CT-FLA and Explainable AI. In recent years CT methods have also been used in the domain of AI. Kuhn et al. [272] laid out how combinatorial testing methods can be used for generating explanations of AI systems. The key idea is that t-way interactions that are present exclusively, e.g., in a single class identified by a classification system, can serve as an explanation *why* the objects of this class have been classified as such.

This conceptual connection between identifying failure-triggering input of software and finding explanations for the output of AI systems, as set out in [272], was applied in [345] for explaining the decisions of image classifiers. The authors present an approach that uses the BEN tool [130], to produce counter factual explanations for decisions made by ML models. Their approach consists of two phases, and starts by performing image segmentation of a given picture that was already classified. Next, each segment of the picture is mapped to an input parameters, which gives rise to an IPM corresponding to the picture, for which a binary CA is generated. Each row of the CA is used to derive a concrete test, by masking certain segments of the original picture, which leads to a new test image. If the classification of the test image matches that of the original picture, this is interpreted as a passing test, otherwise if the class does not match, it is a failing test. Provided this interpretation, the BEN tool can be used to identify failure inducing t-way interactions, which are used to derive counter factual explanations for the classification as set out in [148].

The work [272] of Kuhn et al. is continued and extended significantly by Lanus et al. [346], who propose a novel difference metric based on combinatorial coverage, for measuring and comparing machine learning data sets. The proposed *t-way set difference combinatorial coverage* quantifies over the t-way interactions that are present in a first data set, but are not present in a second data set. It thus provides a notion of a *directed distance* between data sets. The metric's applicability to combinatorial fault localization and explainable AI based on interpretable features in the data is demonstrated. Further, the notion of distance between data sets is of interest for transfer learning, leading to a measure of distance between the set on which a machine learning model was trained on, and the data set it is deployed on. In fact, an initial investigation shows that a higher SDCC value is correlated with a drop in performance in transfer learning.

CT and Autonomous Driving. The authors of [273] propose a CT approach to testing deep neural networks (DNNs) used in autonomous driving systems. They generate synthetic images that can be used for testing a pre-trained DNN, by identifying a set of transformations that can be applied to a given image for which the DNN's prediction is known. The set of transformations, e.g., linear transformations, affine transformations, or the change of the image's brightness level, is used to derive an IPM for the image as follows. Each transformation is mapped to an input parameter. The values of input parameters can be considered the parameters that specify the transformation, e.g., a rotation angle, or a brightness level, where also the meaningfulness (validity) of the transformations parameters is taken into consideration. Based on this IPM a pairwise combinatorial test set of synthetic images is generated and evaluated by comparing the DNNs prediction to that of the original image, where a deviation above a certain threshold indicates an inconsistency of the DNN model. From the results of an experimental evaluation, the authors conclude that t-way tests can identify a number of inconsistent behaviors in DNN models. In a follow-up work [347], the same authors evaluate their approach with regards to the neuron coverage achieved through the synthetic images obtained from their combinatorial approach and compare it to that achieved by DeepTest [348]. Their initial results suggest that their combinatorial approach results in higher neuron coverage.

The work in [349] proposes a general framework for testing automated and autonomous driving functions. The framework consists of three phases. First, an ontology, describing a test scenario is automatically converted to an IPM. For this process two different algorithms are proposed, and their advantages and disadvantages are discussed. Second, based on the derived IPM abstract combinatorial test sets can be generated, which are translated to concrete test cases by specifying the values. Third, the derived concrete test cases are executed and evaluated in a simulation environment.

The authors of [350] adopted a combinatorial three-way testing approach for generating parametrized scenarios, which can be used for the construction of test scenarios for, e.g., multi-lane road sections, intersections without traffic lights and roundabouts. Multiple safety indicators were used to identify critical test cases, which are of special interest for testing automated driving functions. Additionally a clustering method was used to obtain a representative set of critical test cases and to further reduce the number of test cases such that they could be used for closed-road testing.

Dhadyalla, Kumari and Snell [351] present the application of CT to a real hybrid electric vehicle control system. Messages of the Controller Area Network (CAN) protocol are intercepted and replaced with those of a combinatorial test sets. In their early results they find indication for the effectiveness of their approach in exposing incidents in system behavior that is not found during traditional functional testing.

Lastly, we want to mention that also sequential CT, in the form of sequence covering arrays, has been used in the automotive domain [352]. Dhadyalla et al. use sequence covering arrays for 16 events of strength three with 2688 rows for the testing of a real-time safety control function of a prototype electric machine control unit. In their experiments, they find evidence that suggests that the interactions of one specific input event with

other input events have significant impact on triggering failures, while the same input event does not trigger the fault independently.

There exist further closely related works on applications of CT in the automotive domain, however, as they are underlying the work presented in Section 5.4, they are reviewed therein.

5.2 Fast Combinatorial Testing Fault Localization for Hardware Trojan Location

This section documents a use case application of CT-FLA for locating hardware Trojans in cryptographic circuits, based on the publication [68].

We propose an algorithm for fast FIT localization in combination with a non-adaptive CT-FLA approach, when the assumption holds that there is a single FIT in the SUT. This method is then applied in a case study and realized as a logic testing approach for hardware testing, applying it to a field-programmable gate array (FPGA) implementing the AES symmetric-key encryption and decryption algorithm, that got tampered with a combinational HT. By virtue of the CT-FLA method it is possible to excite and *locate* hardware Trojans that are triggered by a combinational ℓ bit pattern in the primary input of the circuit.

In the following we start by providing some background on hardware testing, emphasize its importance and describe the threat model considered hereafter. We then expand once more on the contribution of this work, when considering it in terms of hardware testing, before we review some related work on logic testing and HT location. Thereafter, in Subsection 5.2.3 we outline how the concepts of CT can be linked to those of hardware testing and HT location. Subsequently, in Subsection 5.2.4, we describe three different algorithms for combinatorial fault localization of single FITs, formulated in terms of HT location using annotated test sets. The essential workings of the algorithms are illustrated by a running example. In Subsection 5.2.5 we describe the experimental set-up of our case study and in Subsection 5.2.6 we document our experiments regarding HT location based on combinatorial test sets and randomly generated test sets. We further discuss practical limitations of the presented work in Subsection 5.2.7, and in Subsection 5.2.8 we outline several directions of future work based on the methods presented herein.

5.2.1 Background on Hardware Testing

The security of information and communication technologies and electronic systems in general is often solely related to the security of its software part, leaving hardware security out. However, when treating the security of an electronic system holistically hardware security must be addressed as well. A reliable and secure piece of hardware is expected to implement and execute only what it is designed to and nothing else, even in the presence of an intentional attack. In modern society, the globalization of the semiconductor industry raises additional concerns regarding the authenticity and security of fabricated

integrated circuits (ICs). IC design and manufacturing may involve multiple fabricators and circuits have to run through multiple stages until a final product reaches its customer. In each individual production stage, there is potential for malicious manipulation of an IC. This threat is recognized in the US not only by intelligence service agents [353], but also in reports of government institutions [354]. It has been the subject of scientific discussion and investigation for several [355]. For instance, rumors exist that in 2007, a Syrian radar failed to warn of an incoming air strike due to a backdoor in the system's chips [355]. Similarly, a German missile system located at the Turkish-Syrian border may have carried out "unexplained commands" in 2015, with rumors suggesting that the system had been hacked and tampered hardware might have been used as an entry point [356]. More concrete documentation of attacks based on HTs are difficult to find, possibly due to concerns regarding the impact on security, economy, and society. Regardless of whether these rumors are true or not, there exists scientific evidence that cybersecurity attacks based on vulnerable hardware are possible [357]. Thus, establishing a trustworthy supply chain for information technology equipment is of interest for government institutions [358] as well as researchers, see [359] or [360].

Accounting for the manifold opportunities for threats, it becomes very hard to test if a downstream provider has installed an undesired functionality or if they are fully trustworthy. One of the most severe and threatening attacks to an IC is the integration of a hardware Trojan (abbreviated as HT or simply *Trojan* for short), a malicious modification to FPGAs, application-specific integrated circuits (ASICs), microprocessors or IoT devices [361], [362]. Such modifications can change the functionality of the hardware, e.g. downgrade its performance or provide a backdoor through which sensitive information can be leaked. A valuable survey analyzing the threat posed by hardware Trojans is provided by [363]. To give an example, the practical feasibility of an S-Box substitution attack for AES on FPGA designs has already been demonstrated in [364]. More recently, in [365] a tampering attack on AES ICs is presented that is designed to recover the secret key and thus fully undermine security of the encryption module. The described attack makes use of an HT that consumes (plaintext) input signals.

The motivation for our work lies in providing an efficient and effective method for the location of "small" HTs that are integrated in the IC at the manufacturing stage, i.e. the malicious components are added after the design phase and are not represented in the netlist or register transfer level. The approach shall be non-invasive, treat the circuit under test as a black box and allow for automation. Further, we want to investigate and demonstrate the suitability of combinatorial testing methods to hardware testing and to advance HT location. We believe that particularly the covering and separating properties of detecting arrays (considered in combinatorial testing) will positively influence the future development of HT location techniques.

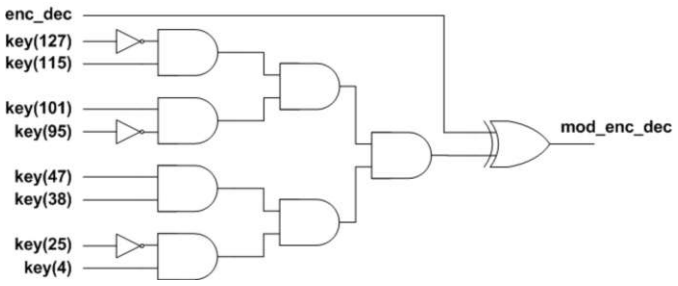


Figure 5.1: An example design of an HT of *length* eight, e.g., for an AES module, that consumes the gates in positions `4-25-38-47-95-101-115-127` corresponding to the key or plaintext input and that is activated when the signals 10110110 occur in these positions.

Hardware Trojans: Components, Types and Counter Measures

While there are implementations of HTs that do not require the addition of any gates, see e.g. [366], in this section we consider HTs that are realized as additional logic circuits. Such HTs generally consist of two parts: the *trigger* and the *payload*. The trigger circuit is always active. Once it recognizes the activating input, it activates the payload circuit that executes the malicious function of the HT, i.e. the HT is *triggered*. We distinguish HTs according to their trigger condition into analog and digital. The latter can be further split into combinational and sequential HTs according to the type of their circuit logic. See [367] or [368] for a comprehensive HT taxonomy. An example of a combinational HT circuit is depicted in Fig. 5.1. This HT is comprised by a trigger circuit consisting of seven AND-gates and three NOT-gates, while the payload circuit consists of only one XOR-gate that changes the encryption-decryption mode of the underlying AES module. The trigger circuit consumes the gates that process the input bits of the key or plaintext at positions `4-25-38-47-95-101-115-127` and activates the payload circuit if the signals 1, 0, 1, 1, 0, 1, 1 and 0 occur in the respective bits. We call the position of the consumed input bits together with their respective values the *trigger pattern* of the HT and the number of consumed input gates the *length* of the HT. Thus, in the following we often identify HTs with their trigger patterns and do not distinguish between them, e.g. we interchangeably use expressions like *length of the trigger pattern* and *length of the HT*.

Researchers have explored various approaches for HT detection. The acquired techniques can be classified as *destructive* methods, which permanently destroy the IC, and *non-destructive*. Non-destructive methods can be further divided into *invasive* techniques, where the layout of the circuit is modified (e.g. runtime methods) and *non-invasive* techniques where the circuit design is unaffected (e.g. logic testing or side channel analysis methods). More details are presented in surveys of these topics, such as [369] or [370]. Additionally, we want to mention the recent survey [371] on physical and logic testing techniques for HT detection as well as the work given in [372], which reviews HT threats and existing detection and prevention methods from a system-on-a-chip life cycle perspective with a focus on the advancement of machine learning in these domains.

Threat Model

We assume that the malicious modification of the design happens via an untrusted electronic design automation (EDA) tool, or in the manufacturing phase, by an untrusted employee at the foundry. Building upon the works of [269] and [343], we consider the same concrete threat model for our case study: We consider an implementation of the AES cryptographic algorithm, with a key length of 128 bits, in the form of hardware IP cores where an attacker integrated an HT triggered by an ℓ bit pattern in the plaintext or key input. The attacker can control the plaintext or the key input and can observe the ciphertext output. We further assume that the attacker combines only a few input signals for the activation, e.g. using a combinational logic relying on AND-gates and NOT-gates for the trigger circuitry, in order to remain undetected. For the payload circuit we assume that the attacker changes the encryption mode of the circuit, switching it from encryption to decryption and vice versa. Under these assumptions, we can recognize misbehavior of the circuit by comparing its output with the output of a *golden chip*, which can either be a trusted hardware or software implementation of the AES algorithm.

Note that these assumptions are made primarily for the sake of clarity and simplicity of the experimental setup. Our proposed testing method does not depend on *how* the HT manipulates the logic of the hardware, the essential precondition needed for our methodology to work is rather the existence of a testing oracle, i.e. a way to recognize *that* the HT is activated.

According to the comprehensive list of threat models presented in [367], we can categorize the described threat model as model B (untrusted foundry) or model C (untrusted EDA tool). In this sense, we can also categorize the HT design considered in this work according to the hardware Trojan taxonomy presented in [367], which is also underlying the benchmark library of [373] and is used in several scientific works (in slightly adapted versions), as in [371]. Adopting this terminology, we consider HTs that are

- inserted in the fabrication phase,
- on the gate-level,
- activated by a combinational pattern in the user input,
- located at the input, and
- changing the functionality of the circuit.

Regarding the physical characteristics of such HTs, we want to point out that if the targeted circuit is an ASIC, the layout will change in many cases. However, it does not change in case the targeted circuit is an FPGA and the integration of the malicious logic can be realized with available space in already used look-up tables, in which case the HT is also not visible in the netlist. Further, the size of a HT is a relevant physical characteristic, as a small and compact design is better from an attacker's point of view, warding off detection through e.g. optical inspection.

Contribution to Logic Testing for Hardware Trojan Location

Having provided some context for hardware testing, we can consider the following work as a contribution to logic testing for hardware Trojan location. It can be considered as a novel logic testing methodology based on CT-FLA methods that can excite and *locate* hardware Trojans that are triggered by certain combinational ℓ bit patterns in the primary input using combinatorial test sets. We instantiate this method by applying it to tampered AES cryptographic circuits. Being non-invasive in nature, our methodology relies solely on the results of the executed test vectors and their combinatorial properties. Further, our method does not rely on a physical realization of a golden chip, i.e. does not require a fully trusted *hardware* realization of the circuit under test. The set of test vectors is optimized aiming for as few vectors as possible while allowing for an efficient method for trigger pattern identification. The proposed work is in line with those presented in [269] and [343], considerably extending their results from *mere detection of the presence of a combinational HT to the precise identification of the consumed input gates* and the respective values that trigger a potential HT – hereafter referred to as *HT location* or HT trigger pattern location.

The devised methodology relies on results coming from CT-FLA. We revisit these results and show how they can be applied in the domain of hardware Trojan detection and location. Further, we devise novel algorithms in order to describe the developed HT location method. An extensive experimental evaluation demonstrates the functionality of the theoretical results applied to an AES module. Although exemplified by means of this application, due to its algorithmic nature, the proposed method can also be applied for HT location in more general setups. In the experiments, we perform HT location for circuits that implement the AES symmetric-key encryption algorithm in ECB mode for 128 bit key length that have been maliciously modified with HTs of length up to eight. In these experiments, we realize the concept of a golden chip in the form of a software implementation of the aforementioned AES algorithm. We also compare our HT location method against a random approach, showcasing the completeness and efficiency of our technique.

Our testing methodology does not rely on knowledge of the internals of the circuit under test and can be considered as a black box testing approach. We firmly believe that this is a strong advantage, as under realistic circumstances, we cannot assume to have knowledge of the internals of the hardware design where the HT is already inserted.

Thus, the main contributions to hardware testing are as follows:

- To the best of my knowledge, we introduce the first “pure” logic testing method for HT trigger pattern identification, i.e. we present a non-invasive logic testing method that relies solely on the results of the executed test vectors and their combinatorial properties to identify combinational patterns in the input that trigger HTs.
- The proposed method treats the circuit under test as a black-box and is independent from the gate-level netlist or any side-channel analysis.

- It does not require a hardware golden chip.
- We map concepts of combinatorial testing to HT testing and realize CT-FLA methods for logic testing.
- We describe the developed HT location method by means of novel algorithms which allow for fast HT trigger identification.
- We conduct an extensive experimental evaluation, locating HTs of length up to eight in FPGAs realizing the AES symmetric-key encryption algorithm in ECB mode for 128 bit key length.
- We further compare our method against a random approach that highlights the completeness and efficiency of our technique.

5.2.2 Related Work

Since we will use combinatorial testing for logic testing, and apply it for HT location, we present related work pertaining to logic testing methods as well as to HT location approaches.

Logic Testing: Test Set Generation

Logic testing approaches generally rely on the execution of test vectors while observing the responses of the circuit under test. Any deviation from the expected result (which can be provided, amongst others, by a golden chip or a simulation) reveals the presence of an HT. Logic testing is thus primarily suited for the detection of HTs that modify the IC's functionality, but it can also be used to enhance side channel analysis, as it was done in [374], [375], [376] and [377]. Here, the goal of logic testing methods is to activate potential HTs with a reasonable number of test vectors. For a survey dedicated to logic testing methods as a countermeasure to HT insertion, see also [378].

Generally, an attacker will try to design an HT to be stealthy under "normal conditions" in order to make it hard to detect. The trigger condition of an HT can thus be assumed to be a rare signal in the IC. Exhaustive testing, however, is usually infeasible due to the number of combinations of inputs and internal states being not tractable. The objective of testing is thus to activate potential HTs within a reasonable test time, aiming for a small number of test vectors, see [379] and [380]. There exist different strategies for test vector generation in order to trigger HTs, where some methods connect the rarity of HT activation to gates with rare values and design vectors in order to activate these signals in the IC, see e.g. [381]. Other methods assume that an attacker has no access to internal gates of the circuit and use CAs to cover all possible trigger patterns up to a certain length in the primary input of the IC, see e.g. [269] and [343].

In [381] a random sampling approach (called MERO) for HT detection is presented, which is based on multiple excitation of low-probability conditions at the internal nodes

of a circuit. MERO works by first identifying *rare nodes* with their associated *rare values*, followed by statistical sampling and execution of test vectors until all rare nodes have been triggered a certain number of times (similar to N-detect tests). Experimental evaluation shows that MERO improves over a random approach by achieving comparable HT detection capabilities while reducing the number of test vectors by 85%.

The work in [382] presents a test generation method based on a genetic algorithm for logic testing of circuits. The key point of this work is to define the fitness function that guides the genetic algorithm based on switching probabilities, controllability and observability parameters. A test vector that activates more rare nodes gets a higher score from the fitness function. A genetic algorithm is used to optimize an initially random set of test vectors until 95% of rare nodes are activated. The authors argue that not covering all of the rare nodes is the main reason why the test generation is faster than MERO [381]. The generated test vectors achieve very competitive results, but ultimately cannot compete with MERO in terms of trigger coverage.

Contrary to these approaches, logic testing based on combinatorial testing treats the circuit under test as a black box. Related works in that regard are [269], [343] and [344], which have already been discussed in the introduction to this chapter, see Section 5.1.

In addition to the "pure" logic testing methods mentioned above, we want to highlight the hybrid approach to HT detection presented in [376], which proposes a side-channel-aware test generation paradigm. The authors introduce the MERS (Multiple Excitation of Rare Switching) algorithm - an evolution of the MERO approach [381] - for test set generation, which takes as input a list of previously identified rare nodes and a set of random vectors. The set of vectors is modified until each rare node is switched (i.e. changes from its non-rare to its rare value) at least a given required number of times. The generated test vectors are then reordered with the goal of minimizing the total switching in the circuit while maintaining or improving the switching in the rare nodes. The aim of this optimization is to improve the side-channel sensitivity of the approach. Two methods for test reordering are proposed, the first is a heuristic based on minimizing the hamming distance of consecutive test vectors, the second is simulation based and reorders the test set based on information of the switching activity in the circuit obtained from iterated simulations.

The logic testing methods reviewed above share one communality, which is the lack of means for HT location. To the best of my knowledge, most existing logic testing approaches do not offer the generation of test sets that are capable of HT trigger identification at a post silicon stage.

Most recently, another method combining power-based side-channel analysis with logic testing allows to fully isolate Trojan signals in some cases [377]. In this work, a three-phase method based on adaptive logic testing is proposed. In the first phase, an N-detect test set for transition delay fault testing is deployed with the goal of exciting any signal from the Trojan circuitry to produce an initial suspicious power signal. The second phase aims to magnify the suspicious signal by modifying test vectors based on a heuristic that

changes small groups of input bits and evaluates the newly generated test vector for its relative power difference value. The process returns the test vector with the highest relative power difference found during the process. The third phase uses adaptive test vector superposition, aiming for test vectors that have a large overlap and cancel out their common effects, in order to detect the presence of a Trojan even under extreme process variation domains. In two documented cases, it was possible to fully isolate the Trojan signal. This method achieves Trojan signal magnification increase by orders of magnitude when compared to ATPG, a significant increase when compared to adaptive ATPG, and is capable of fully isolating the Trojan signals for two instances. The authors of [377] require to apply many modifications to the test vectors in order to find a pair suitable for superposition. To a certain extent, the combinatorial methods for HT location presented in this work can be considered a superposition method that incorporates information of multiple test vectors.

Location of Hardware Trojans

The work in [383] presents the *COTD* technique for Trojan detection and identification that makes it possible to fully identify an inserted HT by isolating its trigger and payload circuit. COTD takes the gate-level netlist as input and combines controllability and observability analysis with unsupervised machine learning to distinguish Trojan gates from genuine gates. A clear advantage of this approach is its independence from a *golden chip* and any test pattern application for partial of full Trojan activation. However, the knowledge of the complete gate-level netlist is required as input, with the Trojan logic inserted, which may not always be available. Similarly, the authors of [384] propose a reference-free scheme for HT trigger location by identifying their rare trigger signals based on the gate-level netlist. They make use of the hypothesis that nodes with rare values (low probability signals) are nodes with an imbalance in 0/1-controllability, which can be calculated by the Synopsys EDA tool TetraMAX. Based on the differences of the 0/1-controllability values 3-means clustering is applied in order to obtain lists of suspicious signals, which can then be refined by a dynamic probability analysis. The authors report that their method can achieve zero false negatives while improving the number of false positives in numerous benchmarks and being very competitive otherwise. Again, this method requires the gate-level netlist with the inserted HT.

The authors of [385] use social network analysis of register transfer level designs for HT trigger and payload location in the design. They assume that the circuit under test, with the potentially inserted HT, is represented as an edge-labelled directed acyclic graph, for which they compute several attributes, such as different centrality measures for vertices or the density of subgraphs. Based on these attributes some of the vertices are marked as possible HT trigger or HT payload nodes, not requiring any simulations or side-channel analysis.

The methods outlined above, in one way or another, assume the knowledge of the Trojan infected design of the circuit, which might not be available, especially if the malicious modification of the circuit happens at the foundry in the silicon stage.

The work in [386] could serve as an important initial step to our work, as it presents a method to identify circuit cites where a potential HT trigger may be inserted. It proposes to first identify nodes with a low controllability based on probability analysis and then to consider those nodes where an insertion of additional HT gates would not result in a significant delay, by considering the nodes with a positive slack. Additionally, the nodes' physical placement in the circuit is taken into account, as a potential HT trigger needs some space in the layout in order to be integrated. Finally, the work proposes to consider subsets of nodes that fulfill the above three criteria, taking into account their physical closeness in the circuit's layout. The authors further propose to generate test vectors designed to trigger the potential HT. However, this step is not carried out, leaving open the problem of (partial) HT triggering and location. Nevertheless, this work treats an important preprocessing step, when one is interested in physically locating an HT. We mention it here, because we believe that such methods can benefit, when supplemented with the hereafter proposed logic testing based on CT-FLA.

5.2.3 Linkage Between Hardware Trojan Location and Combinatorial Testing

In the following we set out how to apply non-adaptive CT-FLA for locating HTs. Next, we bridge the concepts used in HT detection and location with those of combinatorial testing.

We denote with k the total number of the input signals available to the attacker and with ℓ the *length of the HT*, i.e. the number of input signals of the Trojan's trigger circuit. In combinatorial testing terminology the 128 bits of the plaintext yield an IPM consisting of 128 binary parameters. As we assume no input dependencies (i.e., the value of one bit does not depend on the value of any of the other input bits), there are no constraints in our model. According to the threat model, the attacker is using some ℓ bit pattern in the plaintext to activate the HT. In this setting, the natural choice is to map the HT's trigger pattern of length ℓ to the concept of the failure inducing t-way interaction, the FIT we are interested in locating in CT-FLA. In this setting, the natural choice is to map the concept of the length of trigger patterns of HTs to the concept of the strength of failure inducing interactions. *Hence, for the remainder of this section, we may identify trigger patterns of length ℓ and failure inducing t-way interactions with each other and may use the terms interchangeably.* In order to locate the HT's trigger pattern we thus have to construct a binary detecting array which has 128 columns and is capable of locating a single ℓ-way interaction, i.e. a $(1, \ell)$-detecting array or a $(1, \bar{\ell})$-detecting array for 128 binary columns. Since ℓ is unknown to the tester, the key issue is the selection of an appropriate strength t to guide the combinatorial test set generation, allowing us to capture the length ℓ of the integrated HT (note that $t \neq \ell$ in general, as we shall see in the next sections). While this problem is not unknown to general combinatorial testing applications, there is currently no golden rule for selecting the correct value of t. The selection of t is rather a trade-off between detection capabilities and availability of resources. In some cases, there exists empirical evidence that can guide the selection of

Hardware Testing	Combinatorial Testing
Plaintext/key length	Number of input parameters k
Plaintext input/key	Test
Test vector	Row of a covering array
Pattern	t-way interaction
Trigger pattern	Failure inducing t-way interaction (FIT)
(Comparison to) golden chip	(Call of the) testing oracle
Correct ciphertext	Passing test
Incorrect ciphertext	Failing test

Table 5.1: A summary of the mappings between equivalent concepts and notions in HT location and combinatorial testing.

the interaction strength for the combinatorial test set generation. For example, for some applications within the domain of software testing, security testing as well as testing of medical devices there is empirical evidence that suggests that an interaction strength of $t = 4$ to $t = 6$, depending on the use case, is sufficient for testing, in the sense that all previously documented bugs can be triggered by interactions of these strengths, see [298] and [387]. We will demonstrate, however, that based on the arguments given in Remark 5.2.1, we are able to locate HTs of length ℓ with $(1, t)$-detecting arrays, as long as $\ell \leq t$.

Justified by the identification of trigger patterns with failure inducing t-way interactions, we use the same terminology for t-way interactions also for patterns in plaintext inputs. For example, if a plaintext consisting of 128 binary bits contains a certain binary sub-pattern, we also say that the pattern is *covered* by the test vector, just as we do for t-way interactions and binary vectors of length k in general.

Table 5.1 summarizes how concepts and terms from CT can be translated to the domain of hardware testing (with a focus on cryptographics Trojans) and vice versa.

In the following we will make use of the result of Colbourn and McCleary [62, Theorem 8.5] given also in statement (1.1), and make apparent the following consequence.

Remark 5.2.1 *Since every* $\mathsf{CA}(N; t+d, k, v)$ *is also a* $\mathsf{CA}(N; (t-1)+d, k, v)$*, from [62, Theorem 8.5] we immediately get that for* $d < v$ *every* $\mathsf{CA}(N; t+d, k, v)$ *is a* (d, s)*-detecting array for every* $s \leq t$*, and hence for* $d = 1$ *we get further that every* $\mathsf{CA}(N; t+1, k, v)$ *is a* $(1, \bar{t})$*-detecting array. In other words, this shows that every* $\mathsf{CA}(N; t+1, k, v)$ *can be used for the location of one failure inducing* $\leq t$*-way interaction.*

5.2.4 Location Algorithms

Our objective is to first design a testing method that can excite and *locate* an HT using an optimized test set, and second to manifest an efficient procedure to retrieve the trigger pattern from an annotated set of test vectors. In the following we assume that the AES keys or plaintext vectors applied for testing are given as the rows of an array A, and the result of the testing against the golden chip is given as an *oracle assignment*, that is a column o of pass/fail assignments to the test vectors. In other words, we assume we have

an annotated test set (A, o). A naive approach would be to generate and test all possible input vectors and to use simple enumeration methods to locate the HT's trigger pattern. However, this would be practically infeasible due to the size of the input space being exponential in the number of input bits.

Location via full Enumeration

The first approach towards locating trigger patterns can be described as follows. Whenever the HT is triggered we observe a discrepancy in the ciphertext output by comparing to the ciphertext output of the golden chip. On the contrary, for any plaintext that represents a passing test (the ciphertext output is identical with that of the golden chip), we know that the trigger pattern cannot be covered in the plaintext and thus that each pattern in this plaintext cannot be the trigger pattern of the HT (i.e. not the FIT in terms of combinatorial testing).

Therefore, a straightforward approach aiming for identification of the trigger pattern of length ℓ is to iterate over all passing tests and all patterns of length ℓ covered by the individual passing tests to mark them as non-trigger patterns. Algorithm 14 represents a pseudocode for this procedure, which we have already mentioned in the introduction, see Remark 1.4.1, as a location procedure for detecting arrays that was originally mentioned in [62], directly after the introduction of detecting arrays.

If a trigger pattern exists, the set $\mathcal{T}$ returned by Algorithm 14 contains the trigger pattern or a set of potential trigger patterns, depending on the quality of the used test set A. For example, if we use a test set that was randomly generated, we generally cannot expect the returned set $\mathcal{T}$ by Algorithm 14 to contain exactly the trigger pattern, but rather a set of potential trigger patterns. This is because there is no guarantee that each non-trigger pattern is covered in a passing test. In contrast, if the test set is deduced from a $(1, \ell)$-detecting array, in case of the presence of an HT of length ℓ, we have the guarantee that there is exactly one remaining pattern in $\mathcal{T}$ due to the combinatorial properties of detecting arrays: each non-trigger pattern must appear in at least one passing test. Thus, we can formulate the following statement:

Algorithm 14 FullEnumerationLocation(A, o, ℓ)

```
1: INPUT: Test set A, oracle assignment o, length ℓ
2: P ← passing tests(A, o)          ▷ Extract passing tests from annotated test set
3: T ← set of all patterns of length ℓ
4: for p ∈ P do
5:     for all patterns τ covered by p do
6:         T ← T \ {τ}              ▷ Mark τ as non-trigger pattern
7:     end for
8: end for
9: return T                         ▷ Set of unmarked patterns
```

Proposition 5.2.2 *If the test set A is deduced from a $(1,\ell)$-detecting array and the oracle column o is the pass/fail assignment retrieved from testing a modified AES module with an integrated HT of length ℓ, then Algorithm 14 returns the trigger pattern of the HT.*

Example 5.2.3 *We illustrate the location method set out by Algorithm 14 by means of the following example. We use the $(1,2)$-detecting array given in Table 1.1 as an example test set A. As this array has 11 columns, it is only suitable for testing an SUT that consists of 11 binary parameters; however, the same concept applies to testing the AES module presented later in this section. We assume that the occurrence of the values 01 in the positions $1-2$ trigger a failure in the SUT; in other words, we assume that there is one FIT, which is $\{(1,0),(2,1)\}$ denoted as 2-way interaction. Note that the positions of the trigger pattern do not have to be adjacent for our method to work. We assume adjacent positions solely for the sake of better readability. When we use the array given in Table 1.1 for testing, the oracle returns the pass/fail assignment $o = (0,0,0,1,0,0,0,0,1,1,0,0)^T$, where 1 denotes a failing test and 0 denotes a passing test. Henceforth we assume we are only given the annotated test set (A,o) and that we have no information about the trigger pattern, except that its length is two.*

From Algorithm 14, we first extract the passing tests $\mathcal{P}$ from (A,o), which yields the following array:

$$\mathcal{P} = \begin{pmatrix} 1&0&1&0&0&1&0&1&1&0&1\\ 1&1&0&0&1&0&1&1&0&0&1\\ 1&0&0&0&1&1&1&0&1&1&0\\ 1&1&1&1&0&1&1&0&0&0&0\\ 1&0&1&1&1&0&0&0&0&1&1\\ 1&1&0&1&0&0&0&1&1&1&0\\ 0&0&1&1&1&0&1&1&1&0&0\\ 0&0&0&1&0&1&1&1&0&1&1\\ 0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix} \tag{5.1}$$

The set $\mathcal{T}$ is initialised as the set of all $2^2\binom{11}{2}$ binary patterns of length 2, i.e. binary 2-way interactions for $k = 11$:

$$\begin{aligned}\mathcal{T} = \{&\{(1,0),(2,0)\},\{\{(\mathbf{1},\mathbf{0}),(\mathbf{2},\mathbf{1})\}\},\{(1,1),(2,0)\},\{(1,1),(2,1)\},\\ &\{(1,0),(3,0)\},\{(1,0),(3,1)\},\{(1,1),(3,0)\},\{(1,1),(3,1)\},\dots,\\ &\{(10,0),(11,0)\},\{(10,0),(11,1)\},\{(10,1),(11,0)\},\{(10,1),(11,1)\}\}.\end{aligned}$$

While this is somewhat tedious to manually verify, the test vectors defined by the rows of $\mathcal{P}$ cover all patterns except 01 in the positions $1-2$, which is the only remaining pattern in the set $\mathcal{T}$ at the end of Algorithm 14. This means that we have successfully recovered the trigger pattern from the annotated test set (A,o) and $\ell = 2$.

Example 5.2.4 *To compare with HT trigger pattern location when the test set is* randomly generated*, we consider the same preconditions as in Example 5.2.3, but now assume we have tested the SUT with the test set A_{rand} that was randomly generated:*

$$A_{rand} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{5.2}$$

It is of the same size as the $(1,2)$-detecting array used in Example 5.2.3, but each entry has been selected uniformly at random from the set $\{0,1\}$. The oracle assignment to the test set A_{rand} is $o_{rand} = (0,0,0,1,0,0,0,1,0,0,0,0)^T$. Following Algorithm 1 on the input $(A_{rand}, o_{rand}, 2)$, we first extract the passing tests $\mathcal{P}_{rand}$ from (A_{rand}, o_{rand}), which yields the array

$$\mathcal{P}_{rand} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Again, we can iterate over all patterns of length two that are covered by the rows of $\mathcal{P}_{rand}$ and mark them as non-trigger patterns, respectively remove them from $\mathcal{T}$. Doing so, we will find a total of 11 *patterns of length two that are not covered by the rows of $\mathcal{P}$; for example, the patterns* 01 *in positions* $1-2$*,* 11 *in positions* $2-4$ *and* 01 *in positions* $10-11$ *to name a view. In this case, we are not able to locate, i.e. uniquely identify, the trigger pattern.*

Location via Enumeration

A second, slightly different algorithm, also based on the enumeration of patterns, can be described as follows.

This algorithm first splits the executed test vectors into two sets: the failing test vectors $\mathcal{F}$ and the passing test vectors $\mathcal{P}$. As each failing test must cover the trigger pattern, we can select a failing test $\bar{f} \in \mathcal{F}$ at random and we are guaranteed that it covers the trigger pattern. We can now iterate over all patterns $\mathcal{T}_{\bar{f}}$ covered by the failing test $\bar{f}$, check which patterns appear in a passing test and thus mark it as non-trigger pattern. Finally, $\mathcal{T}_{\bar{f}}$ is reduced to a set of (potential) trigger patterns. Algorithm 15 gives a pseudocode of such an algorithmic procedure.

Let us again consider the case where the test set is deduced from a $(1, \ell)$-detecting array for the location of an HT trigger pattern of length ℓ. Again, as for Algorithm 14, we are guaranteed that the returned set $\mathcal{T}_{\bar{f}}$ contains exactly the trigger pattern. This guarantee comes from the property of the $(1, \ell)$-detecting array that each non-trigger pattern must appear in at least one passing test. We can thus formulate the following statement:

Proposition 5.2.5 *If the test set A is deduced from a $(1, \ell)$-detecting array and the oracle column o is the pass/fail assignment retrieved from testing a modified AES module with an integrated HT of length ℓ, then Algorithm 15 returns the trigger pattern of the HT.*

Remark 5.2.6 *While Algorithm 14 has a runtime in $\Theta(\binom{k}{\ell}|\mathcal{P}|)$, as it iterates over all patterns in all passing tests, Algorithm 15 has a runtime in $O(\binom{k}{\ell}|\mathcal{P}|)$, with the potential of a reduced average runtime, since we expect to find a $\tau \in T_{\bar{f}}$ that is a non-trigger pattern covered by a test vector in $\mathcal{P}$ in $|\mathcal{P}|/2$ steps.*

Also, for randomly generated test vectors, we expect Algorithm 15 to perform better than Algorithm 14 when applied to the same input (A, o, ℓ), in the sense that the result is more precise. Since we initialize the set of potential trigger patterns only with those of length ℓ that are covered by the failing test $\bar{f}$ instead of all possible trigger patterns of length ℓ, the base set from which we remove the non-trigger patterns is smaller. As both algorithms remove all patterns covered by some passing test, the set returned by Algorithm 15 must be a subset of the one returned by Algorithm 14. In other words, let $patterns_\ell(\mathcal{P}) := \{\tau | \exists p \in \mathcal{P} : \ p \text{ covers } \tau \text{ and the length of } \tau \text{ is } \ell\}$, then:

$$\mathcal{T}_{\bar{f}} \subseteq \mathcal{T} \Rightarrow T_{\bar{f}} \setminus patterns_\ell(\mathcal{P}) \subseteq T \setminus patterns_\ell(\mathcal{P}).$$

Example 5.2.3 (continuing from p. 247) *We locate the failure inducing pattern of length $\ell = 2$, this time using Algorithm 15. We extract the passing tests $\mathcal{P}$ from (A, o) (see Equation (5.1)), and randomly select one of the failing tests $\bar{f} \in \mathcal{F}$, say*

$$\bar{f} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

Algorithm 15 EnumerationLocation(A, o, ℓ)

```
INPUT: Test set A, oracle assignment o, length ℓ
P ← passing tests(A, o)                    ▷ Extract passing tests from annotated test set
F ← failing tests(A, o)                    ▷ Extract failing tests from annotated test set
Select f̄ ∈ F randomly
T_f̄ ← set of all patterns of length ℓ covered by f̄
for τ ∈ T_f̄ do
    for p ∈ P do
        if p covers τ then
            T ← T \ {τ}                    ▷ mark τ as non-FIT
            go to next τ ∈ T_f̄
        end if
    end for
end for
return T_f̄                                ▷ set of unmarked t-way interactions
```

The set $\mathcal{T}_{\bar{f}}$ of all patterns of length two covered by $\bar{f}$ is then

$$\begin{aligned}\mathcal{T}_{\bar{f}} = \{&\{(\mathbf{1},\mathbf{0}),(\mathbf{2},\mathbf{1})\},\{(1,0),(3,1)\},\{(1,0),(4,0)\},\ldots,\{(1,0),(11,0)\},\\ &\{(2,1),(3,1)\},\{(2,1),(4,0)\},\{(2,1),(5,1)\},\ldots,\{(2,1),(11,0)\},\ldots\\ &\{(8,1),(11,0)\},\{(9,0),(10,1)\},\{(9,0),(11,0)\},\{(10,1),(11,0)\}.\}.\end{aligned}$$

It is once more a tedious to verify manually, but we find every pattern in $\mathcal{T}_{\bar{f}}$, except for $\{(\mathbf{1},\mathbf{0}),(\mathbf{2},\mathbf{1})\}$ covered by some of the passing tests in $\mathcal{P}$. Again, the failure inducing pattern 01 in positions $1-2$ is successfully recovered from the annotated test set (A, o) and $\ell = 2$.

Example 5.2.4 (continuing from p. 248) *To compare against random testing, we follow the same procedure where the test set is A_{rand}, as given in Equation (5.2). We randomly select one of the failing tests, say $\bar{f}_{rand} = \begin{pmatrix}0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0\end{pmatrix}$. Then, from all patterns of length two $\mathcal{T}_{\bar{f}_{rand}}$ covered by $\bar{f}_{rand}$, we remove those patterns that are covered by the rows of $\mathcal{P}_{rand}$. This yields the returned set*

$$\mathcal{T}_{\bar{f}_{rand}} = \{\{(1,0),(2,1)\},\{(2,1),(3,1)\},\{(3,1),(10,0)\}\},$$

which contains 3 patterns of length two. This is a reduction compared to the 11 patterns returned from Algorithm 14 applied to the same test set, but at the same time it does not locate the trigger pattern precisely.

Fast Location via Identification of Shared Patterns

The third algorithm we introduce allows for efficient HT location when the test set is deduced from a detecting array. We present the procedure in Algorithm 16 using HT

location terminology (see Table 5.1). The underlying idea of this location algorithm is as follows: The trigger pattern must be covered in all failing test sets, hence all failing test vectors must be identical in the positions of the trigger pattern. The suggested location procedure thus consists of finding all positions where all failing tests agree, see Algorithm 16.

Algorithm 16 FastLocation(A,o)

```
1: INPUT: Test set A, oracle assignment o
2: F ← failing tests(A, o)          ▷ Extract failing tests from annotated test set
3: τ ← ∅                            ▷ Initialize HT trigger pattern τ as empty
4: Select f̄ ∈ F randomly
5: for i ∈ {1, ..., 128} do
6:     if all f ∈ F agree in position i then
7:         τ = τ ∪ {(i, f̄(i))}
8:     end if
9: end for
10: return τ
```

When a combinatorial test set with sufficiently strong properties is used, we also get guarantees for the result such an Algorithm.

Theorem 5.2.7 (Fast Location for single t-FITs) *Let A be a $\mathsf{CA}(N; t+1, k, v)$ ($1 \leq t \leq k$ and $2 \leq v$) and τ be an arbitrary t-way interaction, representing a failure inducing t-way interaction (t-FIT). Let further $\mathcal{F}$ be the subset of rows of A that cover τ, then τ is uniquely characterized by those positions where all rows of $\mathcal{F}$ agree.*

Proof: Suppose $\tau = \{(p_1, u_1), \ldots, (p_t, u_t)\}$ is the t-FIT, i.e. the failure inducing t-way interaction. From the fact given [62, Th. 8.5], re-stated in (1.1), we know that A is a $(1, t)$-detecting array, i.e. we know already that τ is the unique t-way interaction that is covered only by the rows in $\mathcal{F}$. It merely remains to show that we can recover τ based on the information in $\mathcal{F}$.

Assume j is a position that is not part of τ, i.e. $j \in \{1, \ldots, k\} \setminus \{(p_1, \ldots, p_t)\}$. Then there exist two row vectors, say $f, \bar{f} \in \mathcal{F}$ that disagree in position j: consider the two $(t+1)$-way interactions $\tau \cup \{(j, 0)\}$ and $\tau \cup \{(j, 1)\}$. Since A is a CA of strength $(t+1)$, there must be two rows f and $\bar{f}$ in A that cover these $(t+1)$-way interactions. As they both contain position j, but disagree on the value in this position, they must be distinct $f \neq \bar{f}$. Further, since f and $\bar{f}$ cover τ, we also have $f, \bar{f} \in \mathcal{F}$.

Thus for each position that is not part of τ, we will find two rows in $\mathcal{F}$ that disagree on the value in this position, and we can recover τ from $\mathcal{F}$, by those positions where all rows of $\mathcal{F}$ agree on the value.

For HT location of the considered AES module, this means that in case the test set is derived from a detecting array with the desired properties (i.e. sufficient strength), we

can further conclude that for each position $i \in \{1, \ldots, 128\}$ that is not involved in the trigger pattern, there are at least two failing tests $f, f' \in \mathcal{F}$ that disagree in position i: $f_i \neq f'_i$. Thus, we can characterize the positions that are part of the trigger pattern exactly as those where all failing tests agree. Once the positions of the trigger pattern are determined, i.e. the gates that are consumed by the HT have been identified, we can simply obtain the values of the trigger pattern by looking them up in one of the failing test vectors. Algorithm 16 presents a pseudocode of this algorithmic procedure, and based on Theorem 5.2.7 we can formulate the following statement.

Proposition 5.2.8 *If the test set A is deduced from a $(1, \ell)$-detecting array that is a covering array of strength $(\ell + 1)$ and the oracle column o is the pass/fail assignment retrieved from testing a modified AES module with an integrated HT of length at most ℓ, then Algorithm 16 returns the trigger pattern of the HT.*

For a randomly generated test set A, we cannot expect that the τ returned by Algorithm 16 contains the trigger pattern. First, there is no guarantee that the trigger pattern is covered by one of the tests in A in the first place. Second, in case it is covered, the property that all failing tests agree exclusively on the trigger pattern is not guaranteed to be satisfied.

Example 5.2.3 (continuing from p. 247) *We locate the trigger pattern of length two, this time with Algorithm 16. We first extract all failing tests from (A, o):*

$$\mathcal{F} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

The idea of Algorithm 16 is to find the trigger pattern as the common pattern of length 2 of all failing tests. It is easy to see that the pattern 01 in positions $1-2$ is exactly this common pattern of length 2.

Example 5.2.4 (continuing from p. 248) *A third time, we want to compare against random testing. We follow the same algorithmic procedure (Algorithm 16) with the randomly generated test set A_{rand} given in Equation (5.2) and the pass/fail assignment given by $o_{rand} = (0,0,0,1,0,0,0,1,0,0,0,0)^T$. We extract the failing test cases $\mathcal{F}_{rand}$ from (A_{rand}, o_{rand}):*

$$\mathcal{F}_{rand} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The pattern τ returned by Algorithm 16 is 0111100 in positions $1-2-3-5-6-8-9$, or, written as a 7-way interaction, $\tau = \{(1,0),(2,1),(3,1),(5,1),(6,1),(8,0),(9,0)\}$. However, this 7-way interaction contains 21 different 2-way interactions: $\{(1,0),(2,1)\}, \{(1,0),(3,1)\}, \ldots, \{(2,1),(8,0)\}, \ldots, \{(6,1),(8,0)\}$ and $\{(6,1),(9,0)\}$. The failure inducing 2-way interaction is thus not uniquely identifiable.

Remark 5.2.9 *The time complexity of Algorithm 16, is greatly reduced compared to that of Algorithms 14 and 15, as it only iterates once over the k bits of the test vectors, searching for common entries. Algorithms 14 and 15 both rely on enumeration of length ℓ sub-patterns of test vectors, introducing a factor of $\binom{128}{\ell}$ to the runtime complexity in the case of 128 bit AES, or more general a factor of $\binom{k}{\ell}$ when there are k gates accessible to an attacker. The reduction in runtime becomes apparent when we consider the location of an HT of length $\ell = 8$; in this case, we would have to iterate over $\binom{128}{8}$ (more than 10^{12}) patterns to locate the HT via Algorithms 14 or 15. In contrast to this the runtime of Algorithm 16 is largely independent from the HTs length ℓ. The HTs length clearly indirectly influences the runtime of the algorithm, as an increased length ℓ requires a higher strength of the used detecting array, which may result in a larger number of failing tests in $\mathcal{F}$. We will present the results of an experimental evaluation later in Subsection 5.2.6, Table 5.9.*

5.2.5 Case Study

We demonstrate the efficiency of the proposed HT location based on combinatorial testing in a case study, applying it to an FPGA implementing the AES symmetric-key encryption that got tampered with a combinational HT. Even though we consider a concrete case, we highlight once more that the proposed method is explained by means - but not limited to - the described AES module. In the next paragraphs, we describe the used AES cryptographic module's, the HT variants that were integrated in the modules design, as well as the setup for the conducted experiments.

We consider the same scenario as in [269] and [343], where a tester receives a batch of fabricated AES modules. The testers suspect that (some of) the modules are contaminated with a combinational HT that consumes primary inputs. Their goal is to locate the HT, with the additional aim of reducing the test time per module while attaining a high confidence that the module is HT free.

For our experiments, we opted for the Verilog code of the AES implementation that is provided by the SAKURA-G board 2, which hardware architecture is discussed in detail in [388]. The AES module accepts as input a 128-bit key and a 128-bit plaintext (respectively ciphertext) and produces 128-bit ciphertext (plaintext) as output. The module implements the ECB mode of AES, which can be used as a building block for implementing other modes of AES, such as CBC or OFB, using additional logic for combining and reusing its output. The module can be controlled via a control signal to switch between the encryption and the decryption operation. *In our experiments, we consider the internals of the AES module as a black box.*

As mentioned in Subsection 5.2.1 the output of the AES module (Verilog code simulation) can be checked against the output of a trusted implementation of the algorithm (e.g., a software version from a trusted source), which serves as a testing oracle. If the two outputs differ, then the HT is assumed to be activated, i.e. its trigger pattern must be present in the input.

We used the ModelSim [389] tool with appropriate scripting (do files and shell scripts) in all of our experiments for automating the execution, collection, and comparison of the outputs. The approach can be easily extended to a hardware co-simulation using hardware co-simulation with Vivado [390]. The processing of the test vectors and the testing results was performed using a Matlab implementation of Algorithm 16. The presented time measurements are based on Matlab implementations of the described procedures and were conducted on a machine with an Intel i9-9900 CPU clocked at 3.60 GHz with 64GB of RAM.

5.2.6 Experiments

In this section we demonstrate the capabilities of the proposed HT location using CT-FLA techniques. To this end, we adopt the conceptualization of the experiments performed in [343], i.e. consider cases where AES modules have been contaminated with HTs of different lengths and different trigger patterns. Thereby, we shift the focus from mere HT detection as presented in previous works [269] and [343] to HT location, i.e. the exact identification of the trigger pattern via Algorithm 16, if not stated differently. Further, we analyze the capabilities of randomly generated test sets (respectively arrays), similar to the analogous process in [343], but now performing HT location instead of HT detection. Therefore, as part of our experiments, we consider the following test sets (which are provided online under [391]):

- CTℓ: refers to a combinatorial test set derived from a $(1, \bar{\ell})$-detecting array for 128 binary parameters, suited for HT location of length up to ℓ. The number of rows (i.e. test vectors) is reported in Table 5.2.
- $rand(CT\ell)$: refers to a randomly generated test set derived from a random array (each entry chosen uniformly at random from $\{0, 1\}$) that is of the same size as the $(1, \bar{\ell})$-detecting array underlying the CTℓ array.
- $rand_N(CT\ell)$: refers to a randomly generated test set derived from a random array with N rows (each entry chosen uniformly at random from $\{0, 1\}$). The chosen size N is identical to that of the smallest $(1, \bar{\ell})$-detecting array for 128 binary parameters currently known.

In our experiments we used combinatorial test sets CTℓ that allow for HT location based on Algorithm 16, i.e. we generated $(1, \ell)$-detecting arrays that are CAs of strength $\ell + 1$ for the location of HTs with trigger patterns of length up to ℓ. In comparison to other logic testing approaches these test sets demonstrate the efficiency of combinatorial testing in terms of generating small size test sets. Table 5.2 shows a comparison to other (state of the art) test sets for hardware testing for $k = 128$ input bits. The column headed by "ℓ" denotes the length of the HT for which the respective test set is designed; the column "Lesperance et al." reports the test set sizes given in [392]; "CWV" contains the analogous sizes given in [29]; "CTdetect" shows the sizes from [343]; and "CTlocate"

k	ℓ	Lesperance et al.	CWV	CTdetect	CTlocate (CTℓ)
128	2	2^7	129	11	54
128	3	-	256	37	135
128	4	2^{13}	8,256	112	346
128	5	-	16,256	252	5,921
128	6	-	349,504	720	29,830
128	7	-	682,752	2,462	103,691
128	8	2^{23}	11,009,376	17,544	595,979

Table 5.2: Comparison of sizes of test sets coming from CT methods (CTdetect & CTlocate) against other state of the art logic testing techniques for combinational HT detection.

reports the sizes of the test sets CTℓ for $\ell \in \{1, \ldots, 8\}$ used in the following experiments. We can see that the test sets used for HT location (CTlocate) are considerably larger than those for HT detection (CTdetect). However, these are still smaller by orders of magnitude compared to the test sets of [392] and [29] that provide full coverage of all length ℓ patterns and are designed merely for HT detection. The increased size of the test sets for HT location is caused due to additional structure and tests that are required in order to locate all HTs of a specific length. As the test sets can still be processed very fast in terms of test execution, we do believe that the capability of locating all HTs of length ℓ justifies the increased size.

Further note, that the test sets we used are not necessarily optimal, i.e. it is possible to construct combinatorial test sets with less test vectors. However, the number of test vectors in the combinatorial test set is not relevant for the location capabilities of our approach, as long as the necessary combinatorial properties, i.e. being a $(1, \overline{\ell})$-detecting array for 128 binary parameters, are guaranteed. This becomes more clear when we consider Algorithm 16 and the associated Proposition 5.2.8, which do not depend on the test set size, but only on its combinatorial properties.

In the following sections, we report and discuss the results of a set of experiments, where in each case the test sets are applied to a trusted implementation of the AES algorithm and to a contaminated AES module where an HT is triggered by a pattern unknown to the tester:

(A) In Subsection 5.2.6, we focus on locating Trojans of length ℓ using CTℓ arrays:

- We run eight combinatorial test sets against eight contaminated AES modules that differ in the length of the inserted HT.
- We run eight combinatorial test sets against eight different versions of contaminated AES modules. In each case, the HT is triggered by the signals 11111111, but the eight gates monitored by the HT differ.
- We run eight combinatorial test sets against eight different versions of contaminated AES modules. In each case, the HT monitors the same positions/input gates, but the values of the pattern differ, having a varying number of ones.

(B) In Subsection 5.2.6, we focus on analyzing Trojans of length ℓ using random arrays:

- We run eight random test sets against eight contaminated AES modules that differ in the length of the inserted HT.
- We measure the location capabilities of the random test sets for HTs of length $\ell = 1, 2, 3, 4$ when used in conjunction with Algorithm 16.

(C) In Subsection 5.2.6, we focus on fast pattern location with random arrays that have the same number of rows as the smallest $(1, \bar{\ell})$-detecting array for 128 binary parameters currently known:

- We measure the location capabilities of a second set of random test sets of reduced size for HTs of length $\ell = 1, 2, 3, 4$ when used in conjunction with Algorithm 16.

(D) In Subsection 5.2.6, we focus on slow pattern location properties of random arrays:

- We measure the location capabilities of the first set of random tests for HTs of length $\ell = 1, 2, 3, 4$ when used in conjunction with Algorithm 14.

Locating Length ℓ Trojans with CTℓ Arrays

In the first set of experiments, we demonstrate the HT location capabilities of the combinatorial test sets CTt for $t = 1, \ldots, 8$, in conjunction with Algorithm 16. The results of our experiments with HTs of different length are documented in Table 5.3. The columns contain the following values:

- "ℓ" shows the length of the inserted HT,
- "positions" shows the number of the respective gates that the HT trigger circuit is consuming,
- "pattern" shows the signals that need to appear at these gates to trigger the HT,
- "CTt" shows, for $t = 1, \ldots, 8$, how often the specific HT was triggered (# trig) by the test set CTt and whether the HT was located (loc) using Algorithm 16.

When we are able to locate the HT, i.e. precisely retrieve positions and pattern, we mark this with a ✓ in the corresponding column headed by "loc"; otherwise we denote it as ✗. We can see that all HTs of length up to ℓ can be located with the test set CTℓ, which is expected since CTℓ represents a $(1, \bar{\ell})$-detecting array. In some cases, we can also locate HTs of length ℓ with test sets CTt where $t \leq \ell$. While there is no theoretical guarantee for this capability, it is possible that some trigger patterns can be located using these concrete test sets in combination with Algorithm 16.

Table 5.3: The given patterns were used in the specified positions as HT trigger patterns. The remaining columns detail the trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 16 for HT location.

			CT1		CT2		CT3		CT4		CT5		CT6		CT7		CT8	
ℓ	pattern	positions	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
1	0	13	8	✓	28	✓	69	✓	175	✓	2968	✓	14955	✓	51964	✓	298460	✓
2	11	70-120	4	✗	13	✓	33	✓	82	✓	1 482	✓	7 455	✓	25 971	✓	143 654	✓
3	101	5-26-111	1	✗	7	✗	14	✓	45	✓	754	✓	3 823	✓	13 291	✓	72 937	✓
4	1100	0-30-110-127	0	✗	3	✗	5	✗	23	✓	407	✓	1 902	✓	6 546	✓	48 160	✓
5	01100	0-20-70-126-127	2	✗	2	✗	6	✗	10	✓	219	✓	1 045	✓	3 527	✓	17 112	✓
6	001111	1-21-79-100-101-123	1	✗	1	✗	3	✗	2	✗	115	✓	505	✓	1 735	✓	16 703	✓
7	1101010	1-21-50-82-101-111-123	0	✗	0	✗	0	✗	4	✗	48	✓	230	✓	804	✓	5 746	✓
8	10011100	9-10-30-40-89-90-100-125	0	✗	0	✗	0	✗	2	✗	18	✓	83	✓	330	✓	3 026	✓

Table 5.4: The values 11111111 were used in the positions specified in the first column as HT trigger pattern. The remaining columns detail the trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 16 for HT location.

	CT1		CT2		CT3		CT4		CT5		CT6		CT7		CT8	
positions	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
1-23-27-48-74-95-106-127	1	✗	0	✗	0	✗	1	✗	46	✗	250	✓	759	✓	3 084	✓
3-7-33-45-48-82-94-127	2	✗	2	✗	0	✗	1	✗	86	✓	313	✓	945	✓	7 248	✓
5-7-11-69-73-100-121-126	1	✗	1	✗	1	✗	1	✗	74	✓	369	✓	1 009	✓	6 426	✓
8-13-29-47-60-83-98-110	1	✗	1	✗	2	✗	2	✗	46	✗	222	✓	670	✓	4 125	✓
9-18-28-52-63-89-90-105	1	✗	1	✗	1	✗	1	✗	44	✗	172	✓	554	✓	1 330	✓
10-15-20-50-65-80-92-98	1	✗	1	✗	1	✗	1	✗	26	✓	145	✓	462	✓	5 654	✓
0-2-12-17-87-90-93-97	0	✗	0	✗	0	✗	2	✗	37	✓	185	✓	526	✓	7 073	✓
7-17-30-56-67-90-112-118	1	✗	1	✗	1	✗	2	✗	34	✗	186	✓	537	✓	3 609	✓

Table 5.5: The positions 3-11-25-29-70-88-97-119 are fixed and used with the patterns specified in the first column as HT trigger pattern. The emaining columns detail HT trigger and location capabilities of the CTℓ test sets, using Algorithm 16.

	CT1		CT2		CT3		CT4		CT5		CT6		CT7		CT8	
pattern	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
11111111	1	✗	1	✗	3	✗	0	✗	52	✓	258	✓	812	✓	4 646	✓
10111111	0	✗	0	✗	2	✗	1	✗	10	✗	47	✓	227	✓	1 458	✓
11101101	0	✗	1	✗	1	✗	1	✗	17	✓	65	✓	272	✓	2 587	✓
01111010	0	✗	0	✗	2	✗	2	✗	8	✗	65	✓	252	✓	1 897	✓
00100111	0	✗	1	✗	0	✗	2	✗	8	✗	122	✓	388	✓	1 411	✓
01001010	0	✗	0	✗	1	✗	1	✗	11	✗	61	✓	246	✓	1 446	✓
01000100	0	✗	0	✗	0	✗	0	✗	15	✓	83	✓	307	✓	788	✓
00000100	0	✗	1	✗	0	✗	0	✗	46	✓	135	✓	494	✓	3 168	✓

Table 5.6: The given patterns were used in the specified positions as HT trigger patterns. The remaining columns detail the trigger and location capabilities of the $rand(CT\ell)$ arrays for $\ell = 1, \ldots, 8$, using Algorithm 16 for HT location.

			$rand(CT1)$		$rand(CT2)$		$rand(CT3)$		$rand(CT4)$		$rand(CT5)$		$rand(CT6)$		$rand(CT7)$		$rand(CT8)$	
ℓ	pattern	positions	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc	#trig	loc
1	0	13	7	✗	40	✓	66	✓	161	✓	2 977	✓	14 914	✓	51 906	✓	297 355	✓
2	11	70-120	4	✗	21	✓	31	✓	84	✓	1 459	✓	7 508	✓	25 858	✓	148 628	✓
3	101	5-26-111	3	✗	5	✗	14	✓	32	✓	722	✓	3 835	✓	13 123	✓	13 123	✓
4	1100	0-30-110-127	0	✗	2	✗	10	✓	15	✓	361	✓	1 861	✓	6 492	✓	37 294	✓
5	01100	0-20-70-126-127	1	✗	2	✗	8	✓	7	✗	196	✓	903	✓	3 223	✓	18 606	✓
6	001111	1-21-79-100-101-123	0	✗	1	✗	5	✗	4	✗	90	✓	500	✓	1 584	✓	9 368	✓
7	1101010	1-21-50-82-101-111-123	1	✗	3	✗	4	✗	4	✗	42	✓	252	✓	817	✓	4 630	✓
8	10011100	9-10-30-40-89-90-100-125	0	✗	0	✗	0	✗	1	✗	29	✓	111	✓	470	✓	2 398	✓

Table 5.7: Measurements regarding the fast pattern location properties (i.e. number of patterns that can be located via Algorithm 16) of the $rand(CT\ell)$ arrays. The column 'loc' denotes whether all patterns can be located based on the respective array; the column "miss %" gives the percentage of patterns of length ℓ that can *not* be successfully located - "miss #" the total number; the column "time" denotes the time consumed for the measurement.

	$rand(CT1)$				$rand(CT2)$				$rand(CT3)$				$rand(CT4)$			
ℓ	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time
1	✗	60.54%	155	0 sec	✓	0	0	0 sec	✓	0	0	0 sec	✓	0	0	0 sec
2	✗	97.94%	31 844	3 sec	✗	12.03%	3 914	5 sec	✓	0	0	7 sec	✓	0	0	15 sec
3	✗	99.94%	2 729 530	315 sec	✗	76.64%	2 093 121	372 sec	✗	2.65%	72 462	575 sec	✓	0	0	1 103 sec
4	✗	99.99%	170 686 267	22 097 sec	✗	98.53%	168 175 085	23 065 sec	✗	54.43%	92 904 935	33 770 sec	✗	0.32%	546 726	67 780 sec

Table 5.8: Measurements regarding the fast pattern location properties (i.e. number of patterns that can be located via Algorithm 16) of the $rand_N(CT\ell)$ arrays. The column "loc" denotes whether all patterns can be located based on the respective array; the column "miss %" gives the percentage of patterns of length ℓ that can *not* be successfully located - "miss #" the total number; the column "time" denotes the time consumed for the measurement.

	$rand_{11}(CT1)$				$rand_{37}(CT2)$				$rand_{112}(CT3)$				$rand_{252}(CT4)$			
ℓ	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time	loc	miss %	miss #	time
1	✗	91.79%	235	0 sec	✓	0	0	0 sec	✓	0	0	0 sec	✓	0	0	0 sec
2	✗	99.83%	32 457	4 sec	✗	45.46%	14 779	4 sec	✗	0.009%	3	7 sec	✓	0	0	12 sec
3	✗	99.99%	2 730 926	345 sec	✗	94.30%	2 575 258	334 sec	✗	8.94 %	244 281	516 sec	✗	0.002%	60	862 sec
4	✗	99.99 %	170 687 887	20 985 sec	✗	99.81 %	170 355 238	25 096 sec	✗	71.95 %	122 803 555	30 315 sec	✗	4.90 %	8 372 513	51 522 sec

In the second set of experiments, we use the same combinatorial test sets CTt for $t = 1, \ldots, 8$ in conjunction with Algorithm 16 in order to locate HTs of length eight that have the trigger pattern 11111111 in different positions. Table 5.4 documents the results of our experiments, with column headings corresponding to those of Table 5.3.

In the third set of experiments we fixed the input gates that are consumed by the HT and vary the values in the activation pattern (with varying Hamming weight, i.e. different numbers of ones). Table 5.5 shows the results of these experiments; again, the column headings are identical to those of Table 5.3. Reflecting on the experiments documented in Table 5.4 and 5.5, we can see that some test sets CTt with $t < 8$ can locate the HTs of length eight. More detailed, we see that based on the test sets CT1, CT2, CT3 and CT4 we cannot locate any of the HTs of length eight, which is expected as they are designed for locating lower length HTs. In several cases these test sets also fail to trigger the HTs in the first place. The test set CT5 can trigger all examined HTs of length eight, and it is possible to also locate them in half of the cases. Based on the results of the test sets CT6, CT7 and CT8 it is possible to locate all examined HTs of length eight. In general it is more likely being able to locate a HT when using CTt test sets for larger t, because the test sets are larger. However, there is no guarantee that all HTs of length eight can be located with test sets CTt for $t \leq 7$. Nonetheless, we are ensured to locate all HTs of length eight when using the test set CT8.

We wish to highlight that it is always possible to locate HTs of length ℓ, not just those using the examined trigger patterns, when a combinatorial test set CTt with $\ell \leq t$ is used in conjunction with Algorithm 16. The trigger patterns used in these experiments were arbitrarily selected and can certainly not *guarantee* the effectiveness of our proposed approach, but only serve as a means to exemplify it. However, thanks to the combinatorial properties of detecting arrays and the arguments of Proposition 5.2.8 we can safely argue that the previously mentioned location capabilities always hold.

Analyzing Length ℓ Trojans with Random Arrays

To further illustrate that some trigger patterns can also be located by arbitrary arrays, we conduct the same experiments documented in Table 5.3 with the randomly generated test sets $rand(CT\ell)$ and try to locate the trigger patterns via Algorithm 16. The results of these experiments are given in Table 5.6. We can see that the randomly generated arrays tend to perform similarly to the combinatorial test sets CTℓ, for $\ell = 1, \ldots, 8$, exhibiting comparable trigger and location capabilities.

These experiments raise the question, how well random arrays are suited for HT location in general. As these illustrative experiments rely on some exemplary random samples and do not provide a comprehensive assessment, we measure how many patterns of a given length can be located by Algorithm 16 when using randomly generated test sets, to address this question.

For this purpose, for the arrays $rand(CT\ell)$ (with $\ell = 1, \ldots, 4$), we measure how many patterns are locatable via Algorithm 16. To this end, we first generate the respective oracle

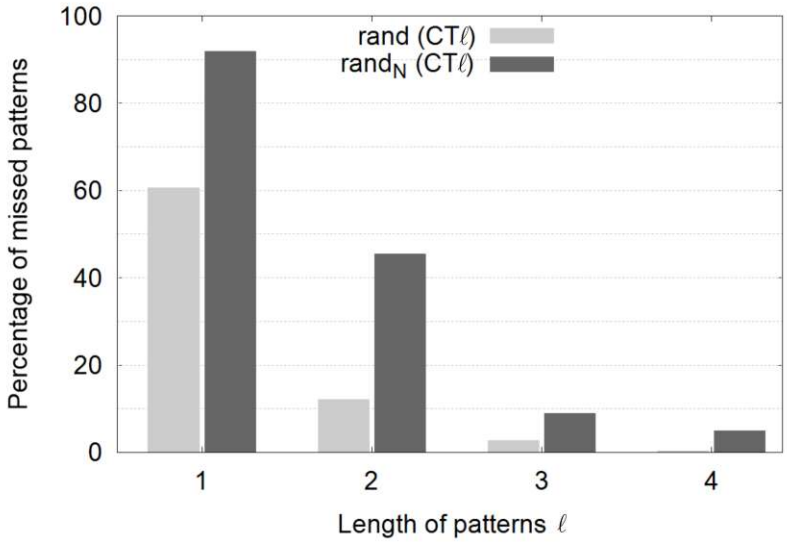


Figure 5.2: The percentage of missed patterns when using randomly generated arrays ($rand(CT\ell)$ and $rand_N(CT\ell)$) on the vertical axis, for the values of $\ell = 1, 2, 3, 4$ on the horizontal axis.

column o for each pattern by checking which test vectors of the random array $rand(CT\ell)$ cover the specific pattern. Second, we call Algorithm 16 on input $(rand(CT\ell), o)$ and check if the returned pattern τ equals the specific pattern at hand. We implemented this procedure in Matlab and conduct these experiments for all patterns of length $\ell = 1, 2, 3, 4$ and measure how many can be located with each random test set, where the focus of interest rests on the number of located patterns of length ℓ when using the $rand(CT\ell)$ test set. The results of these measurements are given in Table 5.7. The diagonal entries of Table 5.7, i.e. the values of the percentage and total number of patterns of length ℓ that were *not* correctly located by $rand(CT\ell)$, are visualized in Fig. 5.2 and Fig. 5.3. We can see that the randomly generated arrays $rand(CT\ell)$ are not capable of locating all patterns of length ℓ, while the equally sized combinatorial test sets CTℓ are capable of doing so. Further, we see that for increasing length of the pattern ℓ, the randomly generated test sets $rand(CT\ell)$ seem to increase their performance, in terms of locating a higher percentage of HTs via Algorithm 16.

Fast Pattern Location with Random Arrays Sized as Optimal Covering Arrays

As the combinatorial test sets utilized in our experiments are not optimal, we conduct the same measurements as reported in Table 5.7 with randomly generated test sets $rand_N(CT\ell)$, that are of the same size as (currently) best known approximates to optimal $(1, \bar{\ell})$-detecting arrays for 128 binary parameters. The number of tests for these sets can be found at [97]. However, as the actual test sets are not provided, we could

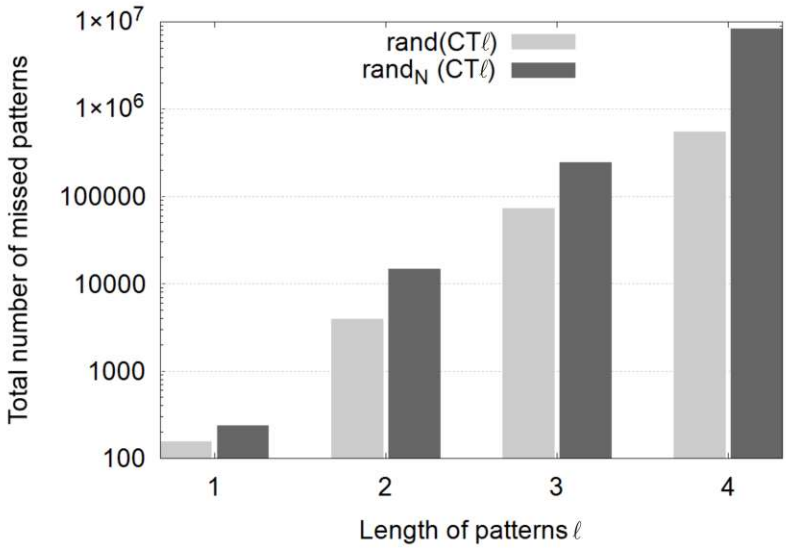


Figure 5.3: The total number of missed patterns when using randomly generated arrays ($rand(CT\ell)$ and $rand_N(CT\ell)$) on the vertical axis as base-10 logarithms, for the values of $\ell = 1, 2, 3, 4$ on the horizontal axis.

not use them for our experiments documented in Subsection 5.2.6. The results of our measurements are presented in Table 5.8. Due to the decreased number of tests, the location capabilities of the $rand_N(CT\ell)$ test sets are further reduced when compared to those of the $rand(CT\ell)$ - we can see that the numbers of HT trigger patterns that are not correctly located increase compared to Table 5.7. Again, we visualize the diagonal entries of Table 5.8, i.e. the results regarding the location of length ℓ HTs with $rand_N(CT\ell)$ for $\ell = 1, 2, 3, 4$ in Fig. 5.2 and Fig. 5.3. These measurements highlight even more the advantage of pattern location based on combinatorial testing compared to random testing approaches.

Slow Pattern Location Properties of Random Arrays

For the sake of completeness, we also conduct experiments regarding the location of HTs when the randomly generated test sets $rand(CT\ell)$ are used in conjunction with Algorithm 14. Due to the time complexity of the location via full enumeration used in Algorithm 14 (see Remark 5.2.6 and Remark 5.2.9), it is not feasible to run a *full enumeration* version for all possible HT trigger patterns, but rather a *decision* version. This means that for given $\ell \in \{1, 2, 3, 4\}$, we iterate over all possible HT trigger patterns of length ℓ, generate the oracle column o for the respective test set A_{rand} and check if Algorithm 14 successfully reconstructs the trigger pattern on input A_{rand}, o and ℓ. If a trigger pattern is reconstructed, we proceed with the next trigger pattern of length ℓ; if the pattern is not reconstructed we abort the search and record that the used test set A_{rand} cannot locate all HTs of length ℓ based on Algorithm 14.

The results of our experiments are documented in Table 5.10. If a pattern exists that is not located correctly, we represent this with ✗, otherwise we use ✓. If we were unable to perform a computation due to restrictions of our existing computing infrastructure, we denote it with **?**. As a result of the decision version of the measurements, we cannot provide the number of HTs of length ℓ that were not located correctly. We can see that the test sets $rand(CT\ell)$ are capable of locating all HT trigger patterns of length ℓ via Algorithm 14. Compared to the location via Algorithm 16, when using Algorithm 14 the test sets $rand(CT\ell)$ can locate all trigger patterns that use one additional gate. This can be explained by the more general location method used in Algorithm 14 that is more suited for pattern location using arbitrary test sets. However, note that conducting such measurements for higher strengths with the aim of obtaining guarantees for the location of HTs is infeasible in practice. For example, to verify that an arbitrary array is capable of locating all HTs of length five, using the technique described above would require processing a multiple of $\binom{128}{5}^2 > 6 \cdot 10^{16}$ patterns of length 5, a requirement that cannot be satisfied using current computing infrastructure.

To showcase this argument and to highlight the advantage of Algorithm 16 in conjunction with combinatorial test sets, we compare the runtimes of Algorithm 16 and Algorithm 14 when used to locate an HT of length $\ell = 1, 2, 3, 4$. As the time needed for test execution does not influence this comparison, we do not include it in the measurement and instead only compare the runtimes of Algorithm 16 over Algorithm 14, provided that the testing oracle o is already known. To represent 100 randomly generated HTs of lengths $\ell = 1, 2, 3$ and 4, we select 100 trigger patterns out of the $2^\ell\binom{128}{\ell}$ total possible trigger patterns of length ℓ uniformly at random. For each of these HTs, we generate the respective testing oracles o and run Algorithms 14 and 16 on the input $(CT\ell, o, \ell)$ and $(CT\ell, o)$, respectively. The two Algorithms are implemented in Matlab and the experiments are run on a machine with an Intel i9-9900 CPU clocked at 3.60 GHz with 64GB of RAM. The results of these measurements can be found in Table 5.9. The table shows that the runtime of Algorithm 16 remains under one millisecond in all cases and appears to grow only linearly in ℓ. This is due to its runtime being primarily influenced by the number of failing tests, which grows slowly with increased ℓ and numbers of test vectors in CTℓ (see, for example, Table 5.3). This observation is further justified by conducting HT location for HTs of length $\ell = 8$ with Algorithm 16 and the CT8 test set, which on average (again for 100 randomly selected HT activation patterns) needs only 0.2260 seconds for locating the HT. Compared to that, the runtime of Algorithm 14 seems to follow an exponential growth in ℓ, which is explained by the factor of $\Theta(\binom{k}{\ell})$ being present in the Algorithm's runtime, see also Remark 5.2.9. These large runtimes for the location of an HT of length $\ell \geq 5$ are the reason why Tables 5.7 and 5.8 are only provided for $\ell \leq 4$.

Finally we want to note, that the experiments and measurements of randomly generated test sets have been conducted only for one specific random test set for each instance. It would have been desirable to conduct the experiments documented in Tables 5.7, 5.8 and 5.10, for all lengths of HTs and for several test sets in order to present results regarding the average and expected performance of random test sets for all $\ell = 1, \ldots, 8$. However,

Algorithm / HT length	Algorithm 14	Algorithm 16
$\ell = 1$	$278 \cdot 10^{-4}$ sec	$1.0 \cdot 10^{-4}$ sec
$\ell = 2$	2.0138 sec	$1.1 \cdot 10^{-4}$ sec
$\ell = 3$	139 sec	$1.2 \cdot 10^{-4}$ sec
$\ell = 4$	5 503 sec	$1.7 \cdot 10^{-4}$ sec

Table 5.9: Comparison of the runtimes of Algorithm 14 and 16 when used to locate an HT of length ℓ based on testing with the combinatorial test set CTℓ and the respective testing oracle. The given runtimes are averaged over 100 runs, where the HTs were placed randomly.

	$rand(T1)$	$rand(T2)$	$rand(T3)$	$rand(T4)$
ℓ	loc	loc	loc	loc
1	✗	✓	✓	✓
2	✗	✗	✓	✓
3	✗	✗	✗	✓
4	✗	✗	✗	?

Table 5.10: Measurements regarding the slow pattern location properties (i.e. patterns that can be located via Algorithm 14) of the $rand(CT\ell)$ arrays. The column 'loc' denotes whether all patterns can be located based on the respective array.

this would involve computationally expensive and unaffordable tasks. In this sense, the experiments conducted for random test sets should be understood as experiments with *arbitrary* test sets with a given number of tests.

5.2.7 Threats to Validity

The presented combinatorial methods for HT trigger pattern identification rely on some information about the length ℓ of the pattern that shall be located. We are aware that a testers generally do not know the length of the trigger pattern that they want to identify. However, we demonstrated that an *upper bound* on the length of the inserted HT is sufficient to precisely identify it with the proposed combinatorial methods (combinatorial test sets in combination with Algorithm 16). A potential attacker is always faced with a trade-off: On one hand longer trigger patterns are more rare and thus harder to detect and locate via logic testing; on the other hand they also necessitate larger HT trigger circuits, thus consuming more area and power, making detection by physical inspection or side-channel analysis more likely. Hence, we can assume that an attacker will not use a full 128-bit pattern, but rather some pattern of length ℓ, where $\ell \ll 128$. A Tester, on the other side, is faced with the inverse problem: The tester has to select a strength t for the combinatorial test set that is high enough to locate a potential HT while avoiding excessive resource consumption. The tester's selection of the chosen strength t is likely influenced by the circuit under test and other HT detection techniques involved in the

testing process. Aside from this, available resources may determine the efforts affordable for testing, which is common to all testing problems in general. For example, a tester can select the strength t based on the test suite size and the available time budget. If combinatorial test sets are precomputed and readily available, a resource consumption assessment is rather easy. Finally, experience and domain knowledge of the tester may influence the choice of t.

The work conducted in this section shows how HTs with trigger patterns of length up to eight can be located precisely.

The threat model of considered in the above work assumes that an attacker designs his HT so that it only consumes primary inputs. This is obviously a drastic restriction, as other works [385] notably exclude primary inputs (and outputs) from their investigations. However, once more, we made these assumptions for the sake of simplicity and clarity of the experimental evaluation and in order to present the proposed methodology in a concise manner. The method proposed in this section can also be applied more generally to any set of gates of the circuit, as long as their input can be actively controlled in order to apply combinatorial testing methods. To this extent the proposed method can be applied to sets of suspicious gates of a circuit that were previously identified using alternative approaches, such as probability analysis or the topology of the circuits layout as proposed in [386].

We want to mention that our approach to HT location scales well with the number k of gates that are modeled as subject to a potential attack and are thus represented as parameters in the combinatorial model. The number of test vectors grows logarithmically in the number of modeled gates k, which is due to the number of rows of detecting arrays growing logarithmically in the number of parameters, which was shown in [62, Th. 8.6].

5.2.8 Summary and Outlook

In this section, we introduced a method for identifying trigger patterns of hardware Trojans, which are triggered by combinational ℓ bit patterns (e.g. trigger circuits composed of AND-gates and NOT-gates) in the primary input. Using concepts from CT-FLA, our method relies only on the applied set of test vectors and the testing results in order to locate the HT trigger pattern, while the circuit under test is considered a black box. We demonstrated the effectiveness of our approach in a concrete case study, utilizing it to locate HTs with trigger patterns of length up to eight embedded in a circuit that implements the AES symmetric-key encryption algorithm with 128 bits key length. Our results show that our testing methodology can perform trigger pattern identification in a negligible amount of time while providing the guarantee of locating any HT with trigger pattern of length up to eight.

As future work, one might seek to lift the restriction of locating HTs with trigger circuits consisting exclusively of AND-gates and NOT-gates. If we apply combinatorial methods for the location of multiple FITs, e.g. when using (d, t)-detecting arrays with $d \geq 2$ for the testing of integrated circuits, we can also capture HTs that additionally

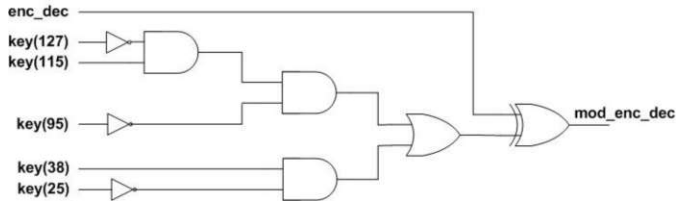


Figure 5.4: An example of an HT with a generalized trigger logic, activated by the signals 10 in position 25-38 or by the signals 010 in positions 95 - 115 - 127.

use OR-gates in their trigger circuit. For example, an HT with a trigger circuit that combines two of the trigger circuits as considered in this section (see e.g. Fig. 5.1) via an OR-gate (on the highest level of its concrete syntax tree) can be located with a $(2,t)$-detecting array of appropriate strength t. Consider an HT with a trigger logic of the form $(\neg g_{25} \wedge g_{38}) \vee (\neg g_{95} \wedge g_{115} \wedge \neg g_{127})$ (see Fig. 5.4). This Trojan can be located applying combinatorial testing based on a $(2,\bar{3})$-detecting array: the HT is triggered, when the pattern 10 or 010 appears in the positions 25 - 38 or 95 - 115 - 127 respectively. In terms of combinatorial testing, this means that we have to locate the failure inducing $\leq$3-way interactions $\{(25,0),(38,1)\}$ and $\{(95,0),(115,1),(127,0)\}$ - which we are guaranteed to find when testing with a binary $(2,\bar{3})$-detecting array. The logic of any combinational circuit is (logically) equivalent to a circuit that uses exclusively OR-gates, AND-gates and NOT-gates, which is provided by considering the disjunctive normal form (DNF) of the trigger logic. Since our proposed method does not rely on the actual implementation of the HT trigger, but merely on its logic, this means that the work presented in this section can be generalized for combinational HTs with arbitrary trigger circuit logic. As straightforward as the combinatorial modeling of such HTs might seem, the challenges arising in this context is caused by the fact that the existing work on the efficient generation of combinatorial test sets enhancing non-adaptive CT-FLA for multiple failure inducing t-way interactions is rather limited [137]. Only a few algorithms capable of generating such combinatorial test sets in a near-optimal manner are known. Additionally, these algorithms have only been investigated for a small number of failure inducing t-way interactions of small strength [393].

In case non-adaptive combinatorial testing fault localization methods are absent, alternatively adaptive methods may be used, as it was done in the case study documented in Section 5.4, or for use cases as they are discussed in the next section.

5.3 Exploring Combinatorial Testing Fault Localization for Explainable Artificial Intelligence

In this section we review combinatorial methods for explainable artificial intelligence (XAI) and consider them from the perspective of different properties of explainable AI proposed by various researchers.

We begin by giving the context by means of recent developments of XAI, before we review recently introduced *properties* for XAI. Thereafter we briefly discuss their feasibility, outlining a parallel to theoretical computer science. Next, we review combinatorial methods for XAI, which are based on combinatorial testing fault localization (CT-FLA) approaches. Last, we reflect on these and hypothesize about potential parallels amongst computer science (software testing) and psychology (human cognition and brain capacity).

5.3.1 Background on Explainable Artificial Intelligence

Artificial intelligence (AI) systems have improved rapidly, with their performance now surpassing human abilities in many tasks, especially vision and image recognition applications, but also in more safety-critical tasks such as autonomous vehicles [394], [395]. The increase in numbers of AI applications, and their integration into everyday life, has created a public demand for understanding the behavior and decisions of AI systems. This demand has led to the research field of explainable AI (XAI) that has the goal of making AI systems or their decision-making humanly understandable.

Recent research in explainable AI has been driven by recognition that despite their success in many areas, current AI/ML systems are opaque to users. Consequently, many researchers around the globe are actively working on bringing the "X" to the "AI". Artificial Neural networks and other black-box functions can be used to generate sufficiently accurate probabilities for classification or decision problems in autonomous systems, but trust depends also on the system's capacity to explain or justify an output [396], [397]. Specifically, users should be able to understand why a particular decision was made, or classification chosen, and why another option was excluded by the AI/ML algorithm. This kind of knowledge is needed not only for human trust, but may also be useful for improving and fine-tuning algorithms, or for analyzing failures that will inevitably occur in any engineered system. In addition to these basic needs, some industries are likely to require explainability in order to certify system safety [398].

Moreover, the nature of many AI/ML algorithms are incompatible with current safety-critical system certification approaches, see [272] and reference [8] therein. These regulations are often based on showing that a system has been tested according to some structural code coverage criteria, such as MCDC [399]. Structural coverage can show that code has been thoroughly tested, according to some definitions, but in AI/ML, classifications and decisions depend on data, and the core algorithm a generic function such as a deep neural network. Evaluating structural coverage in such a system is not sufficient for showing that correct conclusions are reached for inputs that will be encountered in future use. It is sometimes suggested that neuron coverage is an adequate neural net analogous to structural code coverage, but it is not yet clear how much neuron coverage or similar measures can be relied upon for AI/ML certification, and these measures do not provide the necessary explainability that will be needed for human trust and validation [400].

XAI approaches are as diverse as there are AI systems, including self explaining systems, externally explained systems, global explainable AI algorithms (e.g. SHAP [401]) and

per-decision explainable AI algorithms (e.g. LIME [402]). These and other developments in XAI can be found in a recent survey [403]. In general, three types of approaches to XAI exist today. Early expert systems that relied on "if – then" rules had an inherent explainability – if certain properties existed, then the rules provided conclusions, such as symptoms indicating a particular disease. These rule based expert systems are often less accurate than various neural net algorithms [396], so there is a trade-off between accuracy and explainability [404], where the more accurate algorithms provide less natural explainability. A second approach to XAI investigated the internal addition of explanations to artificial neural networks, but it is still unclear if the explanations thus provided are adequate, or if this technique will reduce the performance of the neural networks [405].

A more commonly used XAI approach is model induction, which attempts to infer an explainable model from system inputs and outputs. This approach applies statistical methods to find features that are most closely related with particular outputs. The notion of matching features with outputs can also be generalized to use combinations of values, rather than single feature values [272]. By using combinations of feature values, it is possible to produce explanations or justifications that more closely resemble conventional expert system rules: "if A and B and C, then conclusion is X".

5.3.2 Properties of XAI

Next, we want to have a look at AI and XAI. It is presently difficult to give a generally accepted and detailed definition of AI. There exists a plurality of approaches on how to define AI, for example being centred around human performance or rather around thought processes and reasoning, see [260] and references therein. On top of that the understanding of AI is controversial and may also change over time - just think of Deep Blue defeating Garry Kasparov [406].

However an AI system may appear, today we see various realizations of AI, the majority using Bayesian networks, deep learning or symbolic approaches [407]. We point out these different understandings and realizations of AI, because we believe that it has a heavy impact on the explanations that we can expect to get or produce for the respective AI. A full understanding of XAI will require extensive human factors research. Even though the topic of explanation has been studied in psychology for decades, we believe that much of this work can be adapted to the problem of explainability in AI [408], [409], [410], [411].

Basic principles for XAI are not yet well understood, as reflected in the great variability of ideas that have been proposed. The US National Institute of Standards and Technology (NIST) published a set of principles for public comment, specifically because this field is so new and unsettled [412]. This NIST inter-agency report suggested four principles for discussion, as possible fundamental properties. As consensus develops, a taxonomy of XAI of XAI principles could serve as a means to guide the future development of XAI towards dimensions along which XAI explanation systems could be measured. The four *principles* proposed in [412] are:

- Explanation: Systems deliver accompanying evidence or reason(s) for all outputs.
- Meaningful: Systems provide explanations that are understandable to individual users.
- Explanation Accuracy: The explanation correctly reflects the system's process for generating the output.
- Knowledge Limits: The system only operates under conditions for which it was designed or when the system reaches a sufficient confidence in its output.

A different categorization of the human factors aspects essential for explanations, whether machine or human-generated, is provided in Ehsan et al. [413], who studied how psychological research on understanding can be applied to machine-generated explanations. They consider the following *dimensions* in order to rate the endorsement of explanations:

- Confidence: This rationale makes me confident in the character's ability to perform it's task.
- Human-likeness: This rationale looks like it was made by a human.
- Adequate justification: This rationale adequately justifies the action taken.
- Understandability: This rationale helped me understand why the agent behaved as it did

Acknowledging the lack of agreement on XAI at this time, we adapt from the related work discussed above to propose a set of properties that are well suited to the ideas that are developed in the rest of this section. These properties are summarized below, and are closer elaborated later:

1. *Existence*: For each output an explanation is provided that helps to understand why this output was generated.
2. *Clarity*: The explanations can be understood comprehensible by humans/users.
3. *Adequate justification*: The explanations adequately justifies the system's output or its process for generating the output.
4. *Trust*: In the system to accurately generate the output based on a description of events and its environment.

Together, the first three properties aim for ensuring that generated explanations are *plausible* to humans. We want to mention that there is a fundamental difference between *the output* that a system generates and *how* (the) output is generated, i.e. the process

Ehsan et al. [413]	Draft NISTIR 8312 [412]	Proposed Properties
Understandability	Explanation	*Existence*
Human-likeness	Meaningful	*Clarity*
Adequate justification	Explanation Accuracy	*Adequate justification*
Confidence	Knowledge Limits	*Trust*

Table 5.11: Overview of the four *dimensions* given in Ehsan et al. [413], the four *principles* described in [412] and the properties of XAI explanations proposed in this section.

that led to the output. We therefore need to be precise with regards to our demands to XAI: do we want an explanation for the output of an AI-based system, or an explanation of the underlying process? We will elaborate on this and similar questions in the following. We provide an overview of the different aspects of XAI considered in this section can be found in Table 5.11.

5.3.3 Remarks Related to the Development of XAI

We first want to propose a classification of XAI as an adaptation and addendum to the works cited above. We believe it is worthwhile to explicitly mention distinctive features of XAI as these will help to reason about it, especially with regard to what we can expect and demand from explanations.

Classifications of XAI

For example, the general characteristics of plausible explanations (existence, clarity and adequate justification) must take into account that explanations may need to be varied for different users, who have different levels of knowledge and expectation. However, as Hilton [414] states: "*The verb 'to explain' is a three-place predicate: Someone explains something to someone*". Thus, not only the receiver of the explanation is crucial in this differentiation, but also the matter that is being explained, i.e. the AI system, its input and output. In the following, we give some dimensions and criteria along which we can differentiate XAI. We do not claim completeness or even correctness and leave it to future investigations to revise or improve them. However, we believe it is an important step in understanding where specific explanations can be applied (to which AI) and for whom they are produced (the user or receiver of the explanation). In the following discussions, we refer to these two entities as the *human* and the *AI*.

Who receives the explanation? It is generally accepted that we need to distinguish explanations according to who asks for them, with the expert versus non-expert example being the most prominent one. One possible way is to differentiate a number of groups that differ in the quality and quantity of information they demand or expect.

- *Non-experts*: They want to know the key reasons why a specific output is produced - details are not needed, or even desired.

- *Experts*: They want to have detailed reasons why a specific output is produced, however these need not or should not be dependent on the AIs implementation.
- *Developers*: They want many details that can or should be implementation dependent, in order for the explanation to guide his or her debugging or development process.
- *Algorithms*: They require details in a machine readable format, where requirements can be formally specified.

We can see in the last group, that the *human* (i.e. the explanation receiving part) can also be an AI, or an algorithm more general, e.g. an algorithm that is rating the quality of explanations.

Who gives the explanation? We can differentiate XAI systems according to where the explanation comes from:

- *Self-explainable models/systems*: These are AI systems that provide the needed explanation themselves; these can be systems where the underlying algorithm itself represents the explanation, e.g. AI systems based on decision trees or ones that provide explanations without giving algorithmic details, such as class activation mappings [415].
- *External explanation models/systems*: In this case the explanation of the AI's output is produced based on a separate algorithm.

What is being explained? The subject of explanation can be differentiated in various ways:

- *Decision vs decision process*: Is the output explained or the process that leads to the AI system's output?
- *Global explanations vs per-decision explanations*: Is a single output explained or a set of outputs?
- *Kind of AI system*: For example a classifier/decider, an AI system performing tasks like driving a car or an automated theorem prover.
- *What is the input to the AI*: The explanation for an output has to relate to the input (black-box case). An AI algorithm also starts with the input, hence an explanation for the AI relates to the input.
- *Black-box vs white-box model*: Is the internal mechanism of the AI system accessible or not?

Again, we do not claim that this list of categories is complete. Further, we consider also that the above categories are not necessarily excluding each other and may very well be mutually influenced as there exist some causalities between them. For example, when we ask for an explanation of the decision process of an AI system, then the system is generally a white-box model, as we need to have access to its internal working mechanism in order to explain it. Furthermore, self-explaining systems already provide such insights. Another example is that a global explanation could be used to generate a per-decision explanation or any black-box approach can be also applied to a white-box AI system.

Solution Processes to NP-Complete Problems May Be Too Difficult to Explain

In this section we focus on the comparison of explaining AI output generation processes versus explaining outputs of AI systems.

The "adequate justification" component of explanations can encompass *how* the system came to its conclusion, and the system's "output" itself. Of course, an explanation for an output process can yield an explanation for the resulting output itself, but it can be significantly more difficult to explain an output process compared to an output. In that regard, let us consider the following analogy.

From computational complexity notions we know that *finding a solution* to a problem and *verifying* a solution as such can result in significant difference of computational effort. The well known P versus NP problem, includes the question whether the solution to an NP-complete problem can be found as easily as it can be verified. *Let us assume that P ≠ NP, how does this influence the explainability of AI?* For NP-complete problems the length of the solution derivation would not be bound by any polynomial function in the length of the input, while for the solution verification there would be such a bound.

This analogy to computational complexity is not too far off compared to the explainability of AI: Assume we have developed an AI system that (optimally) solves routing problems, such as TSP. Asking for a *meaningful* and *accurate explanation* of the *decision process*, means asking for an understandable (somewhat short) and correct explanation of the lengthy solution process to an NP-complete problem [416]. To give another example, lets consider a constraint satisfaction problem (CSP) solver as an instance of an AI system. The decision process to a query itself can be extremely lengthy, but once a decision is made, it can be verified fairly easy in some cases. For example, provided the correct formulation, we can query a CSP solver whether a map can be colored with only three different colors, which is an NP-complete problem [417]. The derivation process itself can be extremely lengthy and difficult to follow, but when a solution is found and the answer is "yes", then this can be easily explained by providing the three-coloring of the map, something very accessible to human beings. These remarks beg to ask the question whether there is an analogue notion to NP-completeness in explainable AI, i.e. *a solution process that requires significantly more effort to be explained compared to the explanation of the solution.*

Clarifying if solution processes and solutions to NP-complete problems are an example of this is one way to address this question. Solution processes can appear in the form of a decision or a search algorithm. A related research question is: Does the *length* of a solution process, here we mean the formal length of the derivation, make an explanation more difficult to generate?

5.3.4 CT-FLA for XAI

The work in [272] presents combinatorial methods that are inspired by ideas and methods from CT-FLA for explaining classifications and decisions made by AI systems. The justification of the assignment of an object to a specific class is given by the identification of feature combinations that are present in the object and in members of the assigned class, while being absent (or rare) in objects of other classes. A related black-box approach for per-decision explanations of AI systems is presented in [402].

We briefly outline the connection between CT-FLA and classification systems.

Linkage Between XAI and Notions of CT-FLA

To apply CT-FLA to explain classifications generated by AI systems, we need to correspond the notions of AI classification systems with the respective ones related to CT:

- The *input* to a classification system as the equivalent to a *test vector* in CT,
- The *assigned class* to an object as the equivalent of the *resulting pass/fail-assignment* of the test vector execution,
- The *unique* or *characteristic feature combination* as the equivalent of the *failure inducing t-way intersection (parameter-value combination)*.

Provided this mapping of notions, in order to search for an explanation why an AI system classifies a specific object o to a class c, we simply map class c to *fail*, all members of c to the failing tests and members of any other class than c to the passing tests to obtain an *annotated test set*. Then the identification of a feature combination that explains the classification of objects to class c is equivalent to finding a failure-inducing t-way interaction in the obtained annotated test set. Once a failure inducing t-way interaction of the comprised test set is identified, we have found a feature combination that is present in the members of class c while not present in any other class.

We exemplify this mapping in Table 5.12 where we present a similar example as given in [272], featuring a database of animals with attributes. On the right hand side of Table 5.12 we see a snippet of a database with animal records; due to space limitations only five (of originally 16) attributes are shown. The classification of Testudo as a reptile can be explained by the feature combination triplet (non-aquatic, toothless, four-legged) that is unique to reptiles and present in Testudo. This triplet represents a counterfactual

test	p_1	p_2	p_3	p_4	p_5	result		class	hair	aquatic	egg-laying	toothed	nlegs	object
$t_1 =$	0	0	1	0	3	pass	$\longleftrightarrow$	insect	no	no	yes	no	6	Mantis
$t_2 =$	0	0	1	0	2	fail	$\longleftrightarrow$	reptile	no	*no*	yes	*no*	4	Testudo (Tortoise)
$t_3 =$	0	1	1	0	3	pass	$\longleftrightarrow$	insect	no	yes	yes	no	6	Water scorpion
$t_4 =$	0	1	1	1	1	pass	$\longleftrightarrow$	bird	no	yes	yes	yes	2	Penguin
$t_5 =$	0	0	1	1	2	fail	$\longleftrightarrow$	reptile	no	*no*	*yes*	yes	4	Sand lizard
$t_6 =$	0	0	1	0	3	pass	$\longleftrightarrow$	insect	no	no	yes	no	6	dung beetle
$t_7 =$	0	1	1	1	3	pass	$\longleftrightarrow$	amphibia	no	yes	yes	yes	4	Amphiuma
⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 5.12: Analogy between CT (left) and explanations for an AI system which produces classifications (right).

explanation: if Testudo had 6 legs, it would be classified as an insect. This concludes the review of [272] which shows how methods for CT-FLA can be used to produce counterfactual explanations for AI classification systems.

As an additional remark, we want to mention the notion of *minimal failure inducing t-way interactions* [82], which are parameter-value combinations that when being reduced or deviated, do not necessarily cause tests to fail any more. Translated to the field of XAI, these allow the derivation of *counterfactual cores*, which are feature combinations that when being modified yield different classifications of the AI algorithm and hence serve as a source for counterfactual explanations. Thereby one counter factual core can yield several counterfactual explanations, by modifying one or multiple of its features. For example, the feature combination triplet (non-aquatic, toothless, four-legged) which explains the classification of Testudo as a reptile can give rise to another counterfactual explanation: if Testudo was aquatic and toothed, it would be classified as an amphibia.

Aside from this, it can further be the case that there exist more than one counterfactual core. In software testing there exists the possibility that there appear multiple minimal failure inducing t-way interactions in a single test. Note that the term *minimal* in *minimal failure inducing t-way interaction* should be understood as in lattice theory: it is minimal in the sense, that there is no failure inducing t-way interaction included in it, however it is not (necessarily) a *minimum* in the sense that all failure inducing t-way interactions must include it. Analogously, it can be the case that there are multiple counter factual cores being present in the feature combinations of a single input to an AI classification system. For example, Testudo is carrying not only the feature combination (non-aquatic, toothless, four-legged), but also the feature combination (non-aquatic, egg-laying, four-legged), which is unique to reptiles in the example considered in [272, Fig. 5]. One way to deal with multiple identified counter factual cores is to prioritize the shortest one(s) as primary source for counterfactual explanations. A counter factual explanation that depends only on a single feature is likely to be explained and understood more easily than one that involves, e.g., 6 different features. In other words, more simple explanations can be prioritized over complex explanations.

Further, we want to comment on how to interpret the input to an AI system as a test in CT. In order to do so, one way is to model the input to the AI system by an IPM, analogue the input modeling of an SUT when applying CT. This problem however, is

not genuine to AI and is a rather well known problem in software testing, see e.g. [26], [77] or [1, Chapter 6]. It is not possible to give a generic answer how an AI's input can be represented via an IPM, but since the problem is known in software testing, there is a plurality of methods for abstraction and input space partitioning that can help and provide guidance for the IPM derivation. In some cases the derivation of an IPM can be done in a direct way, e.g. when an AI is classifying entries in a database, as in the example discussed in [272] and reviewed above. In this case an IPM for the input is provided by the database itself, which attributes constitute the parameters of the IPM and the concrete values of the entities in the database yield the values of these parameters. In other cases, the derivation of an IPM might be more complicated, e.g. when an AI is classifying or recognizing images. In this case there is no clear favourite how to model the input images by an IPM, but rather multiple ways can be followed. To mention just two of them: we can consider each individual pixel a parameter and its respective color-code as the parameter value, or - including one step of abstraction - the images can be modeled by an IPM where each parameter represents a feature, that is possible to be present in an images. In the latter case an image is described by a boolean vector indicating absence or presence of specific features of interest.

Once the identification of the AI systems input with test vectors in CT is established, CT-FLA methods can be applied for the identification of characteristic feature-combinations of classes. Again, depending on the application case, different CT-FLA methods are more or less applicable to the case. The crucial point is whether the input that is subject to the AI system can be actively modified and extended. For example if the input can be actively modified, e.g. if the training set of a neural network can be specified in a such a way that the input exhibits the properties of a (d, t)-detecting array for the respective underlying IPM, then non-adaptive CT-FLA methods (as discussed in [62] or [132]) are suited to be applied as a method for XAI. Further, if also additional input to the AI system can be queried, then adaptive CT-FLA methods (such as [130] or [83]) are suited for finding explanations for XAI. The tool BEN [130] might be even applied when the input cannot be extended. In that case the suspicious combinations can provide the sought-after explanations in form of a characteristic feature-combination. Finally, if the input to an AI system is determined, i.e. no manipulation or addition of input is possible, e.g. when the entries of a database are subject to a classification system, tools that identify unique feature-combinations in the given input, such as the ComXAI tool proposed in [272] can be applied to find explanations for AI classifications.

5.3.5 Reflection of CT-FLA Methods for XAI

Having revisited XAI through the four principles and CT-FLA methods, we want to examine *where* the latter can be applied and to which degree we can apply the combinatorial lever.

Research Question: How can CT-FLA methods for XAI be categorized?

Answer: Processing the described categories proposed in Section 5.3.3 bottom up, we can categorize combinatorial methods for XAI as:

- *Black-box*: They are not relying on AI system internals.
- *Input*: They are applicable to systems where the input is modeled via an *IPM*.
- *Kind of AI system*: They are applicable to *classifier and decision systems*.
- *Per-decision explanations*: They provide primarily per-decision explanations.
- *Decisions*: They explain decision outputs, not decision processes.
- *External explanation*: The explanation is independent of the AI system's internals and provided by an external source.
- *Explanation receiver*: The explanations produced are suited for *non-experts, experts* and potentially *other algorithms*.

Note that, CT-FLA methods are primarily suited to produce per-decision explanations, but they can also characterize whole classes and thus not only explain an individual object. Thus, in how far counterfactual cores give a global explanation for an AI system is debatable. Devising the required IPM can be straightforward, e.g. when the input is already given as a list of attributes; or can require to additionally model the input space to the AI system.

Next, we consider again the four properties for XAI that we proposed (see Table 5.11) in conjunction with CT-FLA:

Existence. Translated for CT-FLA methods applied to XAI, especially classifiers, this property requires that for each object that is classified as a member of a specific class, there must be at least one characteristic feature-combination that can be identified for this object and members of its assigned class. Otherwise, the description is clearly inaccurate, and a user cannot trust it, as it has failed to identify any characteristic features.

> **Research Question:** Can decisions of systems that classify input that is modeled via an IPM always be explained via feature-combinations of the input?

Answer: It is up to further investigation whether this question can be answered in the same style as for software faults [418], [25].

Clarity. We outlined how CT-FLA methods provide explanations via feature-combinations.

> **Research Question:** Are explanations generated from CT-FLA for XAI humanly understandable?

Answer: The feature-combinations generated by CT-FLA methods serve as *counterfactual explanations*. We outlined previously (Section 5.3) how minimal failure inducing t-way interactions can yield counterfactual cores. There exist several studies that suggest that counterfactual explanations suit the human way of casual explanations, see e.g. the work of Hilton [414] and references therein, [419]. Further, some works investigate the role of counterfactual explanations in the realm of XAI [409], [420], [421], [422]. This leads us to consider the following:

> **Research Question:** How complex or lengthy can counterfactual explanations become and still be humanly understandable? Further, is the length independent from the classification process?

Answer: A potential answer to this question can be found in the well known observation, by the psychologist Miller [423], [424] states that the capacity of the human brain in terms of short-term memory is limited to about 7 ± 2 *chunks*, i.e. information units. Such or similar insights might translate to an *upper bound* on the strength of feature-combinations that need to be identified as class characteristic by combinatorial methods, as any feature-combination beyond this upper bound is not easily processable by the human brain. This would be a natural bound for the applicability of CT-FLA methods for XAI, and could thus present a psychological analog to the the empirical studies conducted by Kuhn et al as outlined in the introduction 1.2.2, respectively to the *interaction rule* [25] that suggests that it is (largely) sufficient to consider parameter-value combinations of up to six parameters for combinatorial software testing.

Adequate Justification. One cannot expect that "short" feature-combinations can explain all AI decisions, e.g., automated theorem proving or SAT solving, where the results likely depend on the entire input.

> **Research Question:** Are the explanations produced by CT-FLA methods adequate to explain AI decisions?

Answer: This question could be answered (partly) by a case study, comparing explanations from self-explaining classification system with those generated by CT-FLA methods in order to evaluate the explanations. Such a comparison may reveal cases where CT-FLA methods are suited for generating explanations to AI systems, and others where they are not.

Trust. This aspect primarily concerns AI systems, rather than (external) explanation systems, however it raises the following question.

> **Research Question:** Does the absence of a characteristic feature-combination imply an inaccurate action in the given situation?

Answer: This can be reasonably addressed, only once the previous research questions have been addressed, especially we need to know whether decisions within the knowledge limits of the system can lead to characteristic feature-combinations.

To summarize, a number of researchers (see Section 5.3.2) have considered the application of psychological research on explanation quality to the problem of XAI. We investigated the applicability of combinatorial methods to XAI considering these general characteristics of explanation quality and formulated open research questions, providing answers where possible. We can hope that their formulation inspires further research and that answers to these questions lead to a further improvement of combinatorial methods and advance XAI.

5.4 Combinatorial Fault Localization for Automated Driving Functions

In this section we present an application of *combinatorial testing fault localization* (CT-FLA) methods for automated driving function testing in an industrial setting, focusing on a detailed case study based on an autonomous emergency braking system from AVL List GmbH (AVL). Although there exist several related works where CT is used in the automotive domain, to the best of my knowledge, this represents the first study where a *CT-FLA approach* is used in the automotive domain.

In particular, in this section we address the problem of evaluating automatically generated scenarios for virtual validation and verification of automated driving functions. We use CT-FLA to screen parameter settings that lead to critical scenarios in a virtual verification and validation framework used for automated driving function testing – in the following referred to as *virtual driving test platform*. For the domain experts, it is important to find out which parameter settings, and hence which parameters play an essential role in crash scenarios. By means of CT-FLA we are able to identify t-way interactions in the scenario specifications, that (are likely to) result in a crash. The obtained information of (potential) crash inducing t-way interactions can be used in further test scenario generation in order to generate crash and near crash scenarios more targeted. These efforts, must be understood as part of the larger undertaking of verification and validation of automated driving functions, pursued by the contributing parties.

I contributed to this work first and foremost by designing the CT-FLA approach for the given remote setup, and also contributed to the combinatorial test set generation and the analysis of the testing results.

We start by providing some context on automated and autonomous driving, and mention related works underlying the present one. Thereafter, we describe the general boundary conditions and the CT-FLA method we adjusted to them. We give a detailed description of the set-up of our case study, i.e. of the system under test (SUT), the used input-parameter model (IPM), the virtual driving test platform and the testing oracle. Finally, we present and discuss our results.

5.4.1 Background on Automated and Autonomous Driving Systems

The advances in applications of artificial intelligence in the field of automated and autonomous driving systems leads to more complex systems and functions. Advanced driver assistant systems and automated driving systems are safety critical, i.e. a failure or unintended system behavior can cause accidents with severe consequences. This leads inevitable to challenges in terms of quality and safety assurance of automated driving functions. In order to ensure that in every possible situation the an advanced driver assistant or AD system decides for a correct and safe behavior, it is necessary to test related systems and functions extensively under different environmental and traffic conditions. However, the high complexity of the systems in use and the intractable number of input traffic scenarios and environmental conditions makes exhaustive testing impossible, or infeasible to say the least. Rendering validation and verification of AD functions an important and challenging problem [425]. The authors of [426] present a summary of the major challenges in autonomous vehicle testing. They argue that it would require hundreds of a million kilometers driven of road testing, in order to statistically show that an automated vehicle is as safe as a humanly driven vehicle. They thereby point out the that ironically, the safer the vehicle driving is, the higher the number of required test kilometers is. Similarly, Karla and Paddock [427] give an estimation of more than 275 million miles required in order to perform such vehicle testing when underlying the fatality rate of driving in the US. This amounts to a testing time of 400 years assuming that there are 100 cars in use, driving continuously at an average speed of 25 miles per hour.

These numbers stress that distance based validation of autonomous vehicles is not an acceptable solution. Since automated vehicles need to be tested in different traffic situations and in interactions with other vehicles, scenario-based approaches are considered as proper methods for the development of automated driving functions [428], [429], [430]. This however raises the question which scenarios to generate, respectively how to automatically generate scenarios in order to test automated driving functions. Thereby the focus lies on generating *critical scenarios*, i.e. scenarios that lead to a crash or nearly a crash involving the vehicle. In existing works, combinatorial testing was used as one approach to address this question [431], [432].

The automatic emergency braking (AEB) function strongly contributes to the active safety properties of automotive vehicles. Proper functioning of AEB increases vehicle and road safety and further contributes to the acceptance of automated driving in general. Conversely, misbehavior of this function, e.g. by triggering the brake too late or

groundlessly, can cause severe road accidents with fatal consequences. It is hence key to test the AEB functionality extensively. Extending previous efforts, our case study focuses on this central and representative function of automated driving.

5.4.2 Underlying Works on Automated Driving Functions and Combinatorial Testing

The present work can also be seen as an extension of the application of the combinatorial testing cycle (see Fig. 5.5) to the domain of automated driving function testing where the following works have appeared in the literature. The work in [94] presents an algorithm that converts ontologies to input parameter models (IPMs) and further to combinatorial test suits. In [431] the authors discuss main challenges of testing automated and autonomous vehicles and propose to use CT based on automatically extracted IPMs from ontologies as a solution. The main contribution of [431] is the introduction of a general testing methodology for automated and autonomous vehicles, that uses the algorithm of [94] for mapping ontologies of the environment of the vehicle to IPMs for CT. In [433] the previously introduced algorithm [94] is used to deriving an IPM from an ontology, which parameter value ranges and representative values are based on the Euro NCAP protocol. Based on this IPM combinatorial testing and search-based testing are applied for testing an automatic emergency braking function. Thereafter, in [432] the developed ontology based CT methods are improved (CT_ONT2) and used in an industrial setting for testing autonomous driving functions. More specifically, the AEB function system from AVL was tested in a concrete case study based using an ontology-based representation of the domain and using CT for test case generation.

In [434] the focus was to use a genetic algorithm for test parameter optimization in order to efficiently derive critical driving scenarios for virtual testing. So far these efforts culminate in the study presented in [435], where search based testing, combinatorial testing and random testing have been empirically compared for AEB function testing. The study compares the scenarios generated from these methods, with respect to the quantity and quality of the crashes recorded during test execution. Thereby CT leads to the highest total number of crashes and notably was the only method that lead to all different types of crashes that where considered, this holds for strength $t = 3$ *and* $t = 2$. However, in terms of crash frequency, i.e. number of crashes per test scenario, CT was inferior to search based testing.

The revisited works above all realize or refine the automated testing methodology presented in [430]. The work presented below augments this methodology by adding the aspect of combinatorial fault localization to it.

5.4.3 Methodology

In the following paragraphs we first describe the interpretation of the considered testing problem as a combinatorial fault localization problem; second we describe the workflow of the testing procedure; and third the CT-FLA method used for test suite generation and analysis.

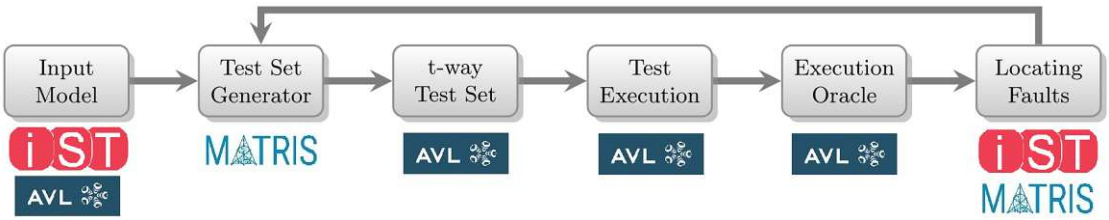


Figure 5.5: Visualization of the combinatorial testing cycle, indicating the working areas of the contributing organizations AVL List GmbH (AVL), Institute for Software Technology, Graz University of Technology (IST) and the MATRIS Research Group from SBA Research (MATRIS) in this case study.

Automated driving functions are tested on virtual driving environment platforms for their behaviour in different driving scenarios. See for example [436] for a review of simulation-based verification and validation methods. The execution of a scenario can be understood as a simulation of a real world traffic situation from the point of view of a vehicle - referred to as the *ego vehicle* in the following - that is controlled using automated driving functions, such as an AEB function. The aim of our work is to get a better understanding why certain scenarios simulated on the virtual driving platform result in a *crash*, while others do not. To that extent in the testing problem at hand, we can understand the AEB function as the SUT. However its input is not modeled directly, but instead the driving scenario that the virtual vehicle undergoes, or more precisely the scenario specification, is specified by means of an IPM that models the traffic situation.

For example, the speed and the type of the ego vehicle as well as the number of other road users, their position and their speed, is captured by the IPM and potential constraints defined between the parameters. A more detailed description of the used IPM will be given in Subsection 5.4.4. The individual parameter-values are entered automatically to a virtual driving platform so that the simulation can be performed. The individual driving scenarios thus represent a test case, which can be derived for example from the rows of a CA. In order to identify critical scenarios we rely the time-to-collision (TTC), a well-known time-based safety indicator, since several decades [437]. The TTC is defined for every point in time, as the time span left until two vehicles would collide, if no evasive action would be taken. In short, the lower the TTC value, the more critical the driving situation, see [438] and [439]. The TTC is collected throughout the whole simulation of a scenario and if it gets dangerously small, i.e. falls below a given threshold TTC_{crit}, at any point in time, the scenario is considered to be critical. A TTC equal or close to zero represents a crash. Finally, using the above interpretation, parameter settings that are likely responsible or essential for critical scenarios map to the concept of failure inducing t-way interactions (FITs) from CT, see e.g. [82] but also [130] and [83] where this notion is referred to as *failure causing schemas*, or *suspicious combinations* respectively. Table 5.13 summarizes the described interpretation of notions from automated driving function testing in terms of CT. In the following the terms *failure inducing t-way interaction*, abbreviated as FIT, and *crash inducing parameter setting* can be understood synonymous.

Virtual driving function testing	Combinatorial testing
AEB function	System under test (SUT)
Driving scenario specification	Input-parameter model (IPM)
Driving scenario	Test vector
Time-to-collision (TTC)	Testing oracle (oracle)
Driving scenario simulation	Test execution
Non-crash scenario (TTC$> TTC_{crit}$)	Passing test
Crash scenario (TTC$\leq TTC_{crit}$)	Failing test
Crash inducing parameter setting	Failure inducing t-way interaction (FIT)

Table 5.13: Mapping between concepts from virtual driving function testing and those of combinatorial testing.

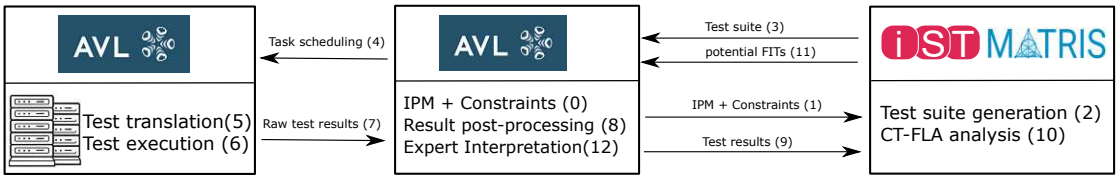


Figure 5.6: The structure of the CT workflow in the considered scenario, the numbers indicate the order in which the steps are performed. The schematics shows leftmost the computation unit from AVL, in the middle the development and testing unit of AVL, and rightmost the research groups of IST and MATRIS. The numbers (i) for $i = 0, \ldots, 12$ indicate the order in which the steps are performed.

The structural and infrastructural circumstances in the documented case study are as follows. Referring to the main phases of the (simplified) combinatorial testing cycle as depicted in Fig. 5.5, the phases are split between the contributing organizations AVL List GmbH (AVL), Institute of Software Technology at Graz University of Technology (IST) and the MATRIS Research Group from SBA Research (MATRIS) in the following way. AVL operates the virtual driving test platform, and the IPM for the driving scenarios was created jointly by AVL and IST. The test suite generation is performed by MATRIS based on this IPM. The t-way test sets are translated to executable tests using AVL's internally developed tool. Test execution is also performed by AVL, however the computationally costly simulations need to be performed on a powerful computing environment. In our case this is a server from AVL where the test scenarios are executed on the virtual driving test platform. The sever however is used by multiple in-house clients and requires to be scheduled in advance in order to perform computations, reserving a time slot and the required computation time. Once AVL receives the testing results to a test suite, they are post-processed and forwarded to MATRIS where the analysis of the results is performed. The transmission of test suites and the testing results between the two parties is not (yet) automated, as the small number of testing cycles did not demand for it so far. The structure of the workflow of the combinatorial testing cycle in the considered case study is depicted in Fig. 5.6.

The given structural and infrastructural circumstances entail certain hurdles in our joint undertaking. Particularly the fact that test generation needs to be performed separate from the test execution platform, and the fact that test execution requires booking of

computational resources timely in advance, required us to follow an adequate CT-FLA method. Prominent adaptive CT-FLA methods such as the one presented in [83] do rely on a frequent communication between test execution unit and the test generation unit. In order to keep the overhead introduced through test suite scheduling low, the used CT-FLA method should not depend on a frequent communication between the test generation unit and the test execution unit. However, application of a non-adaptive CT-FLA method is also not possible, since such are (currently) only applicable and available for a small number of FITs with a small strength, see e.g. [440] or [137] and references therein for descriptions of non-adaptive CT-FLA methods.

Hence we opted to realize an adaptive CT-FLA approach (Algorithm 17), that allows to have a low frequent communication between test execution and generation unit, which works as follows. The initial test suite is derived from a CA of strength t for the given IPM and respecting the constraints. The tests are executed and the execution oracle, which is realized through measuring the minimal occurring TTC in a scenario, categorizes them in failing tests (respectively crash scenarios) or passing tests (respectively non-crash scenarios). Based on the testing results we analyze the test suite and identify all t-way interactions that appear in at least one passing test. All remaining t-way interactions constitute the potentially failure inducing t-way interactions (potential FITs), also known as *suspicious combinations* as they are referred to in [130]. Based on this, the test suite for the next round is computed, which is derived from a CA of strength $t + 1$. Thereby we aim to enrich the generated CA with the additional *separation property*, that each potential FIT appears in a test case that covers no other potential FIT. In some cases this might not be possible for all potential FITs, for example due to given constraints in the IPM, or simply due to the sheer number of potential FITs (as is the case this case study). Therefore for each potential FIT we generate a test that *minimizes* the number other FITs covered by the test. Algorithm 17 gives a high level description of the realized CT-FLA approach.

5.4.4 Case Study

Following we give a description of the testing infrastructure, i.e. the virtual driving test platform, the considered AEB function, and the translation of test cases to executable scenarios. We further describe the IPM with constraints used to model the input to the virtual driving test platform as well as the considered testing oracles and the test execution environment.

System Under Test. In our case study we consider an AEB test procedure as defined by the European New Car Assessment Programme [441], also known as Euro NCAP, a well-established vehicle safety assessment organization.

Euro NCAP introduced a test protocol for AEB test scenario classes based on frequent accident situations. The set-up for these scenario classes is provided in detail, together with relevant parameters and value ranges, which allow to define concrete test scenarios that are executable on test site or in simulation.

Algorithm 17 CT-FLA Approach

Require: initial test strength t_0, maximal test strength t_{max} or testing budget
1: generate initial test suite $\mathcal{T}_t$ of strength $t \leftarrow t_0$
2: **while** $t \leq t_{max}$ and testing budget not exceeded **do**
3: execute tests in $\mathcal{T}_t \rightsquigarrow$ oracle results o_t
4: $potentialFITs \leftarrow$ all t-way interactions that *do not* appear in any passing tests
5: $\mathcal{T}_{t+1} \leftarrow TestSuiteGeneration(t, potentialFITs_t)$
6: $t \leftarrow t + 1$
7: **end while**
8: **return** $potentialFITs$

TestSuiteGeneration$(t, potentialFITs_t)$
9: $\mathcal{T}_{t+1} = \emptyset$
10: **for all** t-way interactions $\tau \in potentialFITs_t$ **do**
11: construct separating test r_τ for τ, containing a minimized number of other elements of $potentialFITs_t$
12: add test r_τ to $\mathcal{T}_{t+1}$
13: **end for**
14: extend $\mathcal{T}_{t+1}$ to a CA of strength $t + 1$
15: **return** $\mathcal{T}_{t+1}$

In order to translate parameter-value assignments to executable test scenarios, we first convert it into an XML format according to the standardized OpenScenario specification [442], an open file format for the description of dynamic contents for vehicle driving simulations. The scenario descriptions are finally input to the automated test execution and evaluation framework. Following, we briefly describe the setup of the framework and the comprised models enabling to run and evaluate the scenario simulations. As the main environment we use AVL's Model.Connect [443] co-simulation platform in order to integrate and connect required models. For executing the given test scenarios, the virtual driving environment Virtual Test Drive from Hexagon AB [444] is used. In addition the AVL VSM tool to represent realistic vehicle dynamics, an AEB function, and a control model to select the target object during scenario execution are implemented. These models and tools are linked in the platform, to enable the exchange of information during run-time. Further a crash reporting tool is connected to automatically evaluate the results related to the executed scenario criticality.

The AEB function is designed to automatically perform a braking manoeuvre upon detection of a potential collision. The investigated prototypical AEB function was developed in the course of internal research activities from AVL and is solely used for demonstration purpose. The system comprises two main components: the vision system and the brake control system. The first is based on a radar sensor model and the second comprises a target object selection algorithm and the AEB function. As soon as an object (e.g. a leading vehicle, or a pedestrian walking orthogonal to the traffic lane) enters the radar's cone-shaped field of view (see Fig. 5.7), the sensor (radar) will detect the object and identify its position and its relative speed compared to the ego vehicle. The radar in this setup is an ideal sensor, no influence of outside noise or other physical

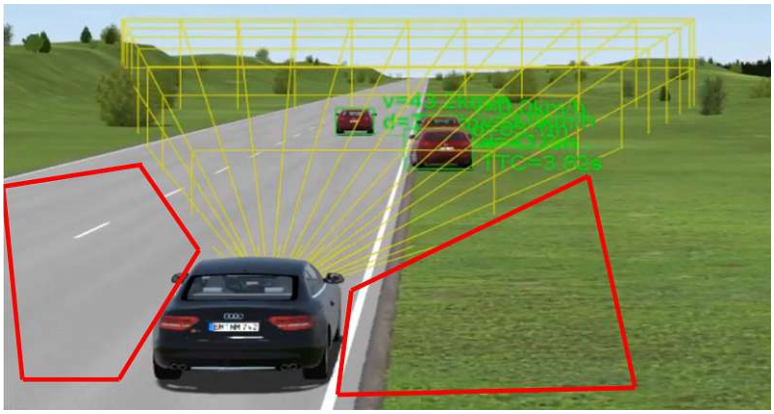


Figure 5.7: Excerpt from the visualization of a driving scenario, visualizing the Ego vehicle's cone-shaped radar field of view in yellow, and the blind area in the red.

phenomena will have any influence on detection. The radar will always detect an object within its cone-shaped detection area. The information from the radar is fed into the target object selection algorithm, which selects the target object based on criticality of its influence on the ego vehicle. The target object information is then forwarded to the AEB function where the minimum required braking time towards that object is calculated, and a velocity-dependent time safety margin is added. When the calculated TTC is equal to or goes below the bound $TTC_{critical}$, a braking maneuver is initiated.

Input Parameter Model. The IPM modeling the input to the virtual driving test platform was previously derived in [431] using methods presented in [94], and was already used in previous works [434] and [432]. We refer the interested reader to [431] for the details on the IPM derivation. The IPM consists of 39 parameters, with the following domain sizes

$$(3,3,31,3,5,1,6,4,12,12,12,10,14,12,12,12,10,14, \\ 31,4,3,20,9,3,3,31,4,3,20,9,3,3,31,4,3,20,9,3,3). \tag{5.3}$$

When sorting these in descending order, we obtain $(31^4, 20^3, 14^2, 12^6, 10^2, 9^3, 6, 5, 4^4, 3^{12}, 1)$, where we use an exponent notation, i.e. x^y means that there are y parameters that can take x values. Additionally there are 42 constraints formulated on the parameters of the IPM, which are expressed as quantifier-free Boolean linear arithmetic formulas. Each constraint uses between two and six parameters of the IPM. When deriving the set of minimal forbidden s-way interactions, that are implicitly forbidden by these constraints, we obtain a list of 1225 s-way interactions of lengths $1 \leq s \leq 4$.

Oracle. As mentioned in Section 5.4.3 the testing oracle is realized through a method that distinguishes critical from non-critical scenarios. This method is based on the time-to-collision measured throughout a scenario. In this case study we are interested in identifying the most critical scenarios, i.e. collisions, which represent AEB failure. This is realized by the oracle function CRS (for CRash Scenario), which assesses any scenario with a TTC equal to or almost zero as a *crash*.

Test suite	Strength t	# Scenarios # Tests	CRS oracle	potential 2-FITs	potential 3-FITs
$\mathcal{T}_2$	2	994	178	5680	395804
$\mathcal{T}_3$	3	39061	7928	0	125789

Table 5.14: Overview of the results after two rounds of testing of the virtual driving test platform following Algorithm 17.

Execution Environment. The test execution in form of driving scenario co-simulation is using two locally connected machines. The first is an Intel Xeon Silver 4214 with a CPU clocked at 2.2GHz and an NVIDIA Quadro RTX 4000 graphics card (running Windows 10 as operating system), which runs the co-simulation platform Model.CONNECT, simulates the vehicle dynamic using AVL VSM, and the AEB function, as well as performing pre-processing of the test cases. The second machine has an Intel Core i7-9700K CPU clocked at 3.6GHz and an NVIDIA Geforce RTX 2070 SUPER graphics card (running Linux Ubuntu 18.04 as operating system) and is used for the driving environment and sensor simulation using Virtual Test Drive by Hexagon AB [444].

In this execution environment, the simulation of a single scenario took on average 38 seconds including pre-processing. The exact execution time depends on the scenario outcome, e.g. if the ego vehicle brakes and stops, the scenario simulation is considered completed and is terminated, leading to a shorter execution time.

Combinatorial Test Suites. Following the CT-FLA approach given in Algorithm 17, the initial test strength was set to be two, $t_0 = 2$. There was no upper bound for the strength specified, i.e. $t_{max} = 39$ as there are 39 parameters, for the reason that the limiting factor was the testing budget, given by the available time on the test execution unit, which main capacities are required for other projects. The initial test suite $\mathcal{T}_2$ was based on a CA of strength two respecting the given constraints, having 994 rows, i.e. yielding that many driving scenarios. The test suite $\mathcal{T}_3$ for the second iteration of the work-flow was based on a CA of strength three, optimizing separation of all potential 2-way FITs derived from the first iteration; it consists of 39061 rows. There where not enough resources, time and computing capacity, in order to perform a third round of testing.

5.4.5 Results

Following we detail the results of the simulation of the driving scenarios performed in the course of this case study, reporting the number of potential FITs identified in the individual testing rounds. Thereafter we evaluate and discuss these results, where we also quantify over the appearance of parameters involved in these FITs. We further give some impressions from the visualization of the potential FITs that are most likely responsible (from a CT-FLA point of view) for crashes in the scenarios they appear in, and report on their assessment through a domain expert.

An overview of the simulation results is given in Table 5.14. The simulation of the 994 scenarios derived from the strength $t = 2$ test suite $\mathcal{T}_2$, resulted in 178 crash scenarios, respectively failing tests. In the parameter specifications of these scenarios we identified 5680 potential 2-way FITs, respectively potential 2-way crash inducing interactions. For the sake of completeness we also report that based on this test suite we identified 395804 potential 3-way FITs, respectively 3-way crash inducing interactions. However, that this number has little to no informative value becomes clear when considering, that the test suite $\mathcal{T}_2$ misses to cover 5659100 different 3-way interactions that respect the given constraints.

According to Algorithm 17 we generated the test suite $\mathcal{T}_3$, by generating tests aiming for separation of the 5680 potential 2-way FITs, i.e. covering each if them in an individual test case. This resulted in 192 tests, where 5417 of the potential 2-way FITs appeared with no other potential 2-way FIT in a test, and the remaining 263 where covered in 64 additional tests. We note again, that a full separation was not possible due to the high number of failing tests, almost 18% of tests, the high number of potential 2-way FITs and the given constraints. These tests for separating the potential 2-FITs where then extended to a CA of strength $t = 3$, which in sum constitute the test suite $\mathcal{T}_3$. From the 39061 scenarios 7928 resulted in a crash, which are more than 20%. Notably, based on these results it was possible to rule out that any 2-way interaction is crash inducing, as each appears in at least one of the passing tests of $\mathcal{T}_2 \cup \mathcal{T}_3$. However, a very high number 125789 of potential 3-way FITs remains. Although we drastically reduced the number of potential 3-way FITs compared the first round of testing, this number remains too high, rendering another test set of separating tests not executable due to test budget constraints. Recall that the execution consumes about 38 seconds, thus a test suite with over 100000 tests would roughly take one and a half month to execute. Even more so, testing with a test suite based on a CA of strength $t = 4$ is not possible, as such a test suite would contain roughly one million of scenarios – recall that the four largest parameter domains contain 31 values, which causes a strength 4 CA to have at least $31^4 = 923521$ rows.

Given the time constraints, it was also not possible to perform another round of testing. This is why we focused on a more detailed analysis of the potential 3-way interactions obtained so far.

Further Analysis Based on Potential 3-way FITs

Investigating the results obtained from the the second round of testing closer, we can make the following qualitative differentiation. For the 125789 potentially 3-way FITs, we count for each in how many failing test it appears. This leads to the distribution given in Table 5.15. These numbers show that the largest fraction of potential 3-way FITs appear only in a small number of failing tests, while on the other hand there are three potential 3-way FITs that are outstanding from all the others, appearing in 60, 53, and 46 failing tests respectively. This difference by itself suggests that these potentially 3-way FITs are of special interest. For us this is reason enough to investigate them even closer, documented in the following subsection.

# pot. 3-FITs	1	1	1	4	8	19	117	387	1634	6500	25872	91245
# failing tests they appear	60	53	46	9	8	7	6	5	4	3	2	1

Table 5.15: The first row shows the quantities of potential 3-way FITs, appearing in the number of failing tests, i.e. crash scenarios, given in the second row.

Aside from the frequencies of appearances of potential 3-way FITs in failing tests, we also considered the frequencies of parameters appearing in potential 3-way FITs, i.e. for each parameter we count how often it appears (with any value) in the set of potential 3-way FITs. Referring to the IPM domain sizes given in equation (5.4), the third parameter has the highest number of appearances, contributing to 62557 potential 3-way FITs. Dividing by this maximal value, we get the following vector of frequencies normalized to 62557:

$$\begin{aligned}(&0.02, 0.01, \mathbf{1.00}, 0.00, 0.00, 0.00, 0.00, 0.01, 0.08, 0.08, 0.07, 0.03, 0.12,\\ &0.10, 0.10, 0.10, 0.05, 0.13, \mathbf{0.99}, 0.00, 0.00, \textit{0.33}, 0.01, 0.00, 0.00, \mathbf{0.99},\\ &0.00, 0.00, \textit{0.36}, 0.02, 0.00, 0.00, \mathbf{1.00}, 0.00, 0.00, \textit{0.40}, 0.02, 0.00, 0.00)\end{aligned} \tag{5.4}$$

We see that the parameters with large domain sizes coincide with the parameters that have the highest frequency of appearances in potential 3-way FITs, where it is worth to mention that these frequencies ≥ 0.99 (displayed **bold** in (5.5)) are roughly three times larger than the frequencies of a second group of parameters appearing ≥ 0.33 (displayed *italics* in (5.5)), and larger by almost one or multiple orders of magnitude compared to the other frequencies. A threat to validity for this presented measure is that the applied test suites are CAs, i.e. all 2-way and 3-way interactions are covered, hence it may penalize some parameters based on their large domain size. Assume there was a 2-way FIT between two Boolean parameters, in a CA of strength $t = 3$ a parameter with v different values appears in at least v different failing tests; for a v very large compared to the other domain sizes, this can cause a skew distribution. Therefore we also consider for each parameter how often it appears in the set of potential 3-way FITs relative to its domain size. Again the third parameter achieves the highest measure, where each of the 31 values appears on average $2017.97 = 62557/31$ in potential 3-way FITs. Dividing by this maximal value, we get the following vector of frequencies per parameter value normalized to 2017.97 appearances (again referring to the IPM domain sizes given in equation (5.4)):

$$\begin{aligned}(&0.18, 0.06, \mathbf{1.0}, 0.00, 0.01, 0.00, 0.00, 0.08, 0.20, 0.21, 0.19, 0.10, 0.28,\\ &0.26, 0.27, 0.26, 0.14, 0.29, \mathbf{0.99}, 0.00, 0.00, \textit{0.52}, 0.04, 0.00, 0.00, \mathbf{0.99},\\ &0.00, 0.00, \textit{0.56}, 0.05, 0.00, 0.00, \mathbf{1.00}, 0.02, 0.04, \textit{0.61}, 0.07, 0.00, 0.04)\end{aligned} \tag{5.5}$$

The frequencies per parameter given in equation (5.6) exhibit a very similar pattern to the one in equation (5.5). In fact it is the same set of parameters that achieve a frequency greater or equal to 0.99. When we look at the input-values that are modeled by these

parameters, we find that all four of them model the start speed of road users:

```
EgoVehicleStartSpeed
Object1StartSpeed
Object2StartSpeed
Object3StartSpeed
```
(5.6)

a result that seems reasonable when considering that our investigations pertain traffic and crash scenarios.

Remark 5.4.1 (Comments on IPM Reduction) *Finally we also counted how often each parameter-value appears in the potential* 3*-way FITs. Clearly, the previously identified parameters that appear with a high frequency in the potential* 3*-way FITs, also have values with a high presence in those. But more interestingly is the insight that there are also parameter-values that do not appear in any of the* 125789 *potential* 3*-way FITs. We therefore refer to them as* 3-way safe values*. The* 3*-way safe values contain potential for reducing the IPM for future scenario generation, by removing any value from it that does not appear in any of the potential* 3*-way FITs. Such an IPM reduction can allow to derive critical scenarios more efficiently in future testing cycles, which can be valuable for practitioners.*

5.4.6 Threats to Validity

There are some clear threats to validity regarding the results of this case study. First and foremost, we applied our CT-FLA approach in a very specific testing set-up – the virtual driving test platform at AVL – which is based on the framework AVL Model.Connect [443] as co-simulation platform which assembles the three parts (1) AVL VSM for vehicle dynamic simulation, (2) VTD from VIRES [444] as a virtual driving environment and (3) the AEB function developed by AVL for internal research activities. Albeit, this is an important first step in applying CT-FLA methods for virtual testing of automated driving functions, this certainly does not allow for generalization to arbitrary virtual driving test platforms and functions. Further, it is unfortunate, that we could not perform additional rounds of combinatorial testing (see also Subsection 5.4.8). To elaborate, it is not clear in how far identified (potential) FITs are responsible for crash scenarios in the virtual driving test platform. In our analysis of the testing results obtained so far we therefore had to rely on relative frequencies of potential FITs, instead of *precisely* identified FITs in terms of CT-FLA. Nevertheless, in order to get a first assessment of the usefulness of the obtained results in form of the potential 3-way FITs, a domain expert from AVL conducted a review.

5.4.7 Review of a Domain Expert

In order to get a better understanding of the results obtained from Algorithm 17, and to get an insight of the meaning of CT-FLA for automated driving function testing, we

asked a domain expert from the field of autonomous driving testing for an opinion on our results. Since it is infeasible to give feedback on each of the 125789 potential 3-way FITs individually, the domain expert considered the three potential 3-way FITs that most likely cause crash scenarios, i.e. the ones that appear in 60, 53 respectively 46 failing tests, see Table 5.15. Listing these in the given order, in the format $(parameter_a = value_a, parameter_b = value_b, parameter_c = value_c)$ they reveal as:

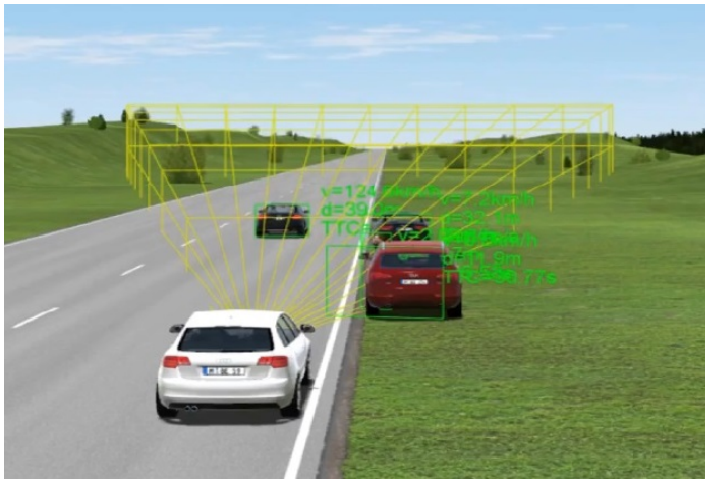
$(EgoVehicleOffset = 2, Pedestrain1StartSpeed = 0.56, Pedestrain1Offset = 4)$,
$(EgoVehicleOffset = 2, Pedestrain1StartSpeed = 0.28, Pedestrain1Offset = 4)$,
$(EgoVehicleOffset = 2, Pedestrain1StartSpeed = 0.56, Pedestrain1Offset = 3)$.

First, we note that interestingly the parameters appearing in these potential 3-way FITs are different from the four parameters given in equation (5.7) that appear with highest frequencies in the potential 3-way FITs. *This fact gives a valuable first indication for the meaningfulness of CT-FLA in the domain of virtual validation and verification of automated driving functions.* We can see that the results are different from those of the simple frequency analysis of the values per parameter.

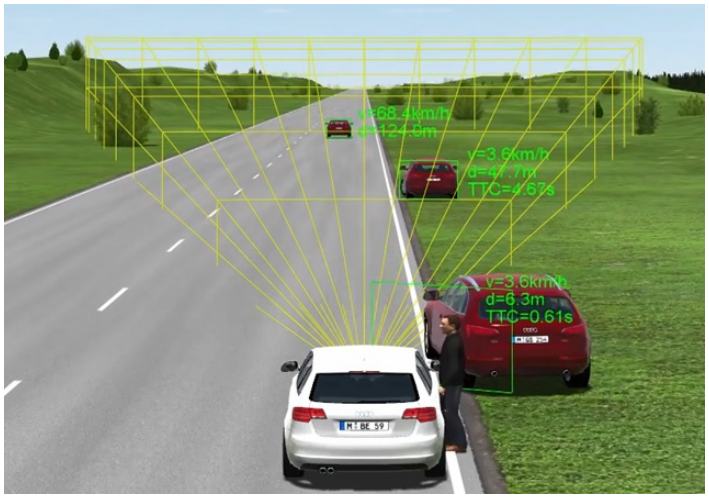
For the review, for each of these 3-way interactions, we arbitrarily selected five (failing) tests where they appear in. Recall, since they are potential FITs, all tests that cover the 3-way interactions must be failing. The corresponding driving scenarios, 15 in total, have been simulated and evaluated visually by the domain expert. In Figure 5.8 we show excerpts from one of the five scenarios for each of the three 3-way interactions. The assessment of the domain expert [445] after evaluating the 15 scenarios can be summarized as follows. All scenarios provided, covering the identified 3-way interactions have the same root cause for the crash: The vehicle is unable to detect an object, or it loses an object during operation, thus not reacting on time or not even reacting to an imminent collision. Further, a large majority of crashes in the scenarios are side-wards collisions with the ego vehicle.

This lead the domain expert to give the following explanation of these outcomes [445]: Since an ideal object sensor is used in the simulations, there are no detection issues if an object is within the radar field of view (see the yellow cone-shape in Fig. 5.7), while objects outside this area, the red area in Fig. 5.7, will not be detected. Scenario specifications, i.e. parameter values that impact the position (offset) of the ego vehicle and relevant objects to the side of the ego vehicle outside of the cone-shaped detection area will contribute more towards outcomes with a collision.

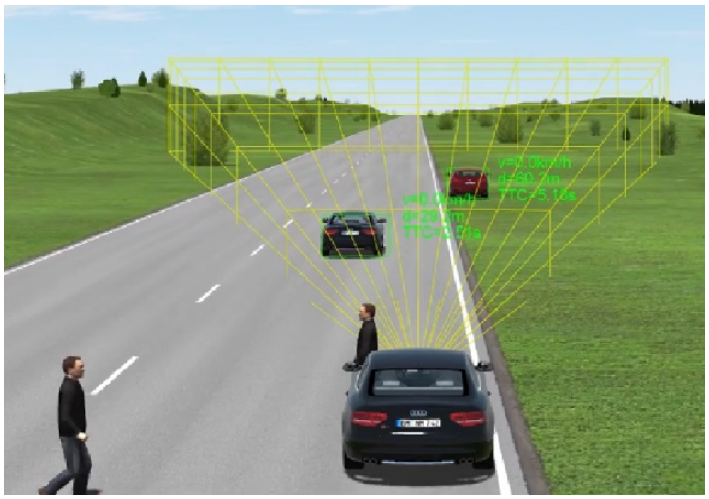
Summarizing this case study, we have realized a CT-FLA cycle, in a remote setup, using an approach based on rounds of combinatorial testing. The results are not concrete in terms of identified actual FITs, only yielding a list of potential FITs and demand for further testing and combinatorial fault localization. Ideally this will be done with an improved communication between the test execution and the analysis and test generation unit. Nevertheless, our preliminary results about specifications in crash scenarios seem to have brought valuable input to a domain expert. This suggests, that the information



(EgoVehicleOffset=2,Pedestrain1StartSpeed=0.56,Pedestrain1Offset=4)



(EgoVehicleOffset=2,Pedestrain1StartSpeed=0.28,Pedestrain1Offset=4)



(EgoVehicleOffset=2,Pedestrain1StartSpeed=0.56,Pedestrain1Offset=3)

Figure 5.8: Excerpts from driving scenarios derived from tests that cover the three potential 3-way FITs that appear in the highest number of failing tests, i.e. crash scenarios. The corresponding potential 3-way FITs are given below the images.

about t-way interactions that have impact on a specific outcome of an SUT is indeed a valuable information. In the next section, we will deepen this thought and investigate the notion of t-way interactions in a different context.

Outlook: CT-FLA for AEB Function Testing The application of other CT-FLA methods, e.g. ones known in the literature [130], [83], to the same SUT will be of interest in order to compare the obtained results. However, in order to do so it will be required to enhance the test execution setup to allow for a better communication between the test execution unit and the test generation unit, see also the discussion in Section 5.4.3. This however will require changes in the infrastructure underlying the setup and is therefore kept for onward future work. As part of our immediate future work we will consider multiple oracle functions that differentiate between different types of crashes. Further, we want to execute more tests coming from CT. In particular, as testing with a CA of strength $t = 4$ is infeasible due to test suite size and execution time, we plan to follow a different approach to further reduce the number of potential 3-way FITs, or to increase certainty that specific 3-way interactions lead to crash scenarios. Finally, it is possible to consider also other automated driving functions in the experimental setup.

5.4.8 Epilogue – Analysis of an Individual Crash Scenario

The results of the CT-FLA approach presented above merely led to the identification of *potential* FITs, i.e., to *potential crash inducing parameter settings*. The testing budgets (computation time on the test execution unit, respectively financial budgets and the project duration) were almost exhausted and did not allow further concretization of the potential FITs at a large scale. However, in an epilogue to the efforts documented above, the contributing organizations (see Figure 5.5) aimed at identifying a more concrete result regarding the existence of crash inducing parameter setting – even if it was just a single such setting. Any findings could be helpful in understanding better the (potential) results of CT-FLA methods in the domain of automated driving function testing, and could further aid the set up of any future case study. To that extent and in order to assess its potential, the idea was to mimic an adaptive CT-FLA method that relies on a higher frequent communication between the test execution and the test generation unit, recall Figure 5.6.

We selected a test that leads to a crash scenario and contains only a single potential FIT according to our analysis of potential 3-way FITs, and examined this test in order to identify a FIT as follows. Based on the selected test we derive new tests by changing each individual parameter-value assignment to all possible different assignments of values for the parameter. This amounts to an examination of the 1-neighborhood in terms of the Hamming distance. More detailed, let us denote the selected test $r = (r_1, \ldots, r_{39}) \in \bigtimes_{i=1}^{39} D_i$ as an element of the Cartesian product of the 39 parameter domains of sizes as described in relation (5.4). Then we derive new tests as follows: for all $i \in \{1, \ldots, 39\}$, and for all $u \in D_i \setminus \{r_i\}$ derive a new test $(r_1, \ldots, r_{i-1}, u, r_{i+1}, \ldots, r_{39})$. This method is also known as *one factor one time* (OFOT) method in the literature [82,

Sec. 3.5.1]. Since the derived tests need also to respect the constraints, this results in at most $\sum_{i=1}^{39} |D_i| - 1 = 357$ derived tests. For the particular selected test, the OFOT method yields 292 derived tests, representing its 1-neighbourhood that respects the given constraints. The relatively high number of derived tests that violate some constraints is explained by the fact that in the selected scenario there are only two cars involved, therefore the parameters modeling the third vehicle in the scenario, e.g. its start-speed, must be set to null.

Execution of the 292 derived OFOT tests leads to crashes in 255 scenarios and to 37 non-crash scenarios respectively, which are of especial interest in this analysis. The crash of a derived OFOT test does not provide a lot of information when considered in isolation: a potentially present FIT has not been affected by the value change, or it was affected but the value change led to the presence of another FIT in the derived test. However, the non-crash scenarios allow to infer much more, even when considered in isolation: the specific parameter-value change alters the result from crash to non-crash. This is a strong indicator that the original parameter-value assignment is part of a FIT, which once removed due to the new parameter-value assignment leads to a non-crash scenario. In addition, this means that this parameter-value assignment must be involved in all FITs that are present in the selected test – otherwise there would remain FITs in the derived OFOT test leading to a crash. Examining all derived OFOT tests yields the following results:

- In total 26 of the 39 parameters can be re-assigned values while respecting the given constraints.
- For each of these 26 parameters there is at least one re-assigned value that leads to a crash scenario.
- For 6 of the parameters there is at least one value that leads to a non-crash scenario.

The fact that for all 26 parameters that are subject to change there exist values that lead to crashes, may indicate that the values need to be changed significantly so to alter the outcome of the test execution. The 37 non-crash scenarios resulting from the OFOT method are due to value changes distributing to 6 parameters. This indicates that these contribute to a FIT, in particular a $\geq$6-way FIT, i.e. a FIT composed of at least six parameter-value pairs (analog to Definition 1.3.8). The reason why these results do not identify this 6-way interaction as a minimal failure inducing interaction (recall Definition 1.3.16), but only show the presence of a $\geq$6-way FIT are as follows. First and foremost, there were 13 parameters that cannot be altered due to constraints, leaving the theoretical possibility for their specification being necessary for the scenario to result in a crash. Further, it is theoretically possible that there are other parameter-value assignments in the selected test which cause it to crash, however, any value change of these may introduce another FIT in the derived test – an option that is more likely when there is a high percentage of failing tests.

To conclude this proof of concept effort, we performed a visual inspection of the crash scenario corresponding to the original selected test, together with 6 non-crash scenarios, each corresponding to a value re-assignment of one of the parameters of the $\geq$6-way FIT. This makes visible for each of the six parameters why the alternation of the values in the original test case yields a non-crash scenario. Using the terminology of the previous Section 5.3, in particular the use of CT-FLA methods to derive counter factual explanations for the output of AI systems as suggested in [272], we may understand each of the six non-crash scenarios as a counter factual explanation of the likely crash inducing $\geq$6-way FIT – a combination of parameter-value assignments that might be tedious to identify manually.

In retrospect and in terms of the failure reports reviewed in the introduction 1.2.2, we have identified a combination of six conditions, respectively parameter settings or parameter-value assignments that (seemingly) is the reason for the resulting crash of the scenarios these conditions are present. This finding is secured by the examination of additional tests that show that any change of conditions outside of the identified ones does still result in a crash scenario, while by appropriate changes of the identified six conditions it is possible to obtain a non-crash scenario.

5.5 Summary

In this chapter we presented two case studies where CT-FLA methods have been applied in new application domains and we have contemplated on CT-FLA methods for XAI with regards to properties of explanations. Common to the presented works is that they concern the extension of the applicability of CT-FLA methods, whether it is by pioneering novel application domains, as for hardware Trojan testing in Section 5.2 and automated driving function testing in Section 5.4, by contributing a new method for fast localization of single failure inducing t-way interactions, as in Algorithm 16 in Section 5.2, or by posing research questions that may point the way for and aide the development of future CT-FLA methods for XAI.

In particular, in our case study where we applied CT-FLA methods for HT localization, we have demonstrated that combinatorial testing can provide the mathematical guarantees to locate HTs. This was exemplified for HTs with trigger patterns of length up to $\ell = 8$, which have been inserted in a circuit that implements the AES symmetric-key encryption algorithm with 128 bits key length. *Although in some cases random arrays appear to have similar capabilities to the ones derived from combinatorial testing they do not provide the guarantees provided by combinatorial test sets.* Moreover, the run-time comparison of Algorithms 14 and 16 highlights that Algorithm 16 is faster by several orders or magnitude. Our experiments demonstrate that the combinatorial test sets CTℓ can locate HTs of length up to ℓ using the efficient location procedure of Algorithm 16. Further, by theory (see Remark 5.2.1 and Proposition 5.2.8) we are guaranteed that this holds for *all* HTs of length up to ℓ.

Potential Impacts. The *location* of HTs can have certain impacts and enable several additional options depending on the scenario. This includes, for example, restricted usage of the infected circuit (e.g. with a reduced key space) in case the identified HT is known to be the only malicious component. Knowledge about the operation of the HT may even be used against the attacker, e.g. by leaking information on purpose. In general, the *location* of HTs can be very valuable, as it allows for understanding the purpose of an attack and thus can help to gain insights about the attacker's intention and capabilities. In a different scenario, the identification of combinational HTs might enhance the analysis of the infected circuit, especially when logic testing is used in combination with other testing techniques, like side-channel analysis [375], since the identified HT can be excited or completely avoided on purpose.

Further, in the exploration of CT-FLA methods for XAI, we have proposed four properties of XAI in the context of similar notions given in the literature and commented or corrected them where appropriate. Thereafter, we presented a categorization of CT-FLA methods interpreted as an approach to XAI, revealing them, as applicable to black-box, decision systems for generating explanations of decision outputs rather than for decision processes. Considering CT-FLA from the view point of the formulated properties of XAI led to the formulation of research questions that can guide future developments of CT-FLA methods for XAI.

Finally, in our second case study we have demonstrated that CT-FLA methods are applicable to screen parameter settings to identify (potential) crash inducing value combinations in a virtual driving function test platform at AVL List GmbH. In our study we were able to drastically reduce the number of potential crash inducing t-way interactions: our results show that there are no such 2-way interactions, and we reduced the potential crash inducing 3-way interactions by two thirds. Further, we have identified a set of 3-way safe values, i.e. parameter-values that certainly do not appear in any crash inducing 3-way interaction, which bears the potential for an IPM reduction. While it was not possible to identify t-way interactions that certainly lead to a crash at a large scale, we were able to identify three very likely crash inducing 3-way interactions, which can serve as valuable input to test engineers. A first review by a domain expert indicates that the three identified 3-way interactions lead to scenarios that result in similar types of crashes. In an epilog we presented the inspection of a single crash scenario, which lead to the identification of a $\geq$6-way FIT that very likely causes driving scenarios to result in a crash whenever it is present. Additionally derived test scenarios may be understood as counter factual explanation for this, underpinning that the $\geq$6-way FIT was correctly identified as such. Although this analysis was conducted only for a single original failing test (respectively crash scenario), in this specific case study setup, it promises successful future applications of CT-FLA methods in this domain.

CHAPTER 6

Conclusion and Outlook

6.1 Advancement of the Combinatorial Testing Cycle

In the following paragraphs we summarize how the contributions underlying this thesis have advanced the CT cycle, respectively how they have extended its applicability to new domains, in reference to the strategy formulated in Section 1.5. In particular, we outline the contributions to input modeling, test set generation and combinatorial fault localization – those parts of a testing cycle that are influenced by combinatorial testing methods. We illustrate this by listing the contributions at the respective steps of the combinatorial testing cycle, see Figure 6.1.

To describe this more thoroughly, the input modeling has been advanced as follows:

- Combinatorial testing methods have been applied for testing communication protocols in smart buildings, together with an industry partner in a feasibility study. As a proof of concept, together with the industry partner, we developed an IPM for the harmonization tool of the APOGEE Insight® workstation which acts as a BACnet client and generated a 2-way test set which got deployed [145].

- The input to large language models has been modeled such that combinatorial testing methods can be applied. The proposed approach to applying CT for testing LLMs has been demonstrated as proof of concept in an experimental evaluation [147].

- Combinatorial testing fault localization approaches for explainable artificial intelligence have been explored. They are applicable when the AI system's input is modeled via an IPM. Combinatorial methods for explainable AI have been critically reviewed through the lens provided by the (desired) properties of explainable AI. This led to research questions that ask to be answered in the future. They have been

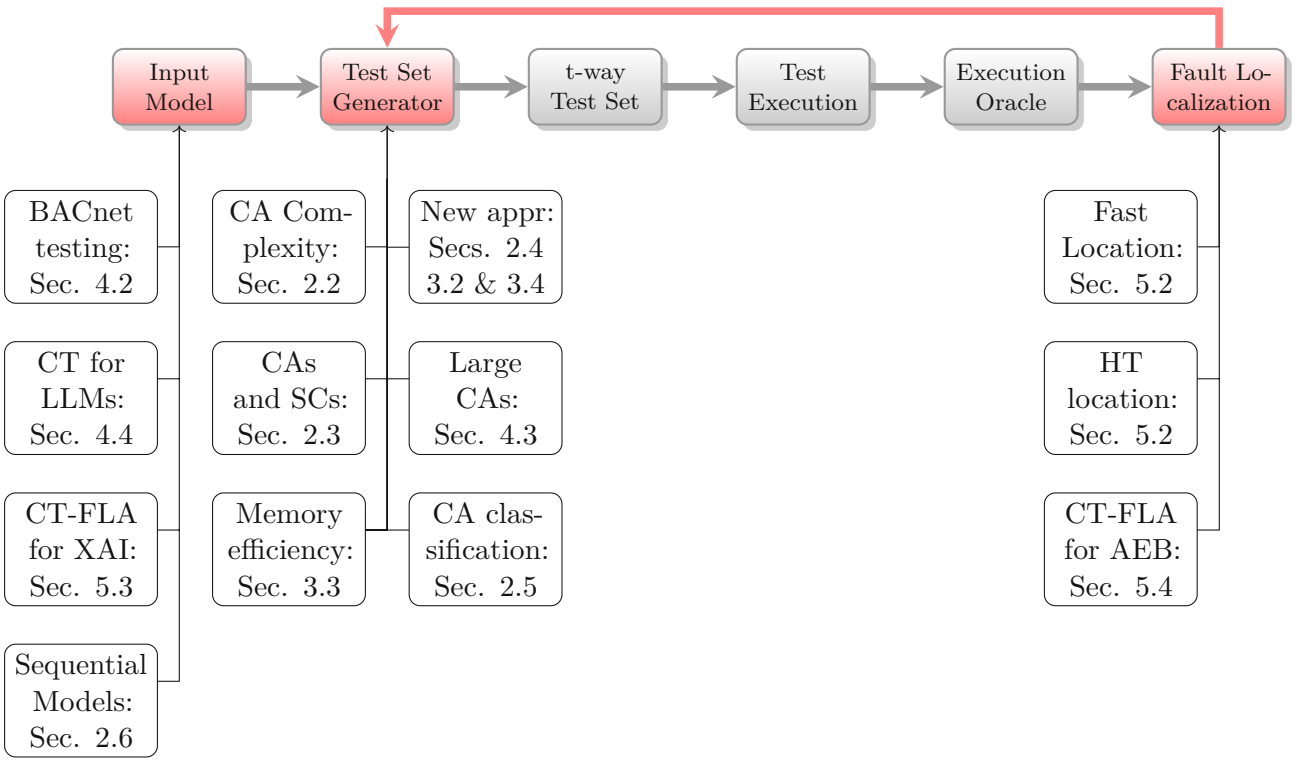


Figure 6.1: Simplified testing process with the CT-affected parts in red, and the contributions of this thesis mapped to the specific parts.

formulated and, where appropriate, possible solutions were pointed out. Answering these questions may further improve and establish combinatorial methods in the relatively novel application domain of explainable AI, [148].

- For SUTs with sequential dependencies amongst their inputs we have introduced an automata-theoretic product operator that reflects certain needs of sequential CT. Thereby, we have left the prevalent approach to sequential CT by means of *arrays* and considered a more flexible approach using test sets based on sequences or families of sequences. The introduced notion is of special help when creating sequential combinatorial test sets for SUTs that are modeled by means of a finite automaton. The need of sequential CT of generating a *small* set of *short* sequences is reflected by the introduced introduced reduced shuffle product operator [140].

The state of the art of optimal covering array generation and algorithmic techniques for covering array generation, respectively combinatorial test set generation has been advanced as follows:

- The optimal covering array generation problem – and related problems – were formally formulated as computational complexity problems and we showed some basic complexity properties. A rigorous review of known complexity results for problems related to optimal CA generation has been presented, with the aim of clarification as well as to avoid incorrect interpretations in the existing and future literature [66].
- The study of the optimized covering array generation problem, in particular its mapping to the minimal set cover problem, was motivated by outlining the connection to combinatorial software testing. Further, algorithms for optimized covering arrays via set covers have also been applied to the generation of variable strength covering arrays, and were compared to algorithms using exact methods, as part of the revision process of [144].
- The usability of an existing greedy method for CA generation has been extended by reducing the resource consumption used for generation. By making use of a combinatorial construction, the memory usage for incidence structures occurring in implementations of CA generation algorithms was reduced [138].
- Novel approaches to optimized CA generation were explored:

  An algebraic formalism capable of capturing the generation problem of finding a $\mathsf{CA}(N; t, k, v)$ for arbitrary parameters t, k and v was developed, using methods arising from commutative algebra and symbolic computation [67].

  A primal application of artificial neural networks for optimized covering array generation was presented, where we relied on previously investigated connections between optimized CA generation and combinatorial optimization problems ([144]). An experimental evaluation shows that the developed algorithms based on artificial neural networks can be used to generate CAs with larger alphabets, heterogeneous alphabets (mixed-level CAs), and CAs with constraints [141, 142].

  A hybrid heuristic combined the heuristic In-Parameter Order (IPO) algorithm with a quantum-inspired metaheuristic for CA generation. The resulting IPO-Q algorithm improves upon the (classical) IPOG-F by finding CAs with less rows for numerous instances [143].
- The applicability of combinatorial testing in industrial settings has been expanded together with the industry partner Adobe by testing their analytics product. To the best of my knowledge, we constructed the largest CAs used in industrial applications and deployed them for testing an SUT modeled with over 2000 input-parameters [146].
- The notion of balanced covering arrays was introduced and used to dissect classes of CAs through a classification of non-equivalent balanced CAs. Classes of covering arrays can thus be represented as nested classes of balanced covering arrays which exhibit different balance properties [139].

Combinatorial testing fault localization approaches have been advanced and used in pioneering applications:

- A non-adaptive combinatorial fault localization method for identifying single failure-inducing t-way interactions which does not rely on full enumeration of t-way interactions was developed [68].
- This method has been applied for locating hardware Trojans in a cryptographic circuit implementing the AES symmetric-key encryption algorithm with 128 bits key length. In a case study, the Trojans were located by means of identifying the Trojans' trigger pattern (up to length eight) in the binary input vectors of length 128 [68].
- Finally, in a pioneering case study with an industry partner, combinatorial testing fault localization was used to screen parameter settings that lead to critical driving scenarios in a virtual verification and validation framework used for automatic emergency braking (AEB) function testing [149].

6.2 Outlook

To conclude this thesis, we briefly outline future challenges of CT and, more generally, the application of combinatorial design theory for software testing and beyond.

1. *Non-deterministic systems:* One of the overarching assumptions underlying the CT methods discussed in this thesis – and also a large majority of the methods in existing literature – is the determinism of the considered SUTs. The extension of CT to non-deterministic systems will be important for the future applicability of CT, not only, but certainly emphasized by the increasing use of non-deterministic building blocks (such as artificial neural networks) in software systems. If CT methods do not keep up with these trends in software development, significant applications may remain inaccessible. However, combinatorial design theory offers a plurality of possibilities and notions that can be applied for non-deterministic systems. In particular, methods from Design of Experiments [446], [55] are promising and might be useful for the development of CT for non-deterministic SUTs. Existing work also shows that testing based on locating arrays can be combined with statistical analysis [447]. Finally, CT based on covering arrays of higher index may be a simple, but important first step in the development of CT for non-deterministic SUTs.
2. *Sequential CT:* It will be of interest to investigate a unified notion of t-way sequences that is general enough to be applied to a wide range of event-driven (software) systems, as well as to capture existing notions in sequential CT. Further, the author believes that a contextualization with notions of automata theory and its applications in software testing will become necessary. This pertains in particular to

automata learning [448], [449]. It is worthwhile to consider for example the relation of t-way sequences to *distinguishing sequences* [450] and *unique input output sequences* [451]. In addition, towards improving the applicability of sequential CT, empirical investigations of failure reports of event-driven systems are important, analogous to the studies of Kuhn et al. regarding combinational failure triggers in the early 2000s as summarized in the motivation (Section 1.2). Finally, efficient algorithms for constructing sequential combinatorial test sets need to be implemented in usable tools to make the respective notion accessible for applicants.

3. *CT-FLA Tools:* Tools are already existing which implement *adaptive CT-FLA* methods [130], [452]. Other promising approaches [83] to adaptive CT-FLA are about to be realized in available tools. However, there is a complete absence of tools realizing the algorithms or constructions that represent the state of the art of *non-adaptive CT-FLA* methods. Making CT-FLA methods accessible through available and usable tools is a major challenge.

4. *CT-FLA cycle:* An extension of the feedback cycle appearing in the CT process from combinatorial fault localization to the input parameter modeling stage should be considered. First steps in that regard have been made, e.g., in [83] where identified FITs are interpreted as constraints of the SUT in order to avoid them in future test cases. Another challenge demanding such a feedback step is the identification of "safe values" in practical applications (see also Remark 5.4.1). Safe values are a concept underlying several CT-FLA methods, but it can be difficult to identify them in practical applications.

5. *CT as part of SDLC:* In order to make CT methods available for applicants they need to be adapted to the contemporary procedures in software development and engineering. The author believes that one way to do so is to explore and extend connections between CT and individual steps of the software development life cycle. For example, combinatorial constructions [93] can help to unify combinatorial coverage with software architecture and testing methods. Further, it will be of interest to combine CT-FLA methods with regression testing, including the identification of critical (failing) tests (early work exists [82]), as well as to combine CT with model-based testing [453].

6. *Balanced CAs:* Finally, the author believes that the notion of balanced CAs (see Section 2.5) or related notions bear combinatorial properties that may turn out to be beneficial in addressing problems of CT (e.g., applications to non-deterministic SUTs), problems of CT-FLA (e.g., fast combinatorial fault localization), or problems related to optimal CA generation. Hypotheses related to balance of optimal CAs have been reviewed in Subsection 2.5.4, but even if it is not possible to make assertions pertaining the balance of (a set of) optimal CAs, as outlined by Corollary 2.5.21, balance properties of CAs can at least aid the (computational) search for optimal CAs. Balanced CAs may also present an alternative approach to CAs of higher index for testing non-deterministic SUTs, since multiple executions of

combinations of certain length can be enforced or prevented via balance constraints. This may be especially of interest when a sub-system of a comprised SUT is non-deterministic. Finally, connections between the intersection and separation properties of locating and detecting arrays (see [62]), and balance properties remain to be investigated.

Novel domains. Ultimately, the interplay between combinatorial design theory and software testing does only represent *one* fruitful interaction of combinatorial design theory with an application domain. Analogous to CT, combinatorial design theory may lead to fruitful interactions with other domains, such as hardware, cyber-physical systems, or – returning to the roots of design of experiments – the medical domain and the agricultural domain.

List of Acronyms

Below we provide a list of acronyms used throughout this thesis, sorted alphabetically. These acronyms are frequently used, and the list below links to the respective first use of the acronym. Nevertheless, the full name might be used, or the acronym may be recalled when deemed appropriate at certain places such as in chapter introductions.

AETG Automatic Efficient Test Generator 7

AI Artificial Intelligence 216

ANNs Artificial Neural Networks 145

ASICs application-specific integrated circuits 237

CA Covering Array 4

CT Combinatorial Testing 5

CT-FLA Combinatorial Testing Fault Localization 11

DAs detecting arrays 17

FIT failure inducing tuple 20

t**-FIT** failure inducing t-way interaction of length t 20

FPGA field-programmable gate array 236

HT hardware Trojan 233

ICs integrated circuits 237

IPM Input Parameter Model 5

LAs locating arrays 17

LLM Large Language Model 215

MCAs Mixed-level Covering Arrays 13

ML Machine Learning 201

PAs packing arrays 78

SCAs Sequence Covering Arrays 18

SUT System Under Test 5

VCA variable strength covering array 17

VCAN variable strength covering array number 17

XAI explainable artificial intelligence 265

List of Figures

1.1 A 9×9 Sudoku puzzle from [23] on the left, and the two (unique) solutions for its completion on the right [23]. 4
1.2 The transposed orthogonal array $\mathsf{OA}(81, 3, 9, 2)$ corresponding to the Latin square in Figure 1.1. 4
1.3 The *combinatorial testing process/cycle*: a simplified testing process (CT-affected parts in red) for a given SUT. See the "Combination Strategies Testing Process" in [26]. 22
1.4 Overall methodology of Combinatorial Design Theory and Software Testing. 31

2.1 Venn diagram of structures related to CAs. 57
2.2 Updated Venn diagram of structures related to CAs. 62
2.3 $N = 12^*$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 12$ rows, strength $t = 3$ and a binary alphabet, see also Table [210]. 102
2.4 $N = 13$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 13$ rows, strength $t = 3$ and a binary alphabet, see also Table [211]. 102
2.5 $N = 14$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 14$ rows, strength $t = 3$ and a binary alphabet, see also Table [212]. 102
2.6 $N = 15^*$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 15$ rows, strength $t = 3$ and a binary alphabet, see also Table [213]. 102
2.7 $N = 16^*$, $t = 3$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 3$ and a binary alphabet, see also Table [214]. 103
2.8 $N = 18$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 18$ rows, strength $t = 4$ and a binary alphabet, see also Table [217]. 105
2.9 $N = 19$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 19$ rows, strength $t = 4$ and a binary alphabet, see also Table [218]. 105

2.10 $N = 20$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 20$ rows, strength $t = 4$ and a binary alphabet, see also Table [219]. 106
2.11 $N = 21^*$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 21$ rows, strength $t = 4$ and a binary alphabet, see also Table [220]. 106
2.12 $N = 22$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 22$ rows, strength $t = 4$ and a binary alphabet, see also Table [221]. 107
2.13 $N = 23$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 23$ rows, strength $t = 4$ and a binary alphabet, see also Table [222]. 107
2.14 $N = 24^*$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 24$ rows, strength $t = 4$ and a binary alphabet, see also Table [223]. 107
2.15 $N = 25$, $t = 4$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 25$ rows, strength $t = 4$ and a binary alphabet, see also Table [224]. 108
2.16 $N = 52^*$, $t = 5$, $v = 2$ – Graphical representation of the classification results of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 52$ rows, strength $t = 5$ and a binary alphabet, see also Table [188]. 109
2.17 On the left, the graph representation of the automaton $\mathfrak{A}$ given in Example 2.6.5; on the right its transition matrix E indexed with the states q_0, q_1, q_2, q_3 for better readability. 117
2.18 The t-wise automaton corresponding to $w = a_1 a_2 \dots a_t$. Note that the weights "$*$" of loops represents any symbol of the alphabet, i.e. Σ. 123
2.19 (a) The infiltration product automaton of the linear automata $\mathfrak{A}_{bab}$ and $\mathfrak{A}_{acb}$, which when adding loops weighted with the entire alphabet Σ (represented as $*$) to each state equals to (b) the Hadamard product of the 3-wise automata $\mathfrak{A}_{bab\leq}$ and $\mathfrak{A}_{acb\leq}$. 125
2.20 (a) An automaton with the behaviour $bab \sqcup\!\sqcup acb$ and (b) a visualization of the enforced synchronization (bold diagonal transition) in reduced shuffle products of linear automata; the dashed transitions are removed in comparison to the infiltration product. 126
2.21 (a) A reduced automaton where dead states and transitions are removed from the automaton in Fig. 2.20a, with behaviour $bab \sqcup\!\sqcup acb$, (b) the automaton in (a) with added loops on each state. 127

3.1 An overview of the presented procedures connecting covering arrays (CAs), set covers (SCs), vertex covers (VCs) and independent sets (IS), which will be solved with Boltzmann machines (BMs). See [181] and [247]. 149
3.2 $G_{2,4,2}$ underlying the CA instance $t = 2$, $k = 4$, $v = 2$. 151

3.3 Subgraph of $G_{2,4,2}$, the underlying graph of the CA instance $t = 2$, $k = 4$, $v = 2$ and resulting reduced graphs applying a graph reduction of *type I* and *type II*. 153
3.4 A graph corresponding to the set cover $(U, \mathcal{S})$ in Example 3.2.9. 156
3.5 Minimal number of rows (smaller is better) of generated CAs by configurations of Algorithm 4 applied to the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$. Not all configurations can find CAs. Note the vertical axis is in $\log_{10}$-scale. 166
3.6 Minimal number of rows of generated CAs by configurations of Algorithm 4 applied to the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. Only two configurations can find CAs. The vertical axis depicts the number of rows (smaller is better) of CAs found by the different configurations. Note the vertical axis is in $\log_{10}$-scale. 166
3.7 Minimal number of rows (smaller is better) of generated CAs by configurations of Algorithm 6 applied to the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$. Not all configurations can produce results in reasonable time. Note the vertical axis is in linear scale. 170
3.8 Minimal number of rows (smaller is better) of generated CAs by configurations of Algorithm 6 applied to the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. Only two configurations can produce results in reasonable time. Note the vertical axis is in linear scale. 170
3.9 A comparison of Algorithms 4, Algorithm 6 and Algorithm 7, each configuration involving HeavyGraph and BMweight and Algorithm 7 using HeavyEdges in addition, by means of the minimal number of rows of generated CAs for the CA instances $(3, k, 2)$ for $5 \leq k \leq 50$. Note the vertical axis is in linear scale. 173
3.10 A comparison of Algorithms 4, Algorithm 6 and Algorithm 7, each configuration using HeavyGraph and BMweight and Algorithm 7 using HeavyEdges in addition, by means of the minimal number of rows of generated CAs for the CA instances $(2, k, 4)$ for $5 \leq k \leq 50$. Note the vertical axis is in linear scale. 173
3.11 The learning progression, evaluated by means of $\mathsf{CAN}/|V \setminus \mathcal{I}_{max}|$, of the two different GraphUpdate procedures for the CA instances $(t = 2, k = 6)$ and $(t = 3, k = 5)$. The growth of $\mathsf{CAN}/|\mathcal{I}_{max}|$ is shown on the vertical axis over the learning phases, on the horizontal axis. 175
3.12 The runtime in seconds (vertical axis) of the different algorithms is depicted for different values of k on a logarithmic scale. 176
3.13 Schematics of the incidence structures used in the implementations of the considered algorithms. 183
3.14 Reduced qubit representation (a) and used update mechanics (b) and (c). 189
3.15 Visualization of the onion concept, with the different bias layers on the left and an example array on the right. 191
3.16 Parameter evaluation for IPO-Q on the instance $\mathsf{CA}(N; 3, k)$ (smaller values are better). 192

3.17 Comparison of the number of rows of generated CAs by IPO-Q and IPOG-F with the best known upper bounds for CAN, maintained at [97], denoted as "CAN <=", for strengths $t = 2, 3, 4$ from top to bottom (smaller values are better). 194

4.1 Testing process for testing the BACnet protocol implementation of the Harmonization Tool of the APOGEE Insight®. 206
4.2 On the left hand side a covering array (of strength 2), and on the right hand side the derived 2-way test set for the IPM of the reduced EEO. 207
4.3 An overview of the proposed testing approach, instantiating a simplified generic CT process (see also Figure 1.3) for LLM testing. CT dependent steps are colored in red, LLM specific steps are colored in gray. 215

5.1 An example design of an HT of *length* eight, e.g., for an AES module, that consumes the gates in positions `4-25-38-47-95-101-115-127` corresponding to the key or plaintext input and that is activated when the signals 10110110 occur in these positions. 238
5.2 The percentage of missed patterns when using randomly generated arrays ($rand(CT\ell)$ and $rand_N(CT\ell)$) on the vertical axis, for the values of $\ell = 1, 2, 3, 4$ on the horizontal axis. 260
5.3 The total number of missed patterns when using randomly generated arrays ($rand(CT\ell)$ and $rand_N(CT\ell)$) on the vertical axis as base-10 logarithms, for the values of $\ell = 1, 2, 3, 4$ on the horizontal axis. 261
5.4 An example of an HT with a generalized trigger logic, activated by the signals 10 in position 25-38 or by the signals 010 in positions 95 - 115 - 127. . . . 265
5.5 Visualization of the combinatorial testing cycle, indicating the working areas of the contributing organizations AVL List GmbH (AVL), Institute for Software Technology, Graz University of Technology (IST) and the MATRIS Research Group from SBA Research (MATRIS) in this case study. 280
5.6 The structure of the CT workflow in the considered scenario, the numbers indicate the order in which the steps are performed. The schematics shows leftmost the computation unit from AVL, in the middle the development and testing unit of AVL, and rightmost the research groups of IST and MATRIS. The numbers (i) for $i = 0, \ldots, 12$ indicate the order in which the steps are performed. 281
5.7 Excerpt from the visualization of a driving scenario, visualizing the Ego vehicle's cone-shaped radar field of view in yellow, and the blind area in the red. 284
5.8 Excerpts from driving scenarios derived from tests that cover the three potential 3-way FITs that appear in the highest number of failing tests, i.e. crash scenarios. The corresponding potential 3-way FITs are given below the images. 290

6.1 Simplified testing process with the CT-affected parts in red, and the contributions of this thesis mapped to the specific parts. 296

List of Tables

1.1 An example of an (optimal) $\mathsf{CA}(12; 3, 11, 2)$ 15

2.1 Overview of complexity results regarding main **classes** of covering arrays. Notice that the results for $CA(G)$ are obtained only for $v = 2$, while for CAFEs the NP-completeness results apply when $v \geq 5$. Results for optimal $\mathrm{CA}_{t,v}$ are for $t > 2$ or $v > 2$ and for optimal $\mathrm{VCA}_{\bar{t},v}$ for $v > 2$. 63

2.2 Comparison of exact algorithms to heuristic algorithms. The values in column "Hnich et al." correspond to the fastest algorithm as documented in [69], Tables 1 and 2. Bold entries indicate the smallest number of rows in this comparison, when no CA was found we denote the size as "-". 67

2.3 Results for non-binary VCAs. The values for VCAN are taken from [59]. . 68

2.4 The column **# Vars** lists the number of unknowns in the respective equation systems; **# Sols** lists the exact number of solutions, i.e. CAs for the respective instance, except for entries $\geq x$, which indicate that at least x solutions were found. Column **CAN** lists the exact values of CAN for given parameters (t, k, v). Column **Reference** gives a reference where the respective CAN value can be found in the literature, and, in case, a reason why there exists no such CA. 78

2.5 Key to the $\mathsf{CAK}^{\boldsymbol{y}}_{\boldsymbol{\lambda}}$ tables [208] and Table 2.6: abbreviations and their meaning. 98

2.6 $N = 15^*$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 15$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}^{\boldsymbol{y}^{\#}}_{\boldsymbol{\lambda}}$ *time*, where # represents the number of non-equivalent $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs and the time is given in seconds. 99

3.1 Results of the experiments with configurations of Algorithm 4, using no learning mechanisms. 167

3.2 Results of experiments with Configurations of Algorithm 6, using weight updates. 168

3.3 Results of experiments with different configurations of Algorithm 7, using graph updates and Algorithm 8. All configurations of Algorithm 7 use BMWEIGHT as initial weighting and the remaining buidling blocks as they are indicated by the column headings. 172

3.4 Experiments for constrained mixed-level covering arrays for real-world combinatorial testing models. 177

3.5 Bounds on number of rows of output CAs, runtime and memory usage. . 183
3.6 Comparison of sizes of generated CAs and runtimes. 185
3.7 Column limits with 16 GB of memory. 186
3.8 Results for selected CAs $\mathsf{CA}(N;t,k)$. Values for "CAN$\leq$" can be found at [97]. 193

4.1 Process Failures Detected using CT. The data processing component was tested using test cases based on the CA with 2127 columns and with alphabet sizes $v = 3$ [301]. 215
4.2 Overview of the linkage between concepts from LLMs (on the left) and those of CT (on the right). 220
4.3 The selected synonyms in the example are reasonable and largely maintain the question's semantics. 221
4.4 A pairwise combinatorial test set for the first question *"can you drink alcohol in public in denmark"* and its corresponding IPM with eight parameters. It consists of nine derived sentences, with the original sentence appearing as first row q_1. 222
4.5 Confusion values in the top half of the table: showing the percentage of "annotation/response" for the original and derived sentences and for each LLM. Performance results in the bottom half of the table: showing the percentage of T = true, F = false, and U = undefined responses, as well as three measures of accuracies (i.e. percentages of passing tests in terms of CT): "acc" is the accuracy over all sentences; "accWUR" is the accuracy without undefined responses is the accuracy when disregarding sentences where the response was U; "accCA" is the average accuracies for derived sentences. . 226
4.6 An IPM for an original sentence, where some of the synonyms change the meaning of the sentence. 227

5.1 A summary of the mappings between equivalent concepts and notions in HT location and combinatorial testing. 245
5.2 Comparison of sizes of test sets coming from CT methods (CTdetect & CTlocate) against other state of the art logic testing techniques for combinational HT detection. 255
5.3 The given patterns were used in the specified positions as HT trigger patterns. The remaining columns detail the trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 16 for HT location. 257
5.4 The values 11111111 were used in the positions specified in the first column as HT trigger pattern. The remaining columns detail the trigger and location capabilities of the CTℓ test sets for $\ell = 1, \ldots, 8$, using Algorithm 16 for HT location. 257
5.5 The positions 3-11-25-29-70-88-97-119 are fixed and used with the patterns specified in the first column as HT trigger pattern. The emaining columns detail HT trigger and location capabilities of the CTℓ test sets, using Algorithm 16. 257

5.6 The given patterns were used in the specified positions as HT trigger patterns. The remaining columns detail the trigger and location capabilities of the $rand(CT\ell)$ arrays for $\ell = 1, \ldots, 8$, using Algorithm 16 for HT location. . . 258
5.7 Measurements regarding the fast pattern location properties (i.e. number of patterns that can be located via Algorithm 16) of the $rand(CT\ell)$ arrays. The column 'loc' denotes whether all patterns can be located based on the respective array; the column "miss %" gives the percentage of patterns of length ℓ that can *not* be successfully located - "miss #" the total number; the column "time" denotes the time consumed for the measurement. 258
5.8 Measurements regarding the fast pattern location properties (i.e. number of patterns that can be located via Algorithm 16) of the $rand_N(CT\ell)$ arrays. The column "loc" denotes whether all patterns can be located based on the respective array; the column "miss %" gives the percentage of patterns of length ℓ that can *not* be successfully located - "miss #" the total number; the column "time" denotes the time consumed for the measurement. 258
5.9 Comparison of the runtimes of Algorithm 14 and 16 when used to locate an HT of length ℓ based on testing with the combinatorial test set CTℓ and the respective testing oracle. The given runtimes are averaged over 100 runs, where the HTs were placed randomly. 263
5.10 Measurements regarding the slow pattern location properties (i.e. patterns that can be located via Algorithm 14) of the $rand(CT\ell)$ arrays. The column 'loc' denotes whether all patterns can be located based on the respective array. 263
5.11 Overview of the four *dimensions* given in Ehsan et al. [413], the four *principles* described in [412] and the properties of XAI explanations proposed in this section. 269
5.12 Analogy between CT (left) and explanations for an AI system which produces classifications (right). 273
5.13 Mapping between concepts from virtual driving function testing and those of combinatorial testing. 281
5.14 Overview of the results after two rounds of testing of the virtual driving test platform following Algorithm 17. 285
5.15 The first row shows the quantities of potential 3-way FITs, appearing in the number of failing tests, i.e. crash scenarios, given in the second row. . . . 287

1 $N = 4$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 4$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 358
2 $N = 5$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 5$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 358
3 $N = 6$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 6$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 358

4 $N = 7$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 7$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 359
5 $N = 8$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 8$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 359
6 $N = 8$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 8$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 359
7 $N = 9$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 9$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 360
8 $N = 10$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 10$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 360
9 $N = 11$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 11$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 360
10 $N = 12^{*}$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 12$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 361
11 $N = 13$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 13$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 362
12 $N = 14$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 14$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 363
13 $N = 16^{*}$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). For this instance we let the computational search terminate which took a maximum of 63 hours. 364
14 $N = 17$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 17$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 366
15 $N = 16$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 368
16 $N = 17$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 17$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 368

17 $N = 18$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 18$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 368
18 $N = 19$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 19$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 368
19 $N = 20$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 20$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 369
20 $N = 21^*$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 21$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 370
21 $N = 22$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 22$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 371
22 $N = 23$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 23$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 373
23 $N = 24$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 24$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 376
24 $N = 25$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 25$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 380
25 $N = 52^*$, $t = 5$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 52$ rows, strength $t = 5$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 386
26 $N = 9^*$, $t = 2$, $v = 3$ – Results for the classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 9$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 389
27 $N = 10$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 10$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 389
28 $N = 11^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 11$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 389
29 $N = 12^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 12$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 389

30 $N = 13^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 13$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 390
31 $N = 14^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 14$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 390
32 $N = 27^*$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 27$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 390
33 $N = 28$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 28$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 391
34 $N = 29$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 29$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 391
35 $N = 30$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 30$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 391
36 $N = 31$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 31$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 391
37 $N = 32$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 32$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 392
38 $N = 33^*$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 33$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 393
39 $N = 16$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 394
40 $N = 17$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 17$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 394
41 $N = 18$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 18$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 394
42 $N = 19$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 19$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). 395

43 $N = 20$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 20$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 395
44 $N = 64$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 64$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 395
45 $N = 65$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 65$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 396
46 $N = 66$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 66$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 396
47 $N = 67$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 67$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds). 396

List of Algorithms

1 Det2Dec 45
2 AlgebraicSearchCAs 72
3 AllColumnExtensions 74
4 BMforCA 155
5 SA 155
6 BMforCAlearning 158
7 BMforCAlearningGraph 160
8 HFforCA 164
9 gAETG (for CAs) 178
10 slicedAETG 180
11 paraslicedAETG 182
12 IPOG(t, k, v) 187
13 IPO-Q 190
14 FullEnumerationLocation(A, o, ℓ) 246
15 EnumerationLocation(A, o, ℓ) 250
16 FastLocation(A,o) 251
17 CT-FLA Approach 283

Bibliography

[1] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2nd ed., 2016.

[2] B. Beizer, *Software testing techniques*. Van Nostrand Reinhold, Inc, New York, 2nd ed., 1990.

[3] M. Dowson, "The Ariane 5 software failure," *ACM SIGSOFT Software Engineering Notes*, vol. 22, p. 84, Mar. 1997.

[4] J. Neely, "Ethiopian Airlines questions Boeing's 'aggressive' software." https://www.reuters.com/article/idUSKCN1R50BE/, Mar. 2019.

[5] T. Hepher, "Airbus knew of software vulnerability before A400M crash." Available online at https://www.reuters.com/article/idUSKBN1D81BG/, Nov. 2017.

[6] D. E. Knuth, "Knuth: Frequently Asked Questions." Available online at https://www-cs-faculty.stanford.edu/~knuth/faq.html.

[7] "The correspondence between Donald E. Knuth and Peter van Emde Boas on priority deques during the spring of 1977." Available online at https://staff.fnwi.uva.nl/p.vanemdeboas/knuthnote.pdfk.

[8] National Institute of Standards and Technology (NIST), RTI Health, Social, and Economics Research, "The economic impacts of inadequate infrastructure for software testing," 2002.

[9] Consortium for Information & Software Quality™, "The Cost of Poor Software Quality in the US: A 2022 Report." Available online at https://www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf, 2022. Accessed on 2023-12-09.

[10] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, A. Bertolino, J. Jenny Li, and H. Zhu, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, pp. 1978–2001, Aug. 2013.

[11] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[12] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, (New York, NY, USA), p. 285–294, Association for Computing Machinery, 1999.

[13] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pp. 153–163, 2011.

[14] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[15] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[16] S. Dalal and A. McIntosh, "When to stop testing for large software systems with changing code," *IEEE Transactions on Software Engineering*, vol. 20, pp. 318–323, Apr. 1994.

[17] P. Cao, K. Yang, and K. Liu, "Optimal selection and release problem in software testing process: A continuous time stochastic control approach," *European Journal of Operational Research*, vol. 285, pp. 211–222, Aug. 2020.

[18] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: a survey," *Software Testing, Verification and Reliability*, vol. 15, no. 3, pp. 167–199, 2005.

[19] C. J. Colbourn and J. H. Dinitz, *Handbook of combinatorial designs*. CRC press, 2006.

[20] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, vol. 16 of *North-Holland Mathematical Library*. Elsevier, 1977.

[21] O. Collins and M. Hizlan, "Determinate state convolutional codes," *IEEE Transactions on Communications*, vol. 41, pp. 1785–1794, Dec. 1993.

[22] C. J. Colbourn, "The complexity of completing partial Latin squares," *Discrete Applied Mathematics*, vol. 8, pp. 25–30, Apr. 1984.

[23] A. M. Herzberg and M. R. Murty, "Sudoku squares and chromatic polynomials," *Notices of the AMS*, vol. 54, no. 6, pp. 708–717, 2007.

[24] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, *Orthogonal arrays: theory and applications*. Springer Science & Business Media, 1st ed., 1999.

[25] D. Kuhn, R. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, Taylor & Francis, 1st ed., 2013.

[26] M. Grindal and J. Offutt, "Input parameter modeling for combination strategies," in *Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering*, SE'07, (Anaheim, CA, USA), pp. 255–260, ACTA Press, 2007.

[27] 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)*, pp. 549–556, 2007.

[28] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 901–924, 2015.

[29] Tang and Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1145–1150, 1983.

[30] G. Seroussi and N. H. Bshouty, "Vector sets for exhaustive testing of logic circuits," *IEEE Transactions on Information Theory*, vol. 34, no. 3, pp. 513–522, 1988.

[31] R. Mandl, "Orthogonal Latin squares: an application of experiment design to compiler testing," *Communications of the ACM*, vol. 28, pp. 1054–1058, Oct. 1985.

[32] A. Williams and R. Probert, "A practical strategy for testing pair-wise coverage of network interfaces," in *Proceedings of ISSRE '96: 7th International Symposium on Software Reliability Engineering*, (White Plains, NY, USA), pp. 246–254, IEEE Comput. Soc. Press, 1996.

[33] R. Brownlie, J. Prowse, and M. S. Phadke, "Robust testing of AT&T PMX/StarMAIL using OATS," *AT&T Technical Journal*, vol. 71, no. 3, pp. 41–47, 1992.

[34] Y. Levendel, "Improving quality with a manufacturing process," *IEEE Software*, vol. 8, pp. 13–25, Mar. 1991.

[35] D. Cohen, S. Dalal, A. Kajla, and G. Patton, "The automatic efficient test generator (AETG) system," in *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*, pp. 303–309, 1994.

[36] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.

[37] S. R. Dalal and G. C. Patton, "Automatic efficient test generator (AETG): A test generation system for screen testing, protocol verification, and feature interactions testing," *Internal Bellcore Technical Memorandum,*, 1993.

[38] K. Burroughs, A. Jain, and R. Erickson, "Improved quality of protocol testing through techniques of experimental design," in *Proceedings of ICC/SUPERCOMM'94 - 1994 International Conference on Communications*, pp. 745–752 vol.2, May 1994.

[39] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino, "Applying design of experiments to software testing: experience report," in *Proceedings of the 19th international conference on Software engineering - ICSE '97*, (Boston, Massachusetts, United States), pp. 205–215, ACM Press, 1997.

[40] N. J. Sloane, "Covering arrays and intersecting codes," *Journal of combinatorial designs*, vol. 1, no. 1, pp. 51–63, 1993.

[41] B. Stevens and E. Mendelsohn, "Efficient software testing protocols," in *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '98, p. 22, IBM Press, 1998.

[42] D. Cohen, S. Dalal, J. Parelius, and G. Patton, "The combinatorial design approach to automatic test generation," *IEEE Software*, vol. 13, pp. 83–88, Sept. 1996. Conference Name: IEEE Software.

[43] D. R. Wallace and D. R. Kuhn, "Failure modes in medical device software: An analysis of 15 years of recall data," *International Journal of Reliability, Quality and Safety Engineering*, vol. 08, no. 04, pp. 351–371, 2001.

[44] D. Kuhn and M. Reilly, "An investigation of the applicability of design of experiments to software testing," in *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings.*, (Greenbelt, MD, USA), pp. 91–95, IEEE Comput. Soc, 2003.

[45] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[46] B. Smith, M. S. Feather, and N. Muscettola, "Challenges and methods in testing the remote agent planner," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems - AIPS 2000 Planning Competition*, 2000.

[47] J. PAN, "The dimensionality of failures-a fault model for characterizing software robustness," *Proc. FTCS '99, June*, 1999.

[48] Z. B. Ratliff, D. R. Kuhn, R. N. Kacker, Y. Lei, and K. S. Trivedi, "The relationship between software bug type and number of factors involved in failures," in *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 119–124, 2016.

[49] K. Z. Bell, "Optimizing effectiveness and efficiency of software testing: A hybrid approach," *PhD Dissertation, North Carolina State University*, 2006.

[50] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Estimating t-way fault profile evolution during testing," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 596–597, 2016.

[51] K. Fögen and H. Lichter, "A case study on robustness fault characteristics for combinatorial testing-results and challenges.," in *2018 6th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2018) co-located with APSEC 2018*, pp. 22 – 29, December 2018.

[52] D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi, "How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation," *Journal of Systems and Software*, vol. 113, pp. 27–43, Mar. 2016.

[53] R. Kuhn, R. N. Kacker, Y. Lei, and D. Simos, "Input space coverage matters," *Computer*, vol. 53, no. 1, pp. 37–44, 2020.

[54] C. J. Wu and M. S. Hamada, *Experiments: Planning, Analysis, and Optimization*, ch. Full Factorial Experiments at Two Levels, pp. 151–204. New Jersey, USA: John Wiley & Sons, 3rd ed., 2021.

[55] A. Dean, M. Morris, J. Stufken, and D. Bingham, eds., *Handbook of Design and Analysis of Experiments*. New York: Chapman and Hall/CRC, June 2015.

[56] L. Hu, W. E. Wong, D. R. Kuhn, and R. N. Kacker, "How does combinatorial testing perform in the real world: An empirical study," *Empirical Software Engineering*, vol. 25, pp. 2661–2693, Jul 2020.

[57] H. Wu, C. Nie, J. Petke, Y. Jia, and M. Harman, "An empirical comparison of combinatorial testing, random testing and adaptive random testing," *IEEE Transactions on Software Engineering*, vol. 46, no. 3, pp. 302–320, 2020.

[58] L. Moura, J. Stardom, B. Stevens, and A. Williams, "Covering arrays with mixed alphabet sizes," *Journal of Combinatorial Designs*, vol. 11, no. 6, pp. 413–432, 2003.

[59] S. Raaphorst, L. Moura, and B. Stevens, "Variable strength covering arrays," *Journal of Combinatorial Designs*, vol. 26, no. 9, pp. 417–438, 2018.

[60] P. Danziger, E. Mendelsohn, L. Moura, and B. Stevens, "Covering arrays avoiding forbidden edges," in *Combinatorial Optimization and Applications*, (Berlin, Heidelberg), pp. 296–308, Springer Berlin Heidelberg, 2008.

[61] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pair-wise coverage with seeding and constraints," *Information and Software Technology*, vol. 48, no. 10, pp. 960–970, 2006. Advances in Model-based Testing.

[62] C. J. Colbourn and D. W. McClary, "Locating and detecting arrays for interaction faults," *Journal of Combinatorial Optimization*, vol. 15, pp. 17–48, Jan 2008.

[63] Y. M. Chee, C. J. Colbourn, D. Horsley, and J. Zhou, "Sequence covering arrays," *SIAM Journal on Discrete Mathematics*, vol. 27, no. 4, pp. 1844–1861, 2013.

[64] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*, pp. 237–266, Boston, MA: Springer US, 2005.

[65] A. Hartman and L. Raskin, "Problems and algorithms for covering arrays," *Discrete Mathematics*, vol. 284, no. 1, pp. 149 – 156, 2004. Special Issue in Honour of Curt Lindner on His 65th Birthday.

[66] L. Kampel and D. E. Simos, "A survey on the state of the art of complexity problems for covering arrays," *Theoretical Computer Science*, vol. 800, pp. 107–124, 2019.

[67] L. Kampel, D. E. Simos, B. Garn, I. S. Kotsireas, and E. Zhereshchin, "Algebraic models for arbitrary strength covering arrays over v-ary alphabets," in *Algebraic Informatics*, (Cham), pp. 177–189, Springer International Publishing, 2019.

[68] L. Kampel, P. Kitsos, and D. E. Simos, "Locating hardware Trojans using combinatorial testing for cryptographic circuits," *IEEE Access*, vol. 10, pp. 18787–18806, 2022.

[69] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith, "Constraint models for the covering test problem," *Constraints*, vol. 11, pp. 199–219, Jul 2006.

[70] J. Lawrence, R. N. Kacker, Y. Lei, D. R. Kuhn, and M. Forbes, "A survey of binary covering arrays," *The Electronic Journal of Combinatorics*, vol. 18, pp. #P84 1–30, Apr. 2011.

[71] R. C. Bryce and C. J. Colbourn, "A density-based greedy algorithm for higher strength covering arrays," *Softw. Test. Verif. Reliab.*, vol. 19, pp. 37–53, Mar. 2009.

[72] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, and J. S. Collofello, "A variable strength interaction testing of components," in *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pp. 413–418, IEEE, 2003.

[73] S. Raaphorst, *Variable strength covering arrays*. PhD thesis, School of Electrical Engineering and Computer Science, Faculty of Engineering, 2013.

[74] Y. Tang, C. J. Colbourn, and J. Yin, "Optimality and constructions of locating arrays," *Journal of Statistical Theory and Practice*, vol. 6, pp. 20–29, Mar. 2012.

[75] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, "Combinatorial methods for event sequence testing," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pp. 601–609, IEEE, 2012.

[76] C. J. Colbourn and V. R. Syrotiuk, "On a combinatorial framework for fault characterization," *Mathematics in Computer Science*, vol. 12, pp. 429–451, Dec. 2018.

[77] M. Grochtmann and K. Grimm, "Classification trees for partition testing," *Software Testing, Verification and Reliability*, vol. 3, no. 2, pp. 63–82, 1993.

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

[79] D. R. Kuhn, R. N. Kacker, Y. Lei, and J. Torres-Jimenez, "Equivalence class verification and oracle-free testing using two-layer covering arrays," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 1–4, 2015.

[80] A. Bhat and S. M. K. Quadri, "Equivalence class partitioning and boundary value analysis - a review," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1557–1562, 2015.

[81] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[82] C. Nie and H. Leung, "The minimal failure-causing schema of combinatorial testing," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, Sept. 2011.

[83] X. Niu, C. Nie, H. Leung, Y. Lei, X. Wang, J. Xu, and Y. Wang, "An interleaving approach to combinatorial testing and failure-inducing interaction identification," *IEEE Transactions on Software Engineering*, vol. 46, no. 6, pp. 584–615, 2020.

[84] L. S. Ghandehari, Y. Lei, D. Kung, R. Kacker, and R. Kuhn, "Fault localization based on failure-inducing combinations," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 168–177, 2013.

[85] K. Nishiura, E. Choi, and O. Mizuno, "Improving faulty interaction localization using logistic regression," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 138–149, 2017.

[86] C. Martínez, L. Moura, D. Panario, and B. Stevens, "Locating errors using elas, covering arrays, and adaptive testing algorithms," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 4, pp. 1776–1799, 2009.

[87] D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial coverage measurement concepts and applications," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 352–361, 2013.

[88] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing," *Software Testing, Verification and Reliability*, vol. 18, no. 3, pp. 125–148, 2008.

[89] S. Preeti, B. Milind, M. S. Narayan, and K. Rangarajan, "Building combinatorial test input model from use case artefacts," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 220–228, 2017.

[90] A. Zamansky, A. Shwartz, S. Khoury, and E. Farchi, "A composition-based method for combinatorial test design," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 249–252, 2017.

[91] M. N. Borazjany, L. S. Ghandehari, Y. Lei, R. Kacker, and R. Kuhn, "An input space modeling methodology for combinatorial testing," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 372–381, 2013.

[92] I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Common patterns in combinatorial models," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pp. 624–629, 2012.

[93] 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)*, pp. 229–238, March 2017.

[94] F. Wotawa and Y. Li, "From ontologies to input models for combinatorial testing," in *Testing Software and Systems*, (Cham), pp. 155–170, Springer International Publishing, 2018.

[95] E. Farchi, I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Combinatorial testing with order requirements," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pp. 118–127, 2014.

[96] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, 2008.

[97] C. J. Colbourn, "Covering Array Tables for t=2,3,4,5,6." Available online at `http://www.public.asu.edu/~ccolbou/src/tabby/catable.html`, Accessed on 2023-08-19.

[98] J. Torres-Jimenez, I. Izquierdo-Marquez, and H. Avila-George, "Methods to construct uniform covering arrays," *IEEE Access*, vol. 7, pp. 42774–42797, 2019.

[99] G. Katona, "Intersection theorems for systems of finite sets," *Acta Mathematica Hungarica*, vol. 15, no. 3-4, pp. 329–337, 1964.

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

[101] K. A. Bush, "Orthogonal arrays of index unity," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 426–434, 1952.

[102] M. A. Chateauneuf, C. J. Colbourn, and D. L. Kreher, "Covering arrays of strength three," *Designs, Codes and Cryptography*, vol. 16, pp. 235–242, May 1999.

[103] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Mathematiche*, vol. LIX, no. I-II, pp. 125–172, 2004.

[104] J. R. Lobb, C. J. Colbourn, P. Danziger, B. Stevens, and J. Torres-Jimenez, "Cover starters for covering arrays of strength two," *Discrete Mathematics*, vol. 312, no. 5, pp. 943 – 956, 2012.

[105] C. J. Colbourn and G. Kéri, "Binary covering arrays and existentially closed graphs," in *Coding and Cryptology*, pp. 22–33, Springer Berlin Heidelberg, 2009.

[106] C. J. Colbourn, "Covering arrays from cyclotomy," *Designs, Codes and Cryptography*, vol. 55, pp. 201–219, May 2010.

[107] C. J. Colbourn, E. Lanus, and K. Sarkar, "Asymptotic and constructive methods for covering perfect hash families and covering arrays," *Designs, Codes and Cryptography*, vol. 86, pp. 907–937, Apr 2018.

[108] M. Wagner, C. J. Colbourn, and D. E. Simos, "In-parameter-order strategies for covering perfect hash families," *Applied Mathematics and Computation*, vol. 421, p. 126952, 2022.

[109] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pp. 72–77 vol.1, 2004.

[110] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Application of particle swarm optimization to uniform and variable strength covering array construction," *Applied Soft Computing*, vol. 12, no. 4, pp. 1330–1347, 2012.

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

[112] J. Torres-Jimenez and H. Avila-George, "Search-based software engineering to construct binary test-suites," in *Trends and Applications in Software Engineering*, (Cham), pp. 201–212, Springer International Publishing, 2016.

[113] M. B. Cohen, C. J. Colbourn, and A. C. Ling, "Constructing strength three covering arrays with augmented annealing," *Discrete Mathematics*, vol. 308, no. 13, pp. 2709 – 2722, 2008. Combinatorial Designs: A tribute to Jennifer Seberry on her 60th Birthday.

[114] J. Torres-Jimenez and I. Izquierdo-Marquez, "Covering arrays of strength three from extended permutation vectors," *Designs, Codes and Cryptography*, vol. 86, no. 11, pp. 2629–2643, 2018.

[115] J. Torres-Jimenez and I. Izquierdo-Marquez, "Improved covering arrays using covering perfect hash families with groups of restricted entries," *Applied Mathematics and Computation*, vol. 369, p. 124826, 2020.

[116] K. Sarkar and C. J. Colbourn, "Two-stage algorithms for covering array construction," *Journal of Combinatorial Designs*, vol. 27, no. 8, pp. 475–505, 2019.

[117] 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*, pp. 370–375, March 2013.

[118] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Software Testing, Verification and Reliability*, vol. 17, no. 3, pp. 159–182, 2007.

[119] E. Maltais and L. Moura, "Hardness results for covering arrays avoiding forbidden edges and error-locating arrays," *Theoretical Computer Science*, vol. 412, pp. 6517–6530, Oct. 2011.

[120] R. C. Bryce and C. J. Colbourn, "Test prioritization for pairwise interaction coverage," in *Proceedings of the 1st International Workshop on Advances in Model-based Testing*, A-MOST '05, (New York, NY, USA), pp. 1–7, ACM, 2005.

[121] P. Nayeri, C. J. Colbourn, and G. Konjevod, "Randomized post-optimization of covering arrays," *European Journal of Combinatorics*, vol. 34, no. 1, pp. 91 – 103, 2013. Combinatorics and Stringology.

[122] L. Yu, Y. Lei, R. N. Kacker, D. R. Kuhn, and J. Lawrence, "Efficient algorithms for t-way test sequence generation," in *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pp. 220–229, 2012.

[123] A. Bombarda and A. Gargantini, "An automata-based generation method for combinatorial sequence testing of finite state machines," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 157–166, 2020.

[124] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas, and C. Zander, "Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers," *Computer Science Education*, vol. 18, no. 2, pp. 93–116, 2008.

[125] I. R. Katz and J. R. Anderson, "Debugging: An analysis of bug-location strategies," *Human–Computer Interaction*, vol. 3, no. 4, pp. 351–399, 1987.

[126] D. Hao, L. Zhang, T. Xie, H. Mei, and J.-S. Sun, "Interactive fault localization using test information," *Journal of Computer Science and Technology*, vol. 24, pp. 962–974, Sept. 2009.

[127] M. Renieres and S. Reiss, "Fault localization with nearest neighbor queries," in *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, (Montreal, Que., Canada), pp. 30–39, IEEE Comput. Soc, 2003.

[128] A. Zeller, "Isolating cause-effect chains from computer programs," *ACM SIGSOFT Software Engineering Notes*, vol. 27, pp. 1–10, Nov. 2002.

[129] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, pp. 707–740, Aug 2016.

[130] L. S. Ghandehari, J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn, "BEN: A combinatorial testing-based fault localization tool," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 1–4, April 2015.

[131] R. Jayaram and R. Krishnan, "Approaches to fault localization in combinatorial testing: A survey," in *Smart Computing and Informatics*, (Singapore), pp. 533–540, Springer Singapore, 2018.

[132] H. Jin and T. Tsuchiya, "Constrained locating arrays for combinatorial interaction testing," *Journal of Systems and Software*, vol. 170, p. 110771, 2020.

[133] Y. Tang, C. J. Colbourn, and J. Yin, "Optimality and constructions of locating arrays," *Journal of Statistical Theory and Practice*, vol. 6, pp. 20–29, Mar. 2012.

[134] S. A. Seidel, K. Sarkar, C. J. Colbourn, and V. R. Syrotiuk, "Separating interaction effects using locating and detecting arrays," in *Combinatorial Algorithms*, pp. 349–360, Cham: Springer International Publishing, 2018.

[135] E. Lanus, C. J. Colbourn, and D. C. Montgomery, "Partitioned search with column resampling for locating array construction," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 214–223, 2019.

[136] H. Jin, C. Shi, and T. Tsuchiya, "Constrained detecting arrays for fault localization in combinatorial testing," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, (New York, NY, USA), p. 1971–1978, Association for Computing Machinery, 2020.

[137] C. J. Colbourn and V. R. Syrotiuk, *There Must be Fifty Ways to Miss a Cover*, pp. 319–333. 1st ed., Florida, USA, Chapman and Hall/CRC, 2019.

[138] L. Kampel, M. Leithner, and D. E. Simos, "Sliced AETG: a memory-efficient variant of the AETG covering array generation algorithm," *Optimization Letters*, vol. 14, no. 6, pp. 1543–1556, 2020.

[139] L. Kampel, I. Hiess, I. S. Kotsireas, and D. E. Simos, "Balanced covering arrays: A classification of covering arrays and packing arrays via exact methods," *Journal of Combinatorial Designs*, vol. 31, no. 4, pp. 205–261, 2023.

[140] L. Kampel, M. Koelbing, and D. E. Simos, "A reduced-shuffe product of finite automata arising from sequential combinatorial testing." Submitted for publication.

[141] L. Kampel, M. Wagner, I. S. Kotsireas, and D. E. Simos, "How to use Boltzmann machines and neural networks for covering array generation," in *Learning and Intelligent Optimization*, (Cham), pp. 53–68, Springer International Publishing, 2020.

[142] L. Kampel, M. Wagner, I. S. Kotsireas, and D. E. Simos, "A primer on the application of neural networks to covering array generation," *Optimization Methods and Software*, vol. 37, no. 3, pp. 1165–1202, 2022.

[143] M. Wagner, L. Kampel, and D. E. Simos, "IPO-Q: A quantum-inspired approach to the IPO strategy used in CA generation," in *Mathematical Aspects of Computer and Information Sciences*, (Cham), pp. 313–323, Springer International Publishing, 2020.

[144] L. Kampel, M. Leithner, B. Garn, and D. E. Simos, "Problems and algorithms for covering arrays via set covers," *Theoretical Computer Science*, vol. 800, pp. 90–106, 2019.

[145] D. E. Simos, L. Kampel, and M. Ozcan, "Combinatorial methods for testing communication protocols in smart cities," in *Learning and Intelligent Optimization*, (Cham), pp. 437–440, Springer International Publishing, 2019.

[146] D. Jarman, R. Smith, G. Gosney, L. Kampel, M. Leithner, D. Simos, R. Kacker, and R. Kuhn, "Applying combinatorial testing to large-scale data processing at Adobe," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 190–193, 2019.

[147] B. Garn, L. Kampel, M. Leithner, B. Celic, C. Çulha, I. Hiess, K. Kieseberg, M. Koelbing, D.-P. Schreiber, M. Wagner, C. Wech, J. Zivanovic, and D. E. Simos, "Applying pairwise combinatorial testing to large language model testing," in *Testing Software and Systems*, (Cham), pp. 247–256, Springer Nature Switzerland, 2023.

[148] L. Kampel, D. E. Simos, D. R. Kuhn, and R. N. Kacker, "An exploration of combinatorial testing-based approaches to fault localization for explainable AI," *Annals of Mathematics and Artificial Intelligence*, vol. 90, no. 7, pp. 951–964, 2022.

[149] L. Kampel, M. Wagner, D. E. Simos, M. Nica, D. Dodig, D. Kaufmann, and F. Wotawa, "Applying CT-FLA for AEB function testing: A virtual driving case study," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 237–245, 2023.

[150] E. Maltais and L. Moura, "Finding the best CAFE is NP-hard," in *LATIN 2010: Theoretical Informatics* (A. López-Ortiz, ed.), (Berlin, Heidelberg), pp. 356–371, Springer Berlin Heidelberg, 2010.

[151] E. Balas and M. W. Padberg, "Set partitioning: A survey," *SIAM review*, vol. 18, no. 4, pp. 710–760, 1976.

[152] A. W. Williams and R. L. Probert, "Formulation of the interaction test coverage problem as an integer program," in *Testing of Communicating Systems XIV: Application to Internet Technologies and Services*, pp. 283–298, Boston, MA: Springer US, 2002.

[153] D. A. Bulutoglu and F. Margot, "Classification of orthogonal arrays by integer programming," *Journal of Statistical Planning and Infece*, vol. 138, no. 3, pp. 654–666, 2008.

[154] B. Garn and D. E. Simos, "Algebraic modelling of covering arrays," in *Applications of Computer Algebra*, (Cham), pp. 149–170, Springer International Publishing, 2017.

[155] B. Garn and D. E. Simos, "Algebraic techniques for covering arrays and related structures," *Electronic Notes in Discrete Mathematics*, vol. 70, pp. 49–54, 2018. TCDM 2018 – 2nd IMA Conference on Theoretical and Computational Discrete Mathematics, University of Derby.

[156] M. Koelbing, B. Garn, E. Iurlano, I. S. Kotsireas, and D. E. Simos, "Algebraic and sat models for sca generation," *Applicable Algebra in Engineering, Communication and Computing*, 2023.

[157] P. Murray and C. Colbourn, "Sequence covering arrays and linear extensions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8986, pp. 274–285, Springer Verlag, 2015.

[158] F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, "An approach to t-way test sequence generation with constraints," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 241–250, 2019.

[159] C. Colbourn, G. Kéri, P. Rivas Soriano, and J.-C. Schlage-Puchta, "Covering and radius-covering arrays: Constructions and classification," *Discrete Applied Mathematics*, vol. 158, no. 11, pp. 1158–1180, 2010.

[160] J. Torres-Jimenez and I. Izquierdo-Marquez, "Construction of non-isomorphic covering arrays," *Discrete Mathematics, Algorithms and Applications*, vol. 08, no. 02, p. 1650033, 2016.

[161] J. I. Kokkala, K. Meagher, R. Naserasr, K. J. Nurmela, P. R. J. Östergård, and B. Stevens, "On the structure of small strength-2 covering arrays," *Journal of Combinatorial Designs*, vol. 28, no. 1, pp. 5–24, 2020.

[162] P. Kaski, P. R. Östergård, and R. Patric, *Classification algorithms for codes and designs*, vol. 15. Springer, 2006.

[163] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "NP-completeness," in *Introduction to algorithms*, pp. 1048–1053, The MIT Press, 3rd ed., 1990.

[164] H. Avila-George, J. Torres-Jimenez, N. Rangel-Valdez, A. Carrión, and V. Hernández, "Supercomputing and grid computing on the verification of covering arrays," *The Journal of Supercomputing*, vol. 62, pp. 916–945, Nov 2012.

[165] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms: generation, enumeration, and search*, vol. 7. CRC press, 1st ed., 1998.

[166] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[167] J. Torres-Jimenez, C. De Alfonso, and V. Hernández, "Computation of ternary covering arrays using a grid," *Applied Computing*, pp. 240–246, 2004.

[168] V. V. Kuliamin and A. A. Petukhov, "A survey of methods for constructing covering arrays," *Programming and Computer Software*, vol. 37, p. 121, May 2011.

[169] R. C. Bryce, Y. Chen, and C. J. Colbourn, "Biased covering arrays for progressive ranking and composition of web services," *International Journal of Simulation and Process Modelling*, vol. 3, no. 1-2, pp. 80–87, 2007.

[170] L. Kampel and D. E. Simos, "Set-based algorithms for combinatorial test set generation," in *Testing Software and Systems*, (Cham), pp. 231–240, Springer International Publishing, 2016.

[171] J. Petke, "Constraints: The future of combinatorial interaction testing," in *2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing*, pp. 17–18, 2015.

[172] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Information and Software Technology*, vol. 94, no. Supplement C, pp. 165 – 185, 2018.

[173] C. T. Cheng, "The test suite generation problem: Optimal instances and their implications," *Discrete Applied Mathematics*, vol. 155, no. 15, pp. 1943 – 1957, 2007.

[174] Y. Lei and K. C. Tai, "In-parameter-order: a test generation strategy for pairwise testing," in *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No.98EX231)*, pp. 254–261, Nov 1998.

[175] M. F. Johansen, Ø. Haugen, and F. Fleurey, *Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible*, pp. 638–652. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

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

[177] K. Meagher and B. Stevens, "Covering arrays on graphs," *Journal of Combinatorial Theory, Series B*, vol. 95, no. 1, pp. 134 – 151, 2005.

[178] C. J. Colbourn, M. B. Cohen, and R. Turban, "A deterministic density algorithm for pairwise interaction coverage.," in *IASTED Conf. on Software Engineering*, pp. 345–352, 2004.

[179] M. Garey, D. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237 – 267, 1976.

[180] K. Kleine and D. E. Simos, "An efficient design and implementation of the in-parameter-order algorithm," *Mathematics in Computer Science*, vol. 12, pp. 51–67, Mar 2018.

[181] L. Kampel, B. Garn, and D. E. Simos, "Covering arrays via set covers," *Electronic Notes in Discrete Mathematics*, vol. 65, pp. 11 – 16, 2018. 7th International Conference on Algebraic Informatics (CAI 2017): Design Theory Track.

[182] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, p. 634–652, jul 1998.

[183] B. Buchberger, "Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal," *J. Symb. Comput.*, vol. 41, pp. 475–511, 2006.

[184] B. Stevens and E. Mendelsohn, "Packing arrays and packing designs," *Designs, Codes and Cryptography*, vol. 27, pp. 165–176, Oct. 2002.

[185] B. Stevens and E. Mendelsohn, "Packing arrays," *Theoretical Computer Science*, vol. 321, no. 1, pp. 125–148, 2004. Latin American Theoretical Informatics.

[186] R. F. Bailey and A. C. Burgess, "Generalized packing designs," *Discrete Mathematics*, vol. 313, no. 11, pp. 1167–1190, 2013. Selected Papers from the 23rd British Combinatorial Conference.

[187] P. Boyvalenkov, T. Marinova, and M. Stoyanova, "Nonexistence of a few binary orthogonal arrays," *Discrete Applied Mathematics*, vol. 217, pp. 144–150, 2017.

[188] MATRIS, "Classification results for CA(52;5,k,2)." Available online `https://matris.sba-research.org/data/balca/results/table_v2_t5_N52`, Accessed on 2024-02-02.

[189] I. Izquierdo-Marquez and J. Torres-Jimenez, "New optimal covering arrays using an orderly algorithm," *Discrete Mathematics, Algorithms and Applications*, vol. 10, no. 01, p. 1850011, 2018.

[190] B. Stevens, *Transversal covers and packings*. PhD thesis, University of Toronto, 1998.

[191] K. A. S. Abdel-Ghaffar and A. El Abbadi, "Optimal disk allocation for partial match queries," *ACM Trans. Database Syst.*, vol. 18, pp. 132–156, Mar. 1993.

[192] M. Banbara, H. Matsunaka, N. Tamura, and K. Inoue, "Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers," in *Logic for Programming, Artificial Intelligence, and Reasoning*, pp. 112–126, 2010.

[193] C. Ansótegui, I. Izquierdo, F. Manyà, and J. Torres Jiménez, "A Max-SAT-based approach to constructing optimal covering arrays," in *Artificial Intelligence Research and Development: Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence*, pp. 51–59, IOS Press, 2013.

[194] A. Yamada, T. Kitamura, C. Artho, E.-H. Choi, Y. Oiwa, and A. Biere, "Optimization of combinatorial testing by incremental SAT solving," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1–10, 2015.

[195] M. B. Cohen, M. B. Dwyer, and J. Shi, "Exploiting constraint solving history to construct interaction test suites," in *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*, pp. 121–132, 2007.

[196] A. Yamada, A. Biere, C. Artho, T. Kitamura, and E.-H. Choi, "Greedy combinatorial test case generation using unsatisfiable cores," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 614–624, 2016.

[197] J. Yan and J. Zhang, "A backtracking search tool for constructing combinatorial test suites," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1681 – 1693, 2008. Selected papers from the 30th Annual International Computer Software and Applications Conference (COMPSAC), Chicago, September 7–21, 2006.

[198] H. Liu, F. Ma, and J. Zhang, "Generating covering arrays with pseudo-Boolean constraint solving and balancing heuristic," in *PRICAI 2016: Trends in Artificial Intelligence*, pp. 262–270, 2016.

[199] S.-W. Gao, J.-H. Lv, B.-L. Du, C. J. Colbourn, and S.-L. Ma, "Balancing frequencies and fault detection in the in-parameter-order algorithm," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 957–968, 2015.

[200] G. Katsirelos, N. Narodytska, and T. Walsh, "On the complexity and completeness of static constraints for breaking row and column symmetry," in *Principles and Practice of Constraint Programming – CP 2010* (D. Cohen, ed.), pp. 305–320, 2010.

[201] P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh, "Breaking row and column symmetries in matrix models," in *Principles and Practice of Constraint Programming - CP 2002* (P. Van Hentenryck, ed.), pp. 462–477, 2002.

[202] J. Yip and P. V. Hentenryck, "Symmetry breaking via lexleader feasibility checkers," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011* (T. Walsh, ed.), pp. 687–692, 2011.

[203] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy, "Symmetry-breaking predicates for search problems," in *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 148–159, 1996.

[204] M. Kiermaier and M. O. Pavčević, "Intersection numbers for subspace designs," *Journal of Combinatorial Designs*, vol. 23, no. 11, pp. 463–480, 2015.

[205] M. Chateauneuf and D. L. Kreher, "On the state of strength-three covering arrays," *Journal of Combinatorial Designs*, vol. 10, no. 4, pp. 217–238, 2002.

[206] G. Katona, "Two applications (for search theory and truth functions) of Sperner type theorems," *Periodica Mathematica Hungarica*, vol. 3, pp. 19–26, 1973.

[207] 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)*, vol. 1, pp. 385–394, 2006.

[208] MATRIS, "Classification of balanced covering arrays." Available online https://matris.sba-research.org/data/balca, Accessed on 2024-02-02.

[209] M. Gebser, B. Kaufmann, and T. Schaub, "Conflict-driven answer set solving: From theory to practice," *Artificial Intelligence*, vol. 187-188, pp. 52–89, 2012.

[210] MATRIS, "Classification results for CA(12;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t3_N12, Accessed on 2024-02-02.

[211] MATRIS, "Classification results for CA(13;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t3_N13, Accessed on 2024-02-02.

[212] MATRIS, "Classification results for CA(14;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t3_N14, Accessed on 2024-02-02.

[213] MATRIS, "Classification results for CA(15;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t3_N15, Accessed on 2024-02-02.

[214] MATRIS, "Classification results for CA(16;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t3_N16, Accessed on 2024-02-02.

[215] MATRIS, "Classification results for CA(11;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/classification_v2_t3_N11, Accessed on 2024-02-02.

[216] MATRIS, "Classification results for CA(12;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/classification_v2_t3_N12, Accessed on 2024-02-02.

[217] MATRIS, "Classification results for CA(18;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N18, Accessed on 2024-02-02.

[218] MATRIS, "Classification results for CA(19;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N19, Accessed on 2024-02-02.

[219] MATRIS, "Classification results for CA(20;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N20, Accessed on 2024-02-02.

[220] MATRIS, "Classification results for CA(21;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N21, Accessed on 2024-02-02.

[221] MATRIS, "Classification results for CA(22;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N22, Accessed on 2024-02-02.

[222] MATRIS, "Classification results for CA(23;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N23, Accessed on 2024-02-02.

[223] MATRIS, "Classification results for CA(24;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N24, Accessed on 2024-02-02.

[224] MATRIS, "Classification results for CA(25;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/table_v2_t4_N25, Accessed on 2024-02-02.

[225] E. Rains, N. Sloane, and J. Stufken, "The lattice of N-run orthogonal arrays," *Journal of Statistical Planning and Inference*, vol. 102, no. 2, pp. 477–500, 2002.

[226] I. Izquierdo-Marquez and J. Torres-Jimenez, "New covering array numbers," *Applied Mathematics and Computation*, vol. 353, pp. 134–146, 2019.

[227] MATRIS, "Classification of balanced covering arrays with a ternary alphabet." Available online https://matris.sba-research.org/data/balca/results#ternary, Accessed on 2024-02-02.

[228] MATRIS, "Classification results for CA(14;3,k,2)." Available online https://matris.sba-research.org/data/balca/results/classification_v2_t3_N14, Accessed on 2024-02-02.

[229] MATRIS, "Classification results for CA(24;4,k,2)." Available online https://matris.sba-research.org/data/balca/results/classification_v2_t4_N24, Accessed on 2024-02-02.

[230] MATRIS, "Classification results for CA(11;2,k,3)." Available online https://matris.sba-research.org/data/balca/results/classification_v3_t2_N11, Accessed on 2024-02-02.

[231] MATRIS, "Classification results for CA(12;2,k,3)." Available online https://matris.sba-research.org/data/balca/results/classification_v3_t2_N12, Accessed on 2024-02-02.

[232] MATRIS, "Classification results for CA(8;2,k,2)." Available online https://matris.sba-research.org/data/balca/results/classification_v2_t2_N8, Accessed on 2024-02-02.

[233] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, 1978.

[234] W. Wang, S. Sampath, Y. Lei, and R. Kacker, "An interaction-based test sequence generation approach for testing web applications," in *2008 11th IEEE High Assurance Systems Engineering Symposium*, pp. 209–218, 2008.

[235] I. Simon, "Piecewise testable events," in *Automata Theory and Formal Languages: 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pp. 214–222, Springer, 1975.

[236] M. Rahman, R. R. Othman, R. B. Ahmad, and M. M. Rahman, "Event driven input sequence t-way test strategy using simulated annealing," in *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, pp. 663–667, 2014.

[237] T. Kameda and P. Weiner, "On the state minimization of nondeterministic finite automata," *IEEE Transactions on Computers*, vol. C-19, no. 7, pp. 617–627, 1970.

[238] T. Jiang and B. Ravikumar, "Minimal NFA problems are hard," *SIAM Journal on Computing*, vol. 22, no. 6, pp. 1117–1141, 1993.

[239] J. E. Pin and J. Sakarovitch, "Some operations and transductions that preserve rationality," in *Theoretical Computer Science*, (Berlin, Heidelberg), pp. 277–288, Springer Berlin Heidelberg, 1982.

[240] H. Basold, H. H. Hansen, J.-É. Pin, and J. Rutten, "Newton series, coinductively," in *Theoretical Aspects of Computing - ICTAC 2015*, (Cham), pp. 91–109, Springer International Publishing, 2015.

[241] L. Kari, "Deletion operations: closure properties," *International Journal of Computer Mathematics*, vol. 52, no. 1-2, pp. 23–42, 1994.

[242] J. Sakarovitch, *Elements of automata theory*. New York, USA: Cambridge University Press, 2009.

[243] M. Droste and W. Kuich, "Semirings and formal power series," in *Handbook of Weighted Automata* (M. Droste, W. Kuich, and H. Vogler, eds.), pp. 3–28, Springer Berlin Heidelberg, 2009.

[244] M. Droste and W. Kuich, "Semirings and formal power series," in *Handbook of Weighted Automata* (M. Droste, W. Kuich, and H. Vogler, eds.), pp. 69–104, Springer Berlin Heidelberg, 2009.

[245] M. Wagner, L. Kampel, and D. E. Simos, "Quantum-inspired evolutionary algorithms for covering arrays of arbitrary strength," in *Analysis of Experimental Algorithms*, (Cham), pp. 300–316, Springer International Publishing, 2019.

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

[247] M. Hifi, V. T. Paschos, and V. Zissimopoulos, "A neural network for the minimum set covering problem," *Chaos, Solitons & Fractals*, vol. 11, no. 13, pp. 2079 – 2089, 2000.

[248] D. Kaznachey and A. Jagota, "Approximating minimum set cover in a Hopfield-style network," *Information Sciences*, vol. 98, no. 1, pp. 203 – 216, 1997.

[249] I. H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Annals of Operations Research*, vol. 63, pp. 511–623, Oct 1996.

[250] K. A. Smith, "Neural networks for combinatorial optimization: A review of more than a decade of research," *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 15–34, 1999.

[251] E. B. Olsen, "RNS hardware matrix multiplier for high precision neural network acceleration: "RNS TPU"," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2018.

[252] Y. Zuo, B. Li, Y. Zhao, Y. Jiang, Y.-C. Chen, P. Chen, G.-B. Jo, J. Liu, and S. Du, "All-optical neural network with nonlinear activation functions," *Optica*, vol. 6, pp. 1132–1137, Sep 2019.

[253] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017. Publisher: Nature Publishing Group.

[254] X. Yao, "Finding approximate solutions to NP-hard problems by neural networks is hard," *Information Processing Letters*, vol. 41, no. 2, pp. 93 – 98, 1992.

[255] J.-Y. Potvin and K. A. Smith, "Artificial neural networks for combinatorial optimization," in *Handbook of Metaheuristics* (F. Glover and G. A. Kochenberger, eds.), pp. 429–455, Boston, MA: Springer US, 2003.

[256] M. Kajiura, Y. Akiyama, and Y. Anzai, "Solving large scale puzzles with neural networks," in *IEEE International Workshop on Tools for Artificial Intelligence*, (Los Alamitos, CA, USA), pp. 562–569, IEEE Computer Society, oct 1989.

[257] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems 30*, pp. 6348–6358, Curran Associates, Inc., 2017.

[258] T.-W. Yue and Z.-C. Lee, "Sudoku solver by Q'tron neural networks," in *Intelligent Computing*, (Berlin, Heidelberg), pp. 943–952, Springer Berlin Heidelberg, 2006.

[259] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines*. John Wiley & SonsNew York, NY, 1988.

[260] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

[261] V. V. Vazirani, *Approximation algorithms*. Springer Berlin Heidelberg, 2nd ed., 2003.

[262] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[263] M. Lundy and A. Mees, "Convergence of an annealing algorithm," *Math. Program.*, vol. 34, no. 1, pp. 111–124, 1986.

[264] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[265] M. Bashiri and A. Farshbaf Geranmayeh, "Tuning the parameters of an artificial neural network using central composite design and genetic algorithm," *Scientia Iranica*, vol. 18, no. 6, pp. 1600–1608, 2011.

[266] H. Pérez-Espinosa, H. Avila-George, J. Rodriguez-Jacobo, H. A. Cruz-Mendoza, J. Martínez-Miranda, and I. Espinosa-Curiel, "Tuning the parameters of a convolutional artificial neural network by using covering arrays," *Research in Computing Science*, vol. 121, pp. 69–81, 2016.

[267] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "DeepCT: Tomographic combinatorial testing for deep learning systems," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 614–618, 2019.

[268] L. Yu, F. Duan, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Constraint handling in combinatorial test generation using forbidden tuples," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 1–9, 2015.

[269] 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)*, pp. 69–76, Nov 2015.

[270] 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)*, pp. 69–79, March 2017.

[271] National Institute of Standards and Technology (NIST), "Covering arrays generated by IPOG-F." Available online at `https://math.nist.gov/coveringarrays/ipof/ipof-results.html`, Accessed on 2024-01-11.

[272] D. R. Kuhn, R. N. Kacker, Y. Lei, and D. E. Simos, "Combinatorial methods for explainable AI," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 167–170, 2020.

[273] J. Chandrasekaran, Y. Lei, R. Kacker, and D. Richard Kuhn, "A combinatorial approach to testing deep neural network-based autonomous driving systems," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 57–66, 2021.

[274] R. Smith, D. Jarman, J. Bellows, R. Kuhn, R. Kacker, and D. Simos, "Measuring combinatorial coverage at Adobe," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 194–197, 2019.

[275] M. Ozcan, "Applications of practical combinatorial testing methods at Siemens industry inc., building technologies division," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 208–215, 2017.

[276] R. C. Bryce, Y. Chen, and C. J. Colbourn, "Biased covering arrays for progressive ranking and composition of Web Services," *International Journal of Simulation and Process Modelling*, July 2007. Publisher: Inderscience Publishers.

[277] B. Garn, D. Sebastian Lang, M. Leithner, D. Richard Kuhn, R. Kacker, and D. E. Simos, "Combinatorially XSSing web application firewalls," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 85–94, 2021.

[278] D. E. Simos, J. Zivanovic, and M. Leithner, "Automated combinatorial testing for detecting sql vulnerabilities in web applications," in *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pp. 55–61, 2019.

[279] D. E. Simos, J. Bozic, F. Duan, B. Garn, K. Kleine, Y. Lei, and F. Wotawa, "Testing TLS using combinatorial methods and execution framework," in *Testing Software and Systems*, Lecture Notes in Computer Science, (Cham), pp. 162–177, Springer International Publishing, 2017.

[280] B. Garn, D. E. Simos, F. Duan, Y. Lei, J. Bozic, and F. Wotawa, "Weighted Combinatorial Sequence Testing for the TLS Protocol," in *2019 IEEE International*

Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 46–51, Apr. 2019.

[281] S. P. Katragadda, T. Cody, P. Beling, and L. Freeman, "Active learning with combinatorial coverage," in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1129–1136, 2022.

[282] T. Cody, E. Lanus, D. D. Doyle, and L. Freeman, "Systematic training and testing for machine learning using combinatorial interaction testing," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 102–109, 2022.

[283] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits." Available online at `http://yann.lecun.com/exdb/mnist/`, 1998.

[284] A. R. Patel, J. Chandrasekaran, Y. Lei, R. N. Kacker, and D. R. Kuhn, "A combinatorial approach to fairness testing of machine learning models," in *2022 IEEE Int. Conf. on Software Testing, Verification and Validation Workshops*, pp. 94–101, 2022.

[285] Centre of Regional Science, Vienna University of Technology, "Smart cities, ranking of European medium-sized cities." Available online at `http://www.smart-cities.eu/download/smart_cities_final_report.pdf`, Accessed on 2023-11-06, 2007.

[286] N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann, "The national human activity pattern survey (NHAPS): a resource for assessing exposure to environmental pollutants," *Journal of Exposure Science and Environmental Epidemiology*, vol. 11, no. 3, p. 231, 2001.

[287] B. Diffey, "An overview analysis of the time people spend outdoors," *British Journal of Dermatology*, vol. 164, no. 4, pp. 848–854, 2011.

[288] Z. A. Baig, P. Szewczyk, C. Valli, P. Rabadia, P. Hannay, M. Chernyshev, M. Johnstone, P. Kerai, A. Ibrahim, K. Sansurooah, N. Syed, and M. Peacock, "Future challenges for smart cities: Cyber-security and digital forensics," *Digital Investigation*, vol. 22, pp. 3–13, 2017.

[289] ISO, "Iso 16484-5:2007 - building automation and control systems — part 5: Data communication protocol." Available online at `https://www.iso.org/standard/44998.html`, 2023. Accessed on 2023-08-12.

[290] M. Ozcan, L. Kampel, and D. E. Simos, "Internal Communication - Personal correspondence between Murat Ozcan and Ludwig Kampel," 2017-10-27.

[291] J. Kaur, J. Tonejc, S. Wendzel, and M. Meier, "Securing BACnet's pitfalls," in *ICT Systems Security and Privacy Protection*, (Cham), pp. 616–629, Springer International Publishing, 2015.

[292] S. T. Bushby and H. M. Newman, "BACnet Today - Significant New Features And Future Enhancements," *ASHRAE journal*, vol. 10, pp. 10–18, 2002.

[293] S. T. Bushby, "BACnetTM: a standard communication infrastructure for intelligent buildings," *Automation in Construction*, vol. 6, no. 5, pp. 529–540, 1997.

[294] B. T. D. Siemens Industry, Inc., "Insight–advanced workstation." Available online at `https://www.iso.org/standard/44998.html`. Document No. 149-333P25, Accessed on 2024-06-17.

[295] M. Ozcan and L. Kampel, "Internal Communication - Personal correspondence between Murat Ozcan and Ludwig Kampel," 2018-06-08.

[296] National Institute of Standards and Technology (NIST), "ACTS software download." Available online at `https://csrc.nist.rip/groups/SNS/acts/documents/comparison-report.html`, Accessed on 2023-08-13.

[297] L. S. G. Ghandehari, M. N. Bourazjany, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Applying combinatorial testing to the Siemens suite," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp. 362–371, March 2013.

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

[299] S. Reid, "An empirical analysis of equivalence partitioning, boundary value analysis and random testing," in *Proceedings Fourth International Software Metrics Symposium*, pp. 64–73, 1997.

[300] S. Martirosyan and v. T. Trung, "On t-covering arrays," *Designs, Codes and Cryptography*, vol. 32, pp. 323–339, May 2004.

[301] MATRIS, "MATRIS large CA index." Available online `https://matris.sba-research.org/data/adobe/`, Accessed on 2022-08-11.

[302] C. H. Hoffmann, "Is AI intelligent? An assessment of artificial intelligence, 70 years after Turing," *Technology in Society*, vol. 68, p. 101893, 2022.

[303] European Commission, "Ethics guidelines for trustworthy AI." Available online at `https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai`, 2019. Accessed 2023-04-24.

[304] High-Level Expert Group on Artificial Intelligence (AI HLEG), "The assessment list for trustworthy artificial intelligence (ALTAI)." Available online at https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=68342, 2020. Accessed 2023-04-24.

[305] K. Sanderson, "GPT-4 is here: what scientists think," *Nature*, vol. 615, no. 7954, p. 773, 2023.

[306] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 3645–3650, Association for Computational Linguistics, July 2019.

[307] D. Khashabi, T. Khot, and A. Sabharwal, "More bang for your buck: Natural perturbation for robust question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Online), pp. 163–170, Association for Computational Linguistics, Nov. 2020.

[308] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "BoolQ: Exploring the surprising difficulty of natural yes/no questions," in *Proc. of the 2019 Conf. of the North American Chapter of the Assoc. for Comp. Linguistics: Human Language Technologies, Vol 1*, (Minneapolis, Minnesota), pp. 2924–2936, Assoc. for Comp. Linguistics, June 2019.

[309] F. Wotawa, "On the use of available testing methods for verification & validation of AI-based software and systems," *CEUR Workshop Proceedings*, vol. 2808, 2021.

[310] E.-S. Jung, S.-Y. Dong, and S.-Y. Lee, "Neural correlates of variations in human trust in human-like machines during non-reciprocal interactions," *Scientific Reports*, vol. 9, no. 1, p. 9975, 2019.

[311] E. J. De Visser, S. S. Monfort, R. McKendrick, M. A. Smith, P. E. McKnight, F. Krueger, and R. Parasuraman, "Almost human: Anthropomorphism increases trust resilience in cognitive agents," *Journal of Experimental Psychology: Applied*, vol. 22, no. 3, pp. 331–349, 2016.

[312] P. Dunn, "The importance of consistency in establishing cognitive-based trust: A laboratory experiment," *Teaching Business Ethics*, vol. 4, pp. 285–306, 2022.

[313] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with GPT-4." Available online at https://doi.org/10.48550/arXiv.2303.12712, Apr. 2023.

[314] M. Jang and T. Lukasiewicz, "Consistency analysis of ChatGPT," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, (Singapore), pp. 15970–15985, Association for Computational Linguistics, Dec. 2023.

[315] M. Gardner, Y. Artzi, V. Basmova, J. Berant, B. Bogin, S. Chen, P. Dasigi, D. Dua, Y. Elazar, A. Gottumukkala, N. Gupta, H. Hajishirzi, G. Ilharco, D. Khashabi, K. Lin, J. Liu, N. Liu, P. Mulcaire, Q. Ning, S. Singh, N. Smith, S. Subramanian, R. Tsarfaty, E. Wallace, A. Zhang, and B. Zhou, "Evaluating models' local decision boundaries via contrast sets," in *Findings of the Association for Computational Linguistics Findings of ACL*, Findings of the Association for Computational Linguistics Findings of ACL: EMNLP 2020, pp. 1307–1323, Association for Computational Linguistics (ACL), 2020.

[316] E. Ruane, T. Faure, R. Smith, D. Bean, J. Carson-Berndsen, and A. Ventresque, "BoTest: A Framework to Test the Quality of Conversational Agents Using Divergent Input Examples," in *Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion*, IUI '18 Companion, (New York, NY, USA), Association for Computing Machinery, 2018.

[317] J. Guichard, E. Ruane, R. Smith, D. Bean, and A. Ventresque, "Assessing the robustness of conversational agents using paraphrases," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 55–62, 2019.

[318] X. Liu, T. Pang, and C. Fan, "Federated prompting and chain-of-thought reasoning for improving llms answering," in *Knowledge Science, Engineering and Management*, (Cham), pp. 3–11, Springer Nature Switzerland, 2023.

[319] A. Nighojkar and J. Licato, "Improving Paraphrase Detection with the Adversarial Paraphrasing Task," in *Proc. of the 59th Annual Meeting of the Assoc. for Comp. Linguistics and the 11th Int. Joint Conf. on Natural Language Processing*, p. 7106–7116, 2021.

[320] J. Ganitkevitch, B. Van Durme, and C. Callison-Burch, "PPDB: The paraphrase database," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, (Atlanta, Georgia), pp. 758–764, Association for Computational Linguistics, 2013.

[321] Z. Li, X. Jiang, L. Shang, and H. Li, "Paraphrase Generation with Deep Reinforcement Learning," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 3865–3878, Assoc. for Computational Linguistics, Oct.-Nov. 2018.

[322] J. Božić, "Ontology-based metamorphic testing for chatbots," *Software Quality Journal*, vol. 30, no. 1, pp. 227–251, 2022.

[323] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "Why Johnny can't prompt: How non-AI experts try (and fail) to design LLM prompts," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[324] Y. Li, "A practical survey on zero-shot prompt design for in-context learning," in *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, (Varna, Bulgaria), pp. 641–647, INCOMA Ltd., Shoumen, Bulgaria, Sept. 2023.

[325] MaTRIS, "CAgen." Available online at `https://matris.sba-research.org/tools/cagen`, Accessed on 2023-09-18.

[326] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models." Available online at `https://doi.org/10.48550/arXiv.2302.13971`, 2023.

[327] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[328] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, (Volume 1: long papers)*, (Vancouver, Canada), pp. 1601–1611, Association for Computational Linguistics, July 2017.

[329] Y. Liu, Y. Xie, and A. Srivastava, "Neural Trojans," in *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 45–48, 2017.

[330] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[331] N. Carlini, M. Jagielski, C. Choquette-Choo, D. Paleka, W. Pearce, H. Anderson, A. Terzis, K. Thomas, and F. Tramèr, "Poisoning web-scale training datasets is practical," in *2024 IEEE Symposium on Security and Privacy (SP)*, (Los Alamitos, CA, USA), pp. 179–179, IEEE Computer Society, may 2024.

[332] E. Creamer, "Authors file a lawsuit against OpenAI for unlawfully 'ingesting' their books." Available online at `https://www.theguardian.com/books/2023/jul/05/authors-file-a-lawsuit-against-openai-for-unlawfully-ingesting-their-b` July 2023. Accessed 21-08-2023.

[333] L. Joseph Saveri Law Firm, "Complaint, case 3:23-cv-03223." Available online at `https://llmlitigation.com/pdf/03223/tremblay-openai-complaint.pdf`, 2023. Accessed 21-08-2023.

[334] L. Joseph Saveri Law Firm, "Complaint, case 3:22-cv-06823." Available online at https://githubcopilotlitigation.com/pdf/06823/1-0-github_complaint.pdf, 2023. Accessed 21-08-2023.

[335] L. Sh. Ghandehari, Y. Lei, R. Kacker, R. Kuhn, T. Xie, and D. Kung, "A combinatorial testing-based approach to fault localization," *IEEE Transactions on Software Engineering*, vol. 46, no. 6, pp. 616–645, 2020.

[336] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact.," *Empirical Software Engineering: An International Journal*, vol. 10, no. 4, pp. 405–435, 2005.

[337] S. Khurshid, D. Marinov, G. Rothermel, T. Xie, W. Motycka, M. B. Dwyer, S. Elbaum, J. Hatcliff, H. Do, and A. Kinneer, "Software-artifact Infrastructure Repository: Home." Available online at http://sir.csc.ncsu.edu/portal/index.php. Accessed on 2023-09-05.

[338] B. Garn, M. Radavelli, A. Gargantini, M. Leithner, and D. E. Simos, "A fault-driven combinatorial process for model evolution in XSS vulnerability detection," in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, (Cham), pp. 207–215, Springer International Publishing, 2019.

[339] D. Blue, A. Hicks, R. Rawlins, and R. Tzoref-Brill, "Practical fault localization with combinatorial test design," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 268–271, 2019.

[340] I. Research, "IBM Functional Coverage Unified Solution - IBM." Available online at https://researcher.watson.ibm.com/researcher/view_group.php?id=1871, July 2016.

[341] R. Compton, M. T. Mehari, C. J. Colbourn, E. De Poorter, and V. R. Syrotiuk, "Screening interacting factors in a wireless network testbed using locating arrays," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, (San Francisco, CA, USA), pp. 650–655, IEEE, Apr. 2016.

[342] Y. Akhtar, F. Zhang, C. J. Colbourn, J. Stufken, and V. R. Syrotiuk, "Scalable level-wise screening experiments using locating arrays," *Journal of Quality Technology*, vol. 55, no. 5, pp. 584–597, 2023.

[343] A. G. Voyiatzis, K. G. Stefanidis, and P. Kitsos, "Efficient triggering of Trojan hardware logic," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pp. 1–6, 2016.

[344] X. Chuan, Y. Yan, and Y. Zhang, "An efficient triggering method of hardware Trojan in AES cryptographic circuit," in *2017 2nd IEEE International Conference on Integrated Circuits and Microsystems (ICICM)*, pp. 91–95, 2017.

[345] J. Chandrasekaran, Y. Lei, R. Kacker, and D. Richard Kuhn, "A combinatorial approach to explaining image classifiers," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 35–43, 2021.

[346] E. Lanus, L. J. Freeman, D. Richard Kuhn, and R. N. Kacker, "Combinatorial testing metrics for machine learning," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 81–84, 2021.

[347] J. Chandrasekaran, A. R. Patel, Y. Lei, R. Kacker, and D. R. Kuhn, "Evaluation of t-way testing of DNNs in autonomous driving systems," in *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pp. 17–18, 2021.

[348] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, (New York, NY, USA), p. 303–314, Association for Computing Machinery, 2018.

[349] Y. Li, J. Tao, and F. Wotawa, "Ontology-based test generation for automated and autonomous driving functions," *Information and Software Technology*, vol. 117, p. 106200, Jan. 2020.

[350] H. Shu, H. Lv, K. Liu, K. Yuan, and X. Tang, "Test scenarios construction based on combinatorial testing strategy for automated vehicles," *IEEE Access*, vol. 9, pp. 115019–115029, 2021.

[351] G. Dhadyalla, N. Kumari, and T. Snell, "Combinatorial testing for an automotive hybrid electric vehicle control system: A case study," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pp. 51–57, 2014.

[352] G. Dhadyalla, C. P. Yang, J. Marco, and P. Jennings, "Real-time sequence testing of an automotive electric machine control systems," tech. rep., SAE Technical Paper 2018-01-0004, 2018.

[353] R. George, "Why we should worry about the supply chain," *Int. J. Crit. Infrastruct. Prot.*, vol. 11, p. 22–23, Dec. 2015.

[354] C. Ruppersberger and M. Rogers, "Investigative report on the US national security issues posed by Chinese telecommunications companies Huawei and ZTE." Available online at `https://intelligence.house.gov/sites/democrats.intelligence.house.gov/files/huawei-zte%20investigative%20report%20(final)_0.pdf`, Oct. 8 2012. Accessed on 2023-09-05.

[355] S. Mitra, H.-S. P. Wong, and S. Wong, "The Trojan-proof chip," *IEEE Spectrum*, vol. 52, no. 2, pp. 46–51, 2015.

[356] Z. Abbany, "Has Germany's Patriot missile system been hacked?." Available online at https://p.dw.com/p/1FvEy, July 8 2015. Accessed on 2023-09-05.

[357] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Cryptographic Hardware and Embedded Systems – CHES 2012*, (Berlin, Heidelberg), pp. 23–40, Springer Berlin Heidelberg, 2012.

[358] C. T. Lopez, "DOD Adopts 'Zero Trust' Approach to Buying Microelectronics." Available online at https://www.defense.gov/News/News-Stories/Article/Article/2192120/dod-adopts-zero-trust-approach-to-buying-microelectronics/, May 19, 2020. Accessed on 2023-09-05.

[359] S. Garg, "Inspiring trust in outsourced integrated circuit fabrication," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 1128–1228, 2017.

[360] Y. Liu, C. Bao, Y. Xie, and A. Srivastava, "Introducing TFUE: The trusted foundry and untrusted employee model in IC supply chain security," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.

[361] J. Dofe, J. Frey, and Q. Yu, "Hardware security assurance in emerging IoT applications," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2050–2053, May 2016.

[362] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and internet-of-things systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 9–20, 2018.

[363] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware Trojan: Threats and emerging solutions," in *2009 IEEE International High Level Design Validation and Test Workshop*, pp. 166–171, 2009.

[364] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "FPGA Trojans through detecting and weakening of cryptographic primitives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, pp. 1236–1249, Aug 2015.

[365] A. Jain and U. Guin, "A novel tampering attack on AES cores with hardware Trojans," in *2020 IEEE International Test Conference in Asia (ITC-Asia)*, pp. 77–82, 2020.

[366] S. Ghandali, T. Moos, A. Moradi, and C. Paar, "Side-channel hardware Trojan for provably-secure SCA-protected implementations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1435–1448, 2020.

[367] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware Trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, pp. 85–102, 2017.

[368] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[369] H. Salmani, "Hardware Trojan attacks and countermeasures," in *Fundamentals of IP and SoC Security: Design, Verification, and Debug*, pp. 247–276, Cham: Springer International Publishing, 2017.

[370] H. Li, Q. Liu, and J. Zhang, "A survey of hardware Trojan threat and defense," *Integration*, vol. 55, pp. 426 – 437, 2016.

[371] S. R. Rajendran, R. Mukherjee, and R. S. Chakraborty, "SoK: Physical and logic testing techniques for hardware Trojan detection," in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, ASHES'20, (New York, NY, USA), pp. 103–116, Association for Computing Machinery, 2020.

[372] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware Trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10796–10826, 2020.

[373] trust-HUB.org, "Chip-level Trojan Benchmarks." Available online at `https://www.trust-hub.org/#/benchmarks/chip-level-trojan`. Accessed on 2023-09-05.

[374] L. Pyrgas and P. Kitsos, "A hybrid FPGA Trojan detection technique based-on combinatorial testing and on-chip sensing," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, (Cham), pp. 294–303, Springer International Publishing, 2018.

[375] A. P. Fournaris, L. Pyrgas, and P. Kitsos, "An efficient multi-parameter approach for FPGA hardware Trojan detection," *Microprocessors and Microsystems*, vol. 71, p. 102863, 2019.

[376] Y. Huang, S. Bhunia, and P. Mishra, "MERS: Statistical test generation for side-channel analysis based Trojan detection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 130–141, Association for Computing Machinery, 2016.

[377] C. Nigh and A. Orailoglu, "AdaTrust: Combinational hardware Trojan detection through adaptive test pattern construction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 544–557, 2021.

[378] S. Dupuis, M. Flottes, G. Di Natale, and B. Rouzeyre, "Protection against hardware Trojans with logic testing: Proposed solutions and challenges ahead," *IEEE Design Test*, vol. 35, no. 2, pp. 73–90, 2018.

[379] M. Flottes, S. Dupuis, P. Ba, and B. Rouzeyre, "On the limitations of logic testing for detecting hardware Trojans horses," in *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pp. 1–5, 2015.

[380] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards Trojan-free trusted ICs: Problem analysis and detection scheme," in *2008 Design, Automation and Test in Europe*, pp. 1362–1365, 2008.

[381] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "MERO: A statistical approach for hardware Trojan detection," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, (Berlin, Heidelberg), pp. 396–410, Springer Berlin Heidelberg, 2009.

[382] M. A. Nourian, M. Fazeli, and D. Hély, "Hardware Trojan detection using an advised genetic algorithm based logic testing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 461–470, 2018.

[383] H. Salmani, "COTD: Reference-free hardware Trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 338–350, 2017.

[384] K. Huang and Y. He, "Trigger identification using difference-amplified controllability and dynamic transition probability for hardware Trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3387–3400, 2020.

[385] S. A. Islam, F. Islam Mime, S. M. Asaduzzaman, and F. Islam, "Socio-network analysis of RTL designs for hardware Trojan localization," in *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, pp. 1–6, 2019.

[386] S. Dupuis, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Identification of Hardware Trojans triggering signals," in *First Workshop on Trustworthy Manufacturing and Utilization of Secure Devices*, (Avignon, France), May 2013.

[387] D. E. Simos, R. Kuhn, A. G. Voyiatzis, and R. Kacker, "Combinatorial methods in security testing," *Computer*, vol. 49, pp. 80–83, oct 2016.

[388] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Advances in Cryptology — ASIACRYPT 2001* (C. Boyd, ed.), (Berlin, Heidelberg), pp. 239–254, Springer Berlin Heidelberg, 2001.

[389] Siemens, "ModelSim." Available online at `https://www.mentor.com/products/fv/modelsim/`. Accessed on 2023-09-05.

[390] Vivado. Available online at `https://www.xilinx.com/video/hardware/hardware-co-simulation-vivado-system-generator-for-dsp.html`. Accessed on 2023-09-05.

[391] MaTRIS, "HT location arrays." Available online at https://matris.sba-research.org/data/HTlocation/. Accessed on 2023-09-05.

[392] N. Lesperance, S. Kulkarni, and Kwang-Ting Cheng, "Hardware Trojan detection using exhaustive testing of k-bit subspaces," in *The 20th Asia and South Pacific Design Automation Conference*, pp. 755–760, 2015.

[393] C. J. Colbourn and V. R. Syrotiuk, "Coverage, location, detection, and measurement," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 19–25, 2016.

[394] G. Lugano, "Virtual assistants and self-driving cars," in *2017 15th International Conference on ITS Telecommunications (ITST)*, pp. 1–5, 2017.

[395] F. Wotawa, "On the importance of system testing for assuring safety of AI systems," *CEUR Workshop Proceedings*, vol. 2419, 2019. 2019 Workshop on Artificial Intelligence Safety, AISafety 2019;.

[396] D. Gunning, "Explainable artificial intelligence (XAI)," *Defense Advanced Research Projects Agency (DARPA)*, 2017.

[397] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "XAI—explainable artificial intelligence," *Science Robotics*, vol. 4, no. 37, 2019.

[398] A. M. Diaz, "Toward a regulatory framework to certify artificial intelligence applications in aviation." Available online at https://doi.org/10.25967/610347, Oct. 2023. Deutscher Luft- und Raumfahrtkongress 2023, Stuttgart.

[399] K. Hayhurst and D. Veerhusen, "A practical approach to modified condition/decision coverage," in *20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219)*, vol. 1, pp. 1B2/1–1B2/10 vol.1, 2001.

[400] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, (New York, NY, USA), p. 851–862, Association for Computing Machinery, 2020.

[401] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, pp. 4765–4774, Curran Associates, Inc., 2017.

[402] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 1135–1144, Association for Computing Machinery, 2016.

[403] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018.

[404] O. Biran and C. Cotton, "Explanation and justification in machine learning: A survey," *IJCAI-17 workshop on explainable AI (XAI)*, vol. 8, no. 1, pp. 8–13, 2017.

[405] D. Alvarez Melis and T. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018.

[406] G. Kasparov, *Deep thinking: where machine intelligence ends and human creativity begins*. Hachette UK, 2017.

[407] H. Salehi and R. Burgueño, "Emerging artificial intelligence methods in structural engineering," *Engineering Structures*, vol. 171, pp. 170–189, 2018.

[408] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 0210–0215, IEEE, 2018.

[409] T. Miller, P. Howe, and L. Sonenberg, "Explainable ai: Beware of inmates running the asylum or: How i learnt to stop worrying and love the social and behavioural sciences." Available online at `https://doi.org/10.48550/arXiv.1712.00547`, 2017.

[410] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (XAI): Toward medical XAI," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 11, pp. 4793–4813, 2021.

[411] Y. Zhang and X. Chen, "Explainable recommendation: A survey and new perspectives," *Found. Trends Inf. Retr.*, vol. 14, p. 1–101, mar 2020.

[412] P. J. Phillips, C. Hahn, P. Fontana, A. Yates, K. K. Greene, D. Broniatowski, and M. A. Przybocki, "Four principles of explainable artificial intelligence." National Institute of Standards and Technology Interagency/Internal Report (NISTIR), available online at `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=933399`, 2021.

[413] U. Ehsan, P. Tambwekar, L. Chan, B. Harrison, and M. O. Riedl, "Automated rationale generation: a technique for explainable AI and its effects on human perceptions," in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 263–274, 2019.

[414] D. J. Hilton, "Conversational processes and causal explanation.," *Psychological Bulletin*, vol. 107, no. 1, p. 65, 1990.

[415] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[416] C. H. Papadimitriou, "The euclidean travelling salesman problem is NP-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237 – 244, 1977.

[417] L. Stockmeyer, "Planar 3-colorability is polynomial complete," *ACM Sigact News*, vol. 5, no. 3, pp. 19–25, 1973.

[418] D. Kuhn, R. Kacker, and Y. Lei, "Practical combinatorial testing." NIST Special Publication 800-142, Available online at `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf`, 2010. DOI: 10.6028/NIST.SP.800-142.

[419] D. R. Mandel, D. J. Hilton, and P. E. Catellani, *The psychology of counterfactual thinking.* Routledge, 2005.

[420] A. Artelt and B. Hammer, "On the computation of counterfactual explanations - A survey," 2019.

[421] A. Artelt, V. Vaquet, R. Velioglu, F. Hinder, J. Brinkrolf, M. Schilling, and B. Hammer, "Evaluating robustness of counterfactual explanations," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 01–09, 2021.

[422] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the GDPR," *Harvard Journal of Law & Technology*, vol. 31 (2), 2018.

[423] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information.," *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[424] A. Baddeley, "The magical number seven: Still magic after all these years?," *Psychological Review*, no. 101(2), pp. 353–356, 1994.

[425] W. Wachenfeld and H. Winner, "The release of autonomous vehicles," in *Autonomous Driving: Technical, Legal and Social Aspects*, pp. 425–449, Springer Berlin Heidelberg, 2016.

[426] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[427] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[428] F. Schuldt, A. Reschka, and M. Maurer, "A method for an efficient, systematic test case generation for advanced driver assistance systems in virtual environments," in *Automotive Systems Engineering II*, pp. 147–175, Cham: Springer International Publishing, 2018.

[429] T. Menzel, G. Bagschik, and M. Maurer, "Scenarios for development, test and validation of automated vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1821–1827, 2018.

[430] F. Wotawa, "Testing autonomous and highly configurable systems: Challenges and feasible solutions," in *Automated Driving: Safer and More Efficient Future Driving*, (Cham), pp. 519–532, Springer International Publishing, 2017.

[431] F. Klück, Y. Li, M. Nica, J. Tao, and F. Wotawa, "Using ontologies for test suites generation for automated and autonomous driving functions," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 118–123, 2018.

[432] J. Tao, Y. Li, F. Wotawa, H. Felbinger, and M. Nica, "On the industrial application of combinatorial testing for autonomous driving functions," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 234–240, 2019.

[433] H. Felbinger, F. Kluck, Y. Li, M. Nica, J. Tao, F. Wotawa, and M. Zimmermann, "Comparing two systematic approaches for testing automated driving functions," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, (Graz, Austria), pp. 1–6, IEEE, Nov. 2019.

[434] F. Klück, M. Zimmermann, F. Wotawa, and M. Nica, "Genetic algorithm-based test parameter optimization for adas system testing," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 418–425, 2019.

[435] F. Klück, Y. Li, J. Tao, and F. Wotawa, "An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems," *Information and Software Technology*, vol. 160, p. 107225, Aug. 2023.

[436] F. Wotawa, B. Peischl, F. Klück, and M. Nica, "Quality assurance methodologies for automated driving," *e & i Elektrotechnik und Informationstechnik*, vol. 135, no. 4, pp. 322–327, 2018.

[437] J. C. Hayward, "Near miss determination through use of a scale of danger," *Highway Research Record*, 1972.

[438] K. Vogel, "A comparison of headway and time to collision as safety indicators," *Accident Analysis and Prevention*, vol. 35, no. 3, pp. 427–433, 2003.

[439] J. C. Hayward, "Near miss determination through use of a scale of danger," *Highway Research Record*, 1972.

[440] H. Jin, C. Shi, and T. Tsuchiya, "Constrained detecting arrays: Mathematical structures for fault identification in combinatorial interaction testing," *Information and Software Technology*, vol. 153, p. 107045, 2023.

[441] European New Car Assessment Programme (EuroNCAP), "Test Protocol - AEB Car-to-Car systems, Version 3.0.3." Available online at `https://cdn.euroncap.com/media/62794/euro-ncap-aeb-c2c-test-protocol-v303.pdf`. Accessed on 2023-01-11.

[442] AVL List GmbH, "ASAM OpenSCENARIO." Available online at `http://www.openscenario.org/`. Accessed on 2023-01-11.

[443] AVL List GmbH, "Model.CONNECT." Available online at `https://www.avl.com/-/model-connect-`. Accessed on 2023-01-11.

[444] Hexagon, "Virtual Test Drive." Available online at `https://hexagon.com/de/products/virtual-test-drive`. Accessed on 2023-01-24.

[445] D. Dodig and D. Kaufmann, "Internal communication between MATRIS and AVL," 2023-01-20.

[446] R. A. Fisher, *The design of experiments*. London: Collier Macmillan Publishers, 9th ed., 1971.

[447] R. Compton, M. T. Mehari, C. J. Colbourn, E. De Poorter, and V. R. Syrotiuk, "Screening interacting factors in a wireless network testbed using locating arrays," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 650–655, 2016.

[448] K. S. Narendra and M. A. L. Thathachar, "Learning automata - a survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-4, no. 4, pp. 323–334, 1974.

[449] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata earning from a practical perspective," in *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, pp. 256–296, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[450] N. Yevtushenko, I. Burdonov, and A. Kossachev, "Deriving distinguishing sequences for input/output automata," in *2020 IEEE East-West Design & Test Symposium (EWDTS)*, pp. 1–5, 2020.

[451] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo, "Automated unique input output sequence generation for conformance testing of FSMs," *The Computer Journal*, vol. 49, pp. 331–344, 03 2006.

[452] J. Bonn, K. Foegen, and H. Lichter, "A framework for automated combinatorial test generation, execution, and fault characterization," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 224–233, 2019.

[453] C. D. Nguyen, A. Marchetto, and P. Tonella, "Combining model-based and combinatorial testing for effective test case generation," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, (New York, NY, USA), pp. 100–110, Association for Computing Machinery, July 2012.

Appendix

Binary Balanced CAs

Binary Balanced CAs of Strength $t = 2$

Table 1: $N = 4$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 4$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(2,1)	$\mathbf{3}^1$	*0*

Table 2: $N = 5$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 5$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(3,2)	$\mathbf{4}^1$	*0*

Table 3: $N = 6$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 6$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(3,1)		(2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(3,2)	$\mathbf{10}^1$	*0*	$\mathbf{10}^1$	ζ
(4,2)	$\mathbf{10}^1$	ψ	$\mathbf{10}^1$	*0*
(4,3)	$\mathbf{10}^1$	ψ,ω	$\mathbf{10}^1$	*0*

Table 4: $N = 7$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 7$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{\lambda}$ / $\boldsymbol{y}$	(3,1)		(2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(4,2)	$\mathbf{7}^1$	*0*	7^1	ζ
(4,3)	$\mathbf{15}^1$	*0*	15^1	ζ
(5,3)	15^1	ψ	$\mathbf{15}^1$	*0*
(5,4)	15^1	ψ,ω	$\mathbf{15}^1$	*0*

Table 5: $N = 8$, $t = 2$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 8$ rows, strength $t = 2$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{\lambda}$ / $\boldsymbol{y}$	(4,2)		(4,1)		(3,1)		(2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(4,2)	$\mathbf{7}^1$	*0*	7^1	κ	7^1	ζ,κ	7^1	ζ,κ
(4,3)	7^1	ω	$\mathbf{35}^1$	*2285*	35^1	ζ	35^1	ζ
(5,3)	7^1	ψ,ω	35^1	ψ	$\mathbf{35}^1$	*2651*	35^1	ζ
(5,4)	7^1	ψ,ω	35^1	ψ,ω	$\mathbf{35}^1$	*2929*	35^1	ζ
(6,3)	7^1	ψ,ω	35^1	ψ	35^1	ψ	$\mathbf{35}^1$	*2587*
(6,4)	7^1	ψ,ω	35^1	ψ,ω	35^1	ψ	$\mathbf{35}^1$	*2801*
(6,5)	7^1	ψ,ω	35^1	ψ,ω	35^1	ψ,ω	$\mathbf{35}^1$	*2764*

Binary Balanced CAs of Strength $t = 3$

Table 6: $N = 8$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 8$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{\lambda}$ / $\boldsymbol{y}$	(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(4,2,1)	$\mathbf{4}^1$	*0*

Table 7: $N = 9$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 9$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{\lambda}$ / $\boldsymbol{y}$	(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(5,3,2)	$\mathbf{4}^2$	*0*

Table 8: $N = 10$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 10$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{\lambda}$ / $\boldsymbol{y}$	(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(5,3,2)	$\mathbf{5}^1$	*0*	$\mathbf{5}^1$	ζ
(6,3,2)	$\mathbf{5}^1$	ψ	$\mathbf{5}^1$	*0*
(6,4,2)	$\mathbf{5}^1$	ψ,ω	$\mathbf{5}^1$	*0*
(6,4,3)	$\mathbf{5}^1$	ψ,ω	$\mathbf{5}^1$	*0*

Table 9: $N = 11$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 11$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{\lambda}$ / $\boldsymbol{y}$	(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(6,3,2)	$\mathbf{5}^1$	*0*	$\mathbf{5}^1$	ζ
(6,4,2)	$\mathbf{5}^1$	*0*	$\mathbf{5}^1$	ζ
(6,4,3)	$\mathbf{5}^4$	*0*	$\mathbf{5}^4$	ζ
(7,4,2)	$\mathbf{5}^1$	ψ	$\mathbf{5}^1$	*0*
(7,4,3)	$\mathbf{5}^4$	ψ	$\mathbf{5}^4$	*0*
(7,5,3)	$\mathbf{5}^4$	ψ,ω	$\mathbf{5}^4$	*0*
(7,5,4)	$\mathbf{5}^4$	ψ,ω	$\mathbf{5}^4$	*0*

Table 10: $N = 12^*$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 12$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ $\mathtt{T}$, where $\mathtt{T}$ denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(6,3,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	$\mathtt{T}$	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	$\mathtt{T}$	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	$\mathtt{T}$	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	$\mathtt{T}$
(6,3,2)	$\mathbf{11}^1$	*0*	11^1	κ	11^1	ζ,κ	11^1	ζ,κ
(6,4,2)	11^1	ω	$\mathbf{11}^1$	*0*	11^1	ζ	11^1	ζ
(6,4,3)	11^1	ω	$\mathbf{11}^1$	*0*	11^1	ζ	11^1	ζ
(7,4,2)	11^1	ψ,ω	11^1	ψ	$\mathbf{11}^1$	*0*	11^1	ζ
(7,4,3)	11^1	ψ,ω	11^1	ψ	$\mathbf{11}^1$	*0*	11^1	ζ
(7,5,3)	11^1	ψ,ω	11^1	ψ,ω	$\mathbf{11}^1$	*1*	11^1	ζ
(7,5,4)	11^1	ψ,ω	11^1	ψ,ω	$\mathbf{11}^1$	*0*	11^1	ζ
(8,4,2)	11^1	ψ,ω	11^1	ψ	11^1	ψ	$\mathbf{11}^1$	*0*
(8,4,3)	11^1	ψ,ω	11^1	ψ	11^1	ψ	$\mathbf{11}^1$	*0*
(8,5,3)	11^1	ψ,ω	11^1	ψ,ω	11^1	ψ	$\mathbf{11}^1$	*0*
(8,6,3)	11^1	ψ,ω	11^1	ψ,ω	11^1	ψ,ω	$\mathbf{11}^1$	*0*
(8,5,4)	11^1	ψ,ω	11^1	ψ,ω	11^1	ψ	$\mathbf{11}^1$	*0*
(8,6,4)	11^1	ψ,ω	11^1	ψ,ω	11^1	ψ,ω	$\mathbf{11}^1$	*0*
(8,6,5)	11^1	ψ,ω	11^1	ψ,ω	11^1	ψ,ω	$\mathbf{11}^1$	*0*

Table 11: $N = 13$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 13$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(6,3,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(7,4,2)	$\mathbf{5}^1$	*0*	$\mathbf{8}^1$	*0*	$\mathbf{8}^1$	ζ	$\mathbf{8}^1$	ζ
(7,4,3)	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ	$\mathbf{11}^8$	ζ
(7,5,3)	$\mathbf{11}^8$	ω	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ	$\mathbf{11}^8$	ζ
(7,5,4)	$\mathbf{11}^8$	ω	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ	$\mathbf{11}^8$	ζ
(8,4,2)	$\mathbf{5}^1$	ψ	$\mathbf{8}^1$	ψ	$\mathbf{8}^1$	*0*	$\mathbf{8}^1$	ζ
(8,4,3)	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ
(8,5,3)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ
(8,6,3)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	*1*	$\mathbf{11}^8$	ζ
(8,5,4)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ
(8,6,4)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ
(8,6,5)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	*0*	$\mathbf{11}^8$	ζ
(9,5,3)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*
(9,6,3)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*
(9,5,4)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*
(9,6,4)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*1*
(9,7,4)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	*0*
(9,6,5)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ	$\mathbf{11}^8$	*0*
(9,7,5)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	*0*
(9,7,6)	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	ψ,ω	$\mathbf{11}^8$	*0*

Table 12: $N = 14$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 14$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ $\mathtt{T}$, where $\mathtt{T}$ denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(7,3,1)		(6,3,1)		(7,2,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(7,4,2)	$\mathbf{8}^{1}$	*0*	8^{1}	ζ	8^{1}	κ	8^{1}	ζ,κ	8^{1}	ζ,κ	8^{1}	ζ,κ
(7,4,3)	$\mathbf{11}^{5}$	*0*	11^{5}	ζ	11^{5}	κ	11^{5}	ζ,κ	11^{5}	ζ,κ	11^{5}	ζ,κ
(8,4,2)	8^{1}	ψ	$\mathbf{8}^{1}$	*0*	8^{1}	ψ,κ	$\mathbf{8}^{1}$	*0*	8^{1}	ζ	8^{1}	ζ
(8,4,3)	11^{5}	ψ	$\mathbf{11}^{16}$	*0*	11^{5}	ψ,κ	$\mathbf{11}^{16}$	*1*	11^{16}	ζ	11^{16}	ζ
(8,5,3)	11^{5}	ψ,ω	$\mathbf{11}^{69}$	*1*	11^{5}	ψ	$\mathbf{11}^{69}$	*1*	11^{69}	ζ	11^{69}	ζ
(8,5,4)	11^{5}	ψ,ω	$\mathbf{11}^{533}$	*2*	11^{5}	ψ	$\mathbf{11}^{533}$	*3*	11^{533}	ζ	11^{533}	ζ
(7,5,3)	11^{5}	ω	11^{5}	ζ,ω	$\mathbf{11}^{5}$	*0*	11^{5}	ζ	11^{5}	ζ	11^{5}	ζ
(7,5,4)	11^{5}	ω	11^{5}	ζ,ω	$\mathbf{11}^{5}$	*0*	11^{5}	ζ	11^{5}	ζ	11^{5}	ζ
(8,6,3)	11^{5}	ψ,ω	11^{69}	ω	11^{5}	ψ,ω	$\mathbf{11}^{69}$	*1*	11^{69}	ζ	11^{69}	ζ
(8,6,4)	11^{5}	ψ,ω	11^{533}	ω	11^{5}	ψ,ω	$\mathbf{11}^{533}$	*3*	11^{533}	ζ	11^{533}	ζ
(8,6,5)	11^{5}	ψ,ω	11^{533}	ω	11^{5}	ψ,ω	$\mathbf{11}^{533}$	*3*	11^{533}	ζ	11^{533}	ζ
(9,5,3)	11^{5}	ψ,ω	11^{69}	ψ	11^{5}	ψ	11^{69}	ψ	$\mathbf{11}^{69}$	*2*	11^{69}	ζ
(9,6,3)	11^{5}	ψ,ω	11^{69}	ψ,ω	11^{5}	ψ,ω	11^{69}	ψ	$\mathbf{11}^{69}$	*1*	11^{69}	ζ
(9,5,4)	11^{5}	ψ,ω	11^{533}	ψ	11^{5}	ψ	11^{533}	ψ	$\mathbf{11}^{533}$	*3*	11^{533}	ζ
(9,6,4)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ	$\mathbf{11}^{533}$	*3*	11^{533}	ζ
(9,7,4)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*3*	11^{533}	ζ
(9,6,5)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ	$\mathbf{11}^{533}$	*4*	11^{533}	ζ
(9,7,5)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*2*	11^{533}	ζ
(9,7,6)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*3*	11^{533}	ζ
(10,5,3)	11^{5}	ψ,ω	11^{69}	ψ	11^{5}	ψ	11^{69}	ψ	11^{69}	ψ	$\mathbf{11}^{69}$	*1*
(10,6,3)	11^{5}	ψ,ω	11^{69}	ψ,ω	11^{5}	ψ,ω	11^{69}	ψ	11^{69}	ψ	$\mathbf{11}^{69}$	*1*
(10,5,4)	11^{5}	ψ,ω	11^{533}	ψ	11^{5}	ψ	11^{533}	ψ	11^{533}	ψ	$\mathbf{11}^{533}$	*2*
(10,6,4)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ	11^{533}	ψ	$\mathbf{11}^{533}$	*4*
(10,7,4)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ	$\mathbf{11}^{533}$	*3*
(10,8,4)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*3*
(10,6,5)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ	11^{533}	ψ	$\mathbf{11}^{533}$	*3*
(10,7,5)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ	$\mathbf{11}^{533}$	*3*
(10,8,5)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*3*
(10,7,6)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ	$\mathbf{11}^{533}$	*4*
(10,8,6)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*2*
(10,8,7)	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{5}	ψ,ω	11^{533}	ψ,ω	11^{533}	ψ,ω	$\mathbf{11}^{533}$	*3*

Table 13: $N = 16^*$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds). For this instance we let the computational search terminate which took a maximum of 63 hours.

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(8,4,2)		(8,4,1)		(8,3,1)		(7,3,1)		(6,3,1)		(8,2,1)		(7,2,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(8,4,2)	8^1	*0*	8^1	κ	8^1	κ	8^1	ζ,κ	8^1	ζ,κ	8^1	κ	8^1	ζ,κ	8^1	ζ,κ	8^1	ζ,κ	8^1	ζ,κ
(8,4,3)	8^1	ω	14^1	*0*	14^1	κ	14^1	ζ,κ	14^1	ζ,κ	14^1	κ	14^1	ζ,κ	14^1	ζ,κ	14^1	ζ,κ	14^1	ζ,κ
(8,5,3)	8^1	ω	14^1	ω	14^2	*260*	14^2	ζ	14^2	ζ	14^2	κ	14^2	ζ,κ	14^2	ζ,κ	14^2	ζ,κ	14^2	ζ,κ
(8,5,4)	8^1	ω	14^1	ω	14^2	*1047*	14^2	ζ	14^2	ζ	14^2	κ	14^2	ζ,κ	14^2	ζ,κ	14^2	ζ,κ	14^2	ζ,κ
(9,5,3)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ	14^2	*627*	14^2	ζ	14^2	ψ,κ	14^2	*642*	14^2	ζ	14^2	ζ	14^2	ζ
(9,6,3)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^2	*635*	14^2	ζ	14^3	ψ	14^3	*533*	14^3	ζ	14^3	ζ	14^3	ζ
(9,5,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ	14^2	*21185*	14^2	ζ	14^2	ψ,κ	14^2	*22328*	14^2	ζ	14^2	ζ	14^2	ζ
(9,6,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	*78514*	14^7	ζ	14^3	ψ	14^8	*86111*	14^8	ζ	14^8	ζ	14^8	ζ
(9,6,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	*96338*	14^7	ζ	14^3	ψ	14^8	*104028*	14^8	ζ	14^8	ζ	14^8	ζ
(10,5,3)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ	14^2	ψ	14^2	*602*	14^2	ψ,κ	14^2	ψ	14^2	*637*	14^2	ζ	14^2	ζ
(10,6,3)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^2	ψ	14^2	*628*	14^3	ψ	14^3	ψ	14^3	*570*	14^3	ζ	14^3	ζ
(10,5,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ	14^2	ψ	14^2	*22037*	14^2	ψ,κ	14^2	ψ	14^2	*21919*	14^2	ζ	14^2	ζ
(10,6,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ	14^7	*100214*	14^3	ψ	14^8	ψ	14^8	*113030*	14^8	ζ	14^8	ζ
(10,7,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	*101466*	14^3	ψ,ω	14^8	ψ	14^8	*113936*	14^8	ζ	14^8	ζ
(10,6,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ	14^7	*161150*	14^3	ψ	14^8	ψ	14^8	*184147*	14^8	ζ	14^8	ζ
(10,7,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	*196240*	14^3	ψ,ω	14^8	ψ	14^8	*219939*	14^8	ζ	14^8	ζ
(10,7,6)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	*205391*	14^3	ψ,ω	14^8	ψ	14^8	*229816*	14^8	ζ	14^8	ζ
(8,6,3)	8^1	ω	14^1	ω	14^2	ω	14^2	ζ,ω	14^2	ζ,ω	14^3	*230*	14^3	ζ	14^3	ζ	14^3	ζ	14^3	ζ
(8,6,4)	8^1	ω	14^1	ω	14^2	ω	14^2	ζ,ω	14^2	ζ,ω	14^3	*910*	14^3	ζ	14^3	ζ	14^3	ζ	14^3	ζ
(8,6,5)	8^1	ω	14^1	ω	14^2	ω	14^2	ζ,ω	14^2	ζ,ω	14^3	*904*	14^3	ζ	14^3	ζ	14^3	ζ	14^3	ζ
(9,7,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ω	14^7	ζ,ω	14^3	ψ,ω	14^8	*73882*	14^8	ζ	14^8	ζ	14^8	ζ
(9,7,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ω	14^7	ζ,ω	14^3	ψ,ω	14^8	*89552*	14^8	ζ	14^8	ζ	14^8	ζ
(9,7,6)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ω	14^7	ζ,ω	14^3	ψ,ω	14^8	*87604*	14^8	ζ	14^8	ζ	14^8	ζ
(10,8,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ω	14^3	ψ,ω	14^8	ψ,ω	14^8	*98766*	14^8	ζ	14^8	ζ
(10,8,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ω	14^3	ψ,ω	14^8	ψ,ω	14^8	*194602*	14^8	ζ	14^8	ζ
(10,8,6)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ω	14^3	ψ,ω	14^8	ψ,ω	14^8	*203381*	14^8	ζ	14^8	ζ
(10,8,7)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ω	14^3	ψ,ω	14^8	ψ,ω	14^8	*201298*	14^8	ζ	14^8	ζ
(11,6,3)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^2	ψ	14^2	ψ	14^3	ψ	14^3	ψ	14^3	ψ	14^3	*543*	14^3	ζ
(11,6,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ	14^7	ψ	14^3	ψ	14^8	ψ	14^8	ψ	14^8	*119136*	14^8	ζ
(11,7,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ	14^3	ψ,ω	14^8	ψ	14^8	ψ	14^8	*121848*	14^8	ζ
(11,8,4)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ	14^8	*105545*	14^8	ζ
(11,6,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ	14^7	ψ	14^3	ψ	14^8	ψ	14^8	ψ	14^8	*183830*	14^8	ζ
(11,7,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ	14^3	ψ,ω	14^8	ψ	14^8	ψ	14^8	*221213*	14^8	ζ
(11,8,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ	14^8	*221916*	14^8	ζ
(11,9,5)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ,ω	14^8	*193094*	14^8	ζ
(11,7,6)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ	14^3	ψ,ω	14^8	ψ	14^8	ψ	14^8	*218826*	14^8	ζ
(11,8,6)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ	14^8	*219599*	14^8	ζ
(11,9,6)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ,ω	14^8	*192241*	14^8	ζ
(11,8,7)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ	14^8	*220120*	14^8	ζ
(11,9,7)	8^1	ψ,ω	14^1	ψ,ω	14^2	ψ,ω	14^7	ψ,ω	14^7	ψ,ω	14^3	ψ,ω	14^8	ψ,ω	14^8	ψ,ω	14^8	*191743*	14^8	ζ

λ / y	(8,4,2)		(8,4,1)		(8,3,1)		(7,3,1)		(6,3,1)		(8,2,1)		(7,2,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(11,9,8)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	*191391*	$\mathbf{14}^8$	ζ
(12,6,3)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^2$	ψ	$\mathbf{14}^2$	ψ	$\mathbf{14}^3$	ψ	$\mathbf{14}^3$	ψ	$\mathbf{14}^3$	ψ	$\mathbf{14}^3$	ψ	$\mathbf{14}^3$	*528*
(12,6,4)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ	$\mathbf{14}^7$	ψ	$\mathbf{14}^3$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*113851*
(12,7,4)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*115076*
(12,8,4)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*98267*
(12,6,5)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ	$\mathbf{14}^7$	ψ	$\mathbf{14}^3$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*180602*
(12,7,5)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*216297*
(12,8,5)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*217321*
(12,9,5)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*217618*
(12,10,5)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	*189727*
(12,7,6)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*226374*
(12,8,6)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*224851*
(12,9,6)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*224932*
(12,10,6)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	*196473*
(12,8,7)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*210056*
(12,9,7)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*211974*
(12,10,7)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	*184631*
(12,9,8)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ	$\mathbf{14}^8$	*213097*
(12,10,8)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	*184174*
(12,10,9)	$\mathbf{8}^1$	ψ,ω	$\mathbf{14}^1$	ψ,ω	$\mathbf{14}^2$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^7$	ψ,ω	$\mathbf{14}^3$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	ψ,ω	$\mathbf{14}^8$	*184106*

Table 14: $N = 17$, $t = 3$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 17$ rows, strength $t = 3$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(8,4,2)		(8,4,1)		(8,3,1)		(7,3,1)		(6,3,1)		(8,2,1)		(7,2,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(9,5,3)	$\mathbf{8}^3$	*0*	$\mathbf{10}^3$	*0*	$\mathbf{10}^{23}$	*81*	$\mathbf{10}^{23}$	ζ	$\mathbf{10}^{23}$	ζ	$\mathbf{10}^{23}$	κ	$\mathbf{10}^{23}$	ζ,κ	$\mathbf{10}^{23}$	ζ,κ	$\mathbf{10}^{23}$	ζ,κ	$\mathbf{10}^{23}$	ζ,κ
(9,5,4)	$\mathbf{8}^3$	ω	$\mathbf{14}^{21}$	*5*	t.o.		-	ζ	-	ζ	-	κ	-	ζ,κ	-	ζ,κ	-	ζ,κ	-	ζ,κ
(9,6,3)	$\mathbf{8}^3$	ω	$\mathbf{10}^3$	ω	$\mathbf{10}^{29}$	*124*	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	*106*	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	ζ
(9,6,4)	$\mathbf{8}^3$	ω	$\mathbf{14}^{21}$	ω	-		-	ζ	-	ζ	t.o.		-	ζ	-	ζ	-	ζ	-	ζ
(9,6,5)	$\mathbf{8}^3$	ω	$\mathbf{14}^{21}$	ω	-		-	ζ	-	ζ	-		-	ζ	-	ζ	-	ζ	-	ζ
(10,5,3)	$\mathbf{8}^3$	ψ	$\mathbf{10}^3$	ψ	$\mathbf{10}^{23}$	ψ	$\mathbf{10}^{29}$	*88*	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{23}$	ψ,κ	$\mathbf{10}^{29}$	*87*	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	ζ	$\mathbf{10}^{29}$	ζ
(10,6,3)	$\mathbf{8}^3$	ψ,ω	$\mathbf{10}^3$	ψ,ω	$\mathbf{10}^{29}$	ψ	$\mathbf{11}^3$	*163*	$\mathbf{11}^3$	ζ	$\mathbf{10}^{29}$	ψ	$\mathbf{11}^3$	*132*	$\mathbf{11}^3$	ζ	$\mathbf{11}^3$	ζ	$\mathbf{11}^3$	ζ
(10,5,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ	-	ψ	t.o.		-	ζ	-	ψ,κ	t.o.		-	ζ	-	ζ	-	ζ
(10,6,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ	-		-	ζ	-	ψ	-		-	ζ	-	ζ	-	ζ
(10,7,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-		-	ζ	-	ψ	-		-	ζ	-	ζ	-	ζ
(10,6,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ	-		-	ζ	-	ψ	-		-	ζ	-	ζ	-	ζ
(10,7,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-		-	ζ	-	ψ	-		-	ζ	-	ζ	-	ζ
(10,7,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-		-	ζ	$\mathbf{16}^3$	ψ	-		-	ζ	-	ζ	-	ζ
(11,6,3)	$\mathbf{8}^3$	ψ,ω	$\mathbf{10}^3$	ψ,ω	$\mathbf{10}^{29}$	ψ	$\mathbf{11}^3$	ψ	$\mathbf{11}^4$	*160*	$\mathbf{10}^{29}$	ψ	$\mathbf{11}^3$	ψ	$\mathbf{11}^4$	*120*	$\mathbf{11}^4$	ζ	$\mathbf{11}^4$	ζ
(11,6,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ	-	ψ	t.o.		-	ψ	-	ψ	t.o.		-	ζ	-	ζ
(11,7,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-		-	ψ	-	ψ	-		-	ζ	-	ζ
(11,8,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ψ,ω	-	ψ	-		-	ζ	-	ζ
(11,6,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ	-	ψ	-		-	ψ	-	ψ	-		-	ζ	-	ζ
(11,7,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-		-	ψ	-	ψ	-		-	ζ	-	ζ
(11,8,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ψ,ω	-	ψ	-		-	ζ	-	ζ
(11,7,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-		$\mathbf{16}^3$	ψ	-	ψ	-		-	ζ	-	ζ
(11,8,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-		$\mathbf{16}^3$	ψ,ω	-	ψ	-		-	ζ	-	ζ
(11,8,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-		$\mathbf{16}^3$	ψ,ω	-	ψ	-		-	ζ	-	ζ
(9,7,4)	$\mathbf{8}^3$	ω	$\mathbf{14}^{21}$	ω	-	ω	-	ζ,ω	-	ζ,ω	-		-	ζ	-	ζ	-	ζ	-	ζ
(9,7,5)	$\mathbf{8}^3$	ω	$\mathbf{14}^{21}$	ω	-	ω	-	ζ,ω	-	ζ,ω	-		-	ζ	-	ζ	-	ζ	-	ζ
(9,7,6)	$\mathbf{8}^3$	ω	$\mathbf{14}^{21}$	ω	-	ω	-	ζ,ω	-	ζ,ω	$\mathbf{16}^3$	*4.42 ·10⁷ s*	$\mathbf{16}^3$	ζ	$\mathbf{16}^3$	ζ	$\mathbf{16}^3$	ζ	$\mathbf{16}^3$	ζ
(10,8,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ω	-	ζ,ω	-	ψ,ω	-		-	ζ	-	ζ	-	ζ
(10,8,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ω	-	ζ,ω	-	ψ,ω	-		-	ζ	-	ζ	-	ζ
(10,8,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ω	-	ζ,ω	$\mathbf{16}^3$	ψ,ω	-		-	ζ	-	ζ	-	ζ
(10,8,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ω	-	ζ,ω	$\mathbf{16}^3$	ψ,ω	-		-	ζ	-	ζ	-	ζ
(11,9,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ω	-	ψ,ω	-	ψ,ω	-		-	ζ	-	ζ
(11,9,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-		-	ζ	-	ζ
(11,9,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-		-	ζ	-	ζ
(11,9,8)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-		-	ζ	-	ζ
(12,6,3)	$\mathbf{8}^3$	ψ,ω	$\mathbf{10}^3$	ψ,ω	$\mathbf{10}^{29}$	ψ	$\mathbf{11}^3$	ψ	$\mathbf{11}^4$	ψ	$\mathbf{10}^{29}$	ψ	$\mathbf{11}^3$	ψ	$\mathbf{11}^4$	ψ	$\mathbf{11}^4$	*128*	$\mathbf{11}^4$	ζ
(12,6,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	t.o.		-	ζ
(12,7,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-		-	ζ
(12,8,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	ψ,ω	-	ψ	-	ψ	-		-	ζ
(12,6,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-		-	ζ
(12,7,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-		-	ζ
(12,8,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	ψ,ω	-	ψ	-	ψ	-		-	ζ

λ / y	(8,4,2)		(8,4,1)		(8,3,1)		(7,3,1)		(6,3,1)		(8,2,1)		(7,2,1)		(6,2,1)		(5,2,1)		(4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(12,9,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-		-	ζ
(12,10,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ζ
(12,7,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	$\mathbf{16}^3$	ψ	-	ψ	-	ψ	-		-	ζ
(12,8,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	$\mathbf{16}^3$	ψ,ω	-	ψ	-	ψ	-		-	ζ
(12,9,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ	-		-	ζ
(12,10,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ζ
(12,8,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	$\mathbf{16}^3$	ψ,ω	-	ψ	-	ψ	-		-	ζ
(12,9,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ	-		-	ζ
(12,10,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ζ
(12,9,8)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ	-		-	ζ
(12,10,8)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ζ
(12,10,9)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-		-	ζ
(13,7,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	t.o.	
(13,8,4)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	ψ,ω	-	ψ	-	ψ	-	ψ	-	
(13,7,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-	ψ	-	
(13,8,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	ψ,ω	-	ψ	-	ψ	-	ψ	-	
(13,9,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	
(13,10,5)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	
(13,7,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	$\mathbf{16}^3$	ψ	-	ψ	-	ψ	-	ψ	-	
(13,8,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	$\mathbf{16}^3$	ψ,ω	-	ψ	-	ψ	-	ψ	-	
(13,9,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	
(13,10,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	
(13,11,6)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	
(13,8,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	$\mathbf{16}^3$	ψ,ω	-	ψ	-	ψ	-	ψ	-	
(13,9,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	
(13,10,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	
(13,11,7)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	
(13,9,8)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ	-	ψ	-	
(13,10,8)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	
(13,11,8)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	
(13,10,9)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ	-	
(13,11,9)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	
(13,11,10)	$\mathbf{8}^3$	ψ,ω	$\mathbf{14}^{21}$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	$\mathbf{16}^3$	ψ,ω	-	ψ,ω	-	ψ,ω	-	ψ,ω	-	

6.0pt

Binary Balanced CAs of Strength $t = 4$

Table 15: $N = 16$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(8,4,2,1)	$\mathbf{5}^1$	*0*

Table 16: $N = 17$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 17$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(9,5,3,2)	$\mathbf{5}^2$	*0*

Table 17: $N = 18$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 18$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(9,5,3,2)	$\mathbf{5}^1$	*0*	$\mathbf{5}^1$	ζ
(10,5,3,2)	$\mathbf{5}^1$	ψ	$\mathbf{5}^3$	*0*
(10,6,3,2)	$\mathbf{5}^1$	ψ,ω	$\mathbf{5}^4$	*0*
(10,6,4,2)	$\mathbf{5}^1$	ψ,ω	$\mathbf{5}^6$	*0*
(10,6,4,3)	$\mathbf{5}^1$	ψ,ω	$\mathbf{5}^9$	*0*

Table 18: $N = 19$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 19$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(10,5,3,2)	$\emptyset$	*0*	$\emptyset$	ζ
(10,6,3,2)	$\mathbf{5}^2$	*0*	$\mathbf{5}^2$	ζ
(10,6,4,2)	$\mathbf{5}^6$	*0*	$\mathbf{5}^6$	ζ
(10,6,4,3)	$\mathbf{5}^{10}$	*0*	$\mathbf{5}^{10}$	ζ

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(11,6,3,2)	$\mathbf{5}^{2}$	ψ	$\mathbf{5}^{2}$	*0*
(11,6,4,2)	$\mathbf{5}^{6}$	ψ	$\mathbf{5}^{10}$	*0*
(11,7,4,2)	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{12}$	*0*
(11,6,4,3)	$\mathbf{5}^{10}$	ψ	$\mathbf{5}^{18}$	*0*
(11,7,4,3)	$\mathbf{5}^{10}$	ψ,ω	$\mathbf{5}^{24}$	*0*
(11,7,5,3)	$\mathbf{5}^{10}$	ψ,ω	$\mathbf{5}^{28}$	*0*
(11,7,5,4)	$\mathbf{5}^{10}$	ψ,ω	$\mathbf{5}^{32}$	*0*

Table 19: $N = 20$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 20$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,5,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(10,5,3,2)	$\emptyset$	*0*	$\emptyset$	κ	$\emptyset$	ζ,κ	$\emptyset$	ζ,κ
(10,6,3,2)	$\emptyset$	ω	$\mathbf{5}^{1}$	*0*	$\mathbf{5}^{1}$	ζ	$\mathbf{5}^{1}$	ζ
(10,6,4,2)	$\emptyset$	ω	$\mathbf{5}^{4}$	*0*	$\mathbf{5}^{4}$	ζ	$\mathbf{5}^{4}$	ζ
(10,6,4,3)	$\emptyset$	ω	$\mathbf{5}^{6}$	*0*	$\mathbf{5}^{6}$	ζ	$\mathbf{5}^{6}$	ζ
(11,6,3,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{1}$	ψ	$\mathbf{5}^{2}$	*0*	$\mathbf{5}^{2}$	ζ
(11,6,4,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{4}$	ψ	$\mathbf{5}^{17}$	*0*	$\mathbf{5}^{17}$	ζ
(11,7,4,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{4}$	ψ,ω	$\mathbf{5}^{25}$	*0*	$\mathbf{5}^{25}$	ζ
(11,6,4,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ	$\mathbf{5}^{29}$	*0*	$\mathbf{5}^{29}$	ζ
(11,7,4,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{54}$	*0*	$\mathbf{5}^{54}$	ζ
(11,7,5,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{66}$	*0*	$\mathbf{5}^{66}$	ζ
(11,7,5,4)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{72}$	*0*	$\mathbf{5}^{72}$	ζ
(12,6,3,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{1}$	ψ	$\mathbf{5}^{2}$	ψ	$\mathbf{5}^{2}$	*0*
(12,6,4,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{4}$	ψ	$\mathbf{5}^{17}$	ψ	$\mathbf{5}^{20}$	*0*
(12,7,4,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{4}$	ψ,ω	$\mathbf{5}^{25}$	ψ	$\mathbf{5}^{34}$	*0*
(12,8,4,2)	$\emptyset$	ψ,ω	$\mathbf{5}^{4}$	ψ,ω	$\mathbf{5}^{25}$	ψ,ω	$\mathbf{5}^{36}$	*0*
(12,6,4,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ	$\mathbf{5}^{29}$	ψ	$\mathbf{5}^{35}$	*0*
(12,7,4,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{54}$	ψ	$\mathbf{5}^{77}$	*0*
(12,8,4,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{54}$	ψ,ω	$\mathbf{5}^{81}$	*0*
(12,7,5,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{66}$	ψ	$\mathbf{5}^{101}$	*0*
(12,8,5,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{66}$	ψ,ω	$\mathbf{5}^{114}$	*0*
(12,8,6,3)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{66}$	ψ,ω	$\mathbf{5}^{117}$	*0*
(12,7,5,4)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{72}$	ψ	$\mathbf{5}^{113}$	*0*
(12,8,5,4)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{72}$	ψ,ω	$\mathbf{5}^{132}$	*0*
(12,8,6,4)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{72}$	ψ,ω	$\mathbf{5}^{141}$	*0*
(12,8,6,5)	$\emptyset$	ψ,ω	$\mathbf{5}^{6}$	ψ,ω	$\mathbf{5}^{72}$	ψ,ω	$\mathbf{5}^{146}$	*0*

Table 20: $N = 21^*$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 21$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,5,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(11,6,3,2)	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(11,6,4,2)	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(11,6,4,3)	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(11,7,4,2)	$\mathbf{6}^1$	ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(11,7,4,3)	$\mathbf{6}^1$	ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(11,7,5,3)	$\mathbf{6}^1$	ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(11,7,5,4)	$\mathbf{6}^1$	ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ	$\mathbf{6}^1$	ζ
(12,6,3,2)	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,6,4,2)	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,7,4,2)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,4,2)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,6,4,3)	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,7,4,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,4,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,7,5,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,5,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,6,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,7,5,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,5,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,6,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(12,8,6,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*	$\mathbf{6}^1$	ζ
(13,7,4,2)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,8,4,2)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,7,4,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,8,4,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,7,5,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,8,5,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,9,5,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*
(13,8,6,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,9,6,3)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*
(13,7,5,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,8,5,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*1*
(13,9,5,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*
(13,8,6,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,9,6,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*
(13,9,7,4)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*
(13,8,6,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*0*
(13,9,6,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*1*
(13,9,7,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*
(13,9,7,6)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*0*

Table 21: $N = 22$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 22$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(11,5,2,1)		(10,5,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(11,6,3,2)	6^1	0	6^1	ζ	6^1	κ	6^1	ζ,κ	6^1	ζ,κ	6^1	ζ,κ
(11,6,4,2)	6^1	0	6^1	ζ	6^1	κ	6^1	ζ,κ	6^1	ζ,κ	6^1	ζ,κ
(11,6,4,3)	6^2	0	6^2	ζ	6^2	κ	6^2	ζ,κ	6^2	ζ,κ	6^2	ζ,κ
(12,6,3,2)	6^1	ψ	6^1	0	6^1	ψ,κ	6^1	0	6^1	ζ	6^1	ζ
(12,6,4,2)	6^1	ψ	6^1	0	6^1	ψ,κ	6^1	0	6^1	ζ	6^1	ζ
(12,7,4,2)	6^1	ψ,ω	6^2	0	6^1	ψ	6^2	0	6^2	ζ	6^2	ζ
(12,6,4,3)	6^2	ψ	6^3	0	6^2	ψ,κ	6^3	0	6^3	ζ	6^3	ζ
(12,7,4,3)	6^2	ψ,ω	6^8	0	6^2	ψ	6^8	0	6^8	ζ	6^8	ζ
(12,7,5,3)	6^2	ψ,ω	6^8	0	6^2	ψ	6^8	0	6^8	ζ	6^8	ζ
(12,7,5,4)	6^2	ψ,ω	6^8	0	6^2	ψ	6^8	1	6^8	ζ	6^8	ζ
(11,7,4,2)	6^1	ω	6^1	ζ,ω	6^1	0	6^1	ζ	6^1	ζ	6^1	ζ
(11,7,4,3)	6^2	ω	6^2	ζ,ω	6^2	0	6^2	ζ	6^2	ζ	6^2	ζ
(11,7,5,3)	6^2	ω	6^2	ζ,ω	6^2	0	6^2	ζ	6^2	ζ	6^2	ζ
(11,7,5,4)	6^2	ω	6^2	ζ,ω	6^2	0	6^2	ζ	6^2	ζ	6^2	ζ
(12,8,4,2)	6^1	ψ,ω	6^2	ω	6^1	ψ,ω	6^2	0	6^2	ζ	6^2	ζ
(12,8,4,3)	6^2	ψ,ω	6^8	ω	6^2	ψ,ω	6^8	0	6^8	ζ	6^8	ζ
(12,8,5,3)	6^2	ψ,ω	6^8	ω	6^2	ψ,ω	6^8	0	6^8	ζ	6^8	ζ
(12,8,6,3)	6^2	ψ,ω	6^8	ω	6^2	ψ,ω	6^8	0	6^8	ζ	6^8	ζ
(12,8,5,4)	6^2	ψ,ω	6^8	ω	6^2	ψ,ω	6^8	0	6^8	ζ	6^8	ζ
(12,8,6,4)	6^2	ψ,ω	6^8	ω	6^2	ψ,ω	6^8	0	6^8	ζ	6^8	ζ
(12,8,6,5)	6^2	ψ,ω	6^8	ω	6^2	ψ,ω	6^8	0	6^8	ζ	6^8	ζ
(13,7,4,2)	6^1	ψ,ω	6^2	ψ	6^1	ψ	6^2	ψ	6^2	0	6^2	ζ
(13,8,4,2)	6^1	ψ,ω	6^2	ψ,ω	6^1	ψ,ω	6^2	ψ	6^2	0	6^2	ζ
(13,7,4,3)	6^2	ψ,ω	6^8	ψ	6^2	ψ	6^8	ψ	6^8	0	6^8	ζ
(13,8,4,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	0	6^8	ζ
(13,7,5,3)	6^2	ψ,ω	6^8	ψ	6^2	ψ	6^8	ψ	6^8	0	6^8	ζ
(13,8,5,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	0	6^8	ζ
(13,9,5,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	1	6^8	ζ
(13,8,6,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	0	6^8	ζ
(13,9,6,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	0	6^8	ζ
(13,7,5,4)	6^2	ψ,ω	6^8	ψ	6^2	ψ	6^8	ψ	6^8	0	6^8	ζ
(13,8,5,4)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	1	6^8	ζ
(13,9,5,4)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	0	6^8	ζ
(13,8,6,4)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	1	6^8	ζ
(13,9,6,4)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	0	6^8	ζ
(13,9,7,4)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	1	6^8	ζ
(13,8,6,5)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	0	6^8	ζ
(13,9,6,5)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	1	6^8	ζ
(13,9,7,5)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	0	6^8	ζ
(13,9,7,6)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ,ω	6^8	1	6^8	ζ
(14,7,4,2)	6^1	ψ,ω	6^2	ψ	6^1	ψ	6^2	ψ	6^2	ψ	6^2	0
(14,8,4,2)	6^1	ψ,ω	6^2	ψ,ω	6^1	ψ,ω	6^2	ψ	6^2	ψ	6^2	0
(14,7,4,3)	6^2	ψ,ω	6^8	ψ	6^2	ψ	6^8	ψ	6^8	ψ	6^8	0
(14,8,4,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	ψ	6^8	0
(14,7,5,3)	6^2	ψ,ω	6^8	ψ	6^2	ψ	6^8	ψ	6^8	ψ	6^8	0
(14,8,5,3)	6^2	ψ,ω	6^8	ψ,ω	6^2	ψ,ω	6^8	ψ	6^8	ψ	6^8	1

y \ λ	(11,5,2,1)		(10,5,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(14,9,5,3)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,10,5,3)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,8,6,3)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,9,6,3)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*1*
(14,10,6,3)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*0*
(14,7,5,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^2$	ψ	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,8,5,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*1*
(14,9,5,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,10,5,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,8,6,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,9,6,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*1*
(14,10,6,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,9,7,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,10,7,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,10,8,4)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,8,6,5)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*0*
(14,9,6,5)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*1*
(14,10,6,5)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*0*
(14,9,7,5)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*1*
(14,10,7,5)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,10,8,5)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*0*
(14,9,7,6)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ	$\mathbf{6}^8$	*1*
(14,10,7,6)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*
(14,10,8,6)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*0*
(14,10,8,7)	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	ψ,ω	$\mathbf{6}^8$	*1*

Table 22: $N = 23$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 23$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(11,5,2,1)		(10,5,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(12,6,3,2)	$\mathbf{6}^{1}$	*0*	$\mathbf{6}^{1}$	ζ	$\mathbf{6}^{1}$	κ	$\mathbf{6}^{1}$	ζ,κ	$\mathbf{6}^{1}$	ζ,κ	$\mathbf{6}^{1}$	ζ,κ
(12,6,4,2)	$\mathbf{6}^{1}$	*0*	$\mathbf{6}^{1}$	ζ	$\mathbf{6}^{1}$	κ	$\mathbf{6}^{1}$	ζ,κ	$\mathbf{6}^{1}$	ζ,κ	$\mathbf{6}^{1}$	ζ,κ
(12,7,4,2)	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	ζ
(12,6,4,3)	$\mathbf{6}^{1}$	*0*	$\mathbf{6}^{1}$	ζ	$\mathbf{6}^{1}$	κ	$\mathbf{6}^{1}$	ζ,κ	$\mathbf{6}^{1}$	ζ,κ	$\mathbf{6}^{1}$	ζ,κ
(12,7,4,3)	$\mathbf{6}^{14}$	*0*	$\mathbf{6}^{14}$	ζ	$\mathbf{6}^{14}$	*0*	$\mathbf{6}^{14}$	ζ	$\mathbf{6}^{14}$	ζ	$\mathbf{6}^{14}$	ζ
(12,7,5,3)	$\mathbf{6}^{16}$	*0*	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	*0*	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	ζ
(12,7,5,4)	$\mathbf{6}^{20}$	*0*	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	*0*	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ
(13,7,4,2)	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	ζ
(13,8,4,2)	$\mathbf{6}^{2}$	ψ,ω	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	ζ
(13,7,4,3)	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{22}$	*1*	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{22}$	*0*	$\mathbf{6}^{22}$	ζ	$\mathbf{6}^{22}$	ζ
(13,8,4,3)	$\mathbf{6}^{14}$	ψ,ω	$\mathbf{6}^{25}$	*0*	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{25}$	*1*	$\mathbf{6}^{25}$	ζ	$\mathbf{6}^{25}$	ζ
(13,7,5,3)	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{26}$	*0*	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{26}$	*0*	$\mathbf{6}^{26}$	ζ	$\mathbf{6}^{26}$	ζ
(13,8,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	*0*	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ	$\mathbf{6}^{40}$	ζ
(13,8,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	*0*	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ	$\mathbf{6}^{40}$	ζ
(13,7,5,4)	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{32}$	*0*	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{32}$	*1*	$\mathbf{6}^{32}$	ζ	$\mathbf{6}^{32}$	ζ
(13,8,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*0*	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,8,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*0*	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,8,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*0*	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(12,8,4,2)	$\mathbf{6}^{2}$	ω	$\mathbf{6}^{2}$	ζ,ω	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	ζ	$\mathbf{6}^{2}$	ζ
(12,8,4,3)	$\mathbf{6}^{14}$	ω	$\mathbf{6}^{14}$	ζ,ω	$\mathbf{6}^{14}$	*0*	$\mathbf{6}^{14}$	ζ	$\mathbf{6}^{14}$	ζ	$\mathbf{6}^{14}$	ζ
(12,8,5,3)	$\mathbf{6}^{16}$	ω	$\mathbf{6}^{16}$	ζ,ω	$\mathbf{6}^{16}$	*0*	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	ζ
(12,8,6,3)	$\mathbf{6}^{16}$	ω	$\mathbf{6}^{16}$	ζ,ω	$\mathbf{6}^{16}$	*0*	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	ζ	$\mathbf{6}^{16}$	ζ
(12,8,5,4)	$\mathbf{6}^{20}$	ω	$\mathbf{6}^{20}$	ζ,ω	$\mathbf{6}^{20}$	*1*	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ
(12,8,6,4)	$\mathbf{6}^{20}$	ω	$\mathbf{6}^{20}$	ζ,ω	$\mathbf{6}^{20}$	*0*	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ
(12,8,6,5)	$\mathbf{6}^{20}$	ω	$\mathbf{6}^{20}$	ζ,ω	$\mathbf{6}^{20}$	*0*	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ	$\mathbf{6}^{20}$	ζ
(13,9,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ	$\mathbf{6}^{40}$	ζ
(13,9,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	*0*	$\mathbf{6}^{40}$	ζ	$\mathbf{6}^{40}$	ζ
(13,9,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,9,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,9,7,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,9,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,9,7,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(13,9,7,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ	$\mathbf{6}^{54}$	ζ
(14,7,4,2)	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ
(14,8,4,2)	$\mathbf{6}^{2}$	ψ,ω	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	*0*	$\mathbf{6}^{2}$	ζ
(14,7,4,3)	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{22}$	ψ	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{22}$	ψ	$\mathbf{6}^{22}$	*0*	$\mathbf{6}^{22}$	ζ
(14,8,4,3)	$\mathbf{6}^{14}$	ψ,ω	$\mathbf{6}^{25}$	ψ	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{25}$	ψ	$\mathbf{6}^{25}$	*1*	$\mathbf{6}^{25}$	ζ
(14,7,5,3)	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{26}$	ψ	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{26}$	ψ	$\mathbf{6}^{26}$	*0*	$\mathbf{6}^{26}$	ζ
(14,8,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ
(14,9,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*2*	$\mathbf{6}^{40}$	ζ
(14,10,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ
(14,8,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ
(14,9,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ
(14,10,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{40}$	*1*	$\mathbf{6}^{40}$	ζ
(14,7,5,4)	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{32}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{32}$	ψ	$\mathbf{6}^{32}$	*0*	$\mathbf{6}^{32}$	ζ
(14,8,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ

y \ λ	(11,5,2,1)		(10,5,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T
(14,9,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,10,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ
(14,8,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,9,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,10,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,9,7,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ
(14,10,7,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*	$\mathbf{6}^{54}$	ζ
(14,10,8,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ
(14,8,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ
(14,9,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,10,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*	$\mathbf{6}^{54}$	ζ
(14,9,7,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,10,7,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*	$\mathbf{6}^{54}$	ζ
(14,10,8,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*1*	$\mathbf{6}^{54}$	ζ
(14,9,7,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,10,7,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*	$\mathbf{6}^{54}$	ζ
(14,10,8,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(14,10,8,7)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*	$\mathbf{6}^{54}$	ζ
(15,8,4,2)	$\mathbf{6}^{2}$	ψ,ω	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	ψ	$\mathbf{6}^{2}$	*0*
(15,8,4,3)	$\mathbf{6}^{14}$	ψ,ω	$\mathbf{6}^{25}$	ψ	$\mathbf{6}^{14}$	ψ	$\mathbf{6}^{25}$	ψ	$\mathbf{6}^{25}$	ψ	$\mathbf{6}^{25}$	*0*
(15,8,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*2*
(15,9,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*
(15,10,5,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*2*
(15,8,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{16}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*
(15,9,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*
(15,10,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{40}$	ψ	$\mathbf{6}^{40}$	*1*
(15,11,6,3)	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{16}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{40}$	ψ,ω	$\mathbf{6}^{40}$	*1*
(15,8,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*
(15,9,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*1*
(15,10,5,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*
(15,8,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*1*
(15,9,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,10,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,6,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*
(15,9,7,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*
(15,10,7,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,7,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*
(15,10,8,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*
(15,11,8,4)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*
(15,8,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{20}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*1*
(15,9,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,10,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,6,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*
(15,9,7,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*
(15,10,7,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,7,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*
(15,10,8,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,8,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*
(15,11,9,5)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*
(15,9,7,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*

y \ λ	(11,5,2,1)		(10,5,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T
(15,10,7,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,7,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*4*
(15,10,8,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*2*
(15,11,8,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*
(15,11,9,6)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*
(15,10,8,7)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{54}$	*3*
(15,11,8,7)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*3*
(15,11,9,7)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*
(15,11,9,8)	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{20}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{54}$	*2*

Table 23: $N = 24$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 24$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(12,6,3,2)	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(12,6,4,2)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(12,6,4,3)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(12,7,4,2)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(12,7,4,3)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(12,7,5,3)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(12,7,5,4)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ	$\mathbf{12}^1$	ζ,κ
(13,7,4,2)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,4,2)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,7,4,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,4,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,7,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,7,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,8,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,7,4,2)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,4,2)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,7,4,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,4,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,7,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,7,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ,κ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*6*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*7*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,8,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*8*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,9,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*7*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(12,8,4,2)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(12,8,4,3)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(12,8,5,3)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(12,8,6,3)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ

y \ λ	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(12,8,5,4)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(12,8,6,4)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(12,8,6,5)	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*3*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(13,9,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ζ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*8*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*8*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,8,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*6*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*8*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,8,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*7*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,8,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*7*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(14,10,8,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*7*	$\mathbf{12}^1$	ζ	$\mathbf{12}^1$	ζ
(15,8,4,2)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*	$\mathbf{12}^1$	ζ
(15,8,4,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*2*	$\mathbf{12}^1$	ζ
(15,8,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ
(15,9,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ
(15,10,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ
(15,8,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ
(15,9,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ
(15,10,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ
(15,11,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ
(15,8,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ
(15,9,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*7*	$\mathbf{12}^1$	ζ
(15,10,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*6*	$\mathbf{12}^1$	ζ
(15,8,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*	$\mathbf{12}^1$	ζ
(15,9,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*	$\mathbf{12}^1$	ζ
(15,10,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*	$\mathbf{12}^1$	ζ
(15,11,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*11*	$\mathbf{12}^1$	ζ
(15,9,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*8*	$\mathbf{12}^1$	ζ
(15,10,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,11,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*11*	$\mathbf{12}^1$	ζ
(15,10,8,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*	$\mathbf{12}^1$	ζ
(15,11,8,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*8*	$\mathbf{12}^1$	ζ
(15,8,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*	$\mathbf{12}^1$	ζ

y \ λ	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(15,9,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*	$\mathbf{12}^1$	ζ
(15,10,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*	$\mathbf{12}^1$	ζ
(15,11,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*11*	$\mathbf{12}^1$	ζ
(15,9,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ
(15,10,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,11,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,10,8,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*	$\mathbf{12}^1$	ζ
(15,11,8,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,11,9,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ
(15,9,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ
(15,10,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*14*	$\mathbf{12}^1$	ζ
(15,11,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,10,8,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*	$\mathbf{12}^1$	ζ
(15,11,8,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,11,9,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ
(15,10,8,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*	$\mathbf{12}^1$	ζ
(15,11,8,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*	$\mathbf{12}^1$	ζ
(15,11,9,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*10*	$\mathbf{12}^1$	ζ
(15,11,9,8)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*	$\mathbf{12}^1$	ζ
(16,8,4,2)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*0*
(16,8,4,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*1*
(16,8,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*
(16,9,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*6*
(16,10,5,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*
(16,8,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*3*
(16,9,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*
(16,10,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*
(16,11,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*
(16,12,6,3)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*4*
(16,8,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*
(16,9,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*7*
(16,10,5,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*6*
(16,8,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*4*
(16,9,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*
(16,10,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*
(16,11,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*
(16,12,6,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*11*
(16,9,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*7*
(16,10,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*
(16,11,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*
(16,12,7,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*11*
(16,10,8,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*
(16,11,8,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*9*
(16,12,8,4)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*8*
(16,8,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*5*

λ / y	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(16,9,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*
(16,10,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*13*
(16,11,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*
(16,12,6,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*12*
(16,9,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*8*
(16,10,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*14*
(16,11,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*14*
(16,12,7,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*14*
(16,10,8,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*
(16,11,8,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*16*
(16,12,8,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*
(16,11,9,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*
(16,12,9,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*
(16,12,10,5)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*
(16,9,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*8*
(16,10,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*14*
(16,11,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*15*
(16,12,7,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*14*
(16,10,8,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*
(16,11,8,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*16*
(16,12,8,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*13*
(16,11,9,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*
(16,12,9,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*14*
(16,12,10,6)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*
(16,10,8,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*10*
(16,11,8,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*15*
(16,12,8,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*12*
(16,11,9,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*11*
(16,12,9,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*12*
(16,12,10,7)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*10*
(16,11,9,8)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ	$\mathbf{12}^1$	*12*
(16,12,9,8)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*12*
(16,12,10,8)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*11*
(16,12,10,9)	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	ψ,ω	$\mathbf{12}^1$	*9*

Table 24: $N = 25$, $t = 4$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 25$ rows, strength $t = 4$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	time
(13,7,4,2)	$\mathbf{5}^{1}$	*0*	$\mathbf{5}^{3}$	*0*	$\mathbf{5}^{21}$	*0*	$\mathbf{5}^{21}$	ζ	$\mathbf{5}^{21}$	ζ	$\mathbf{5}^{21}$	κ	$\mathbf{5}^{21}$	ζ,κ	$\mathbf{5}^{21}$	ζ,κ	$\mathbf{5}^{21}$	ζ,κ	$\mathbf{5}^{21}$	ζ,κ
(13,7,4,3)	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ
(13,7,5,3)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ
(13,7,5,4)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ	$\mathbf{12}^{5}$	ζ,κ
(13,8,4,2)	$\mathbf{5}^{1}$	ω	$\mathbf{5}^{3}$	ω	$\mathbf{6}^{3}$	*0*	$\mathbf{6}^{3}$	ζ	$\mathbf{6}^{3}$	ζ	$\mathbf{6}^{3}$	*0*	$\mathbf{6}^{3}$	ζ	$\mathbf{6}^{3}$	ζ	$\mathbf{6}^{3}$	ζ	$\mathbf{6}^{3}$	ζ
(13,8,4,3)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(13,8,5,3)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(13,8,6,3)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(13,8,5,4)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(13,8,6,4)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(13,8,6,5)	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	ω	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,7,4,2)	$\mathbf{5}^{1}$	ψ	$\mathbf{5}^{3}$	ψ	$\mathbf{5}^{21}$	ψ	$\mathbf{5}^{25}$	*0*	$\mathbf{5}^{25}$	ζ	$\mathbf{5}^{21}$	ψ,κ	$\mathbf{5}^{27}$	*0*	$\mathbf{5}^{27}$	ζ	$\mathbf{5}^{27}$	ζ	$\mathbf{5}^{27}$	ζ
(14,8,4,2)	$\mathbf{5}^{1}$	ψ,ω	$\mathbf{5}^{3}$	ψ,ω	$\mathbf{6}^{3}$	ψ	$\mathbf{6}^{4}$	*0*	$\mathbf{6}^{4}$	ζ	$\mathbf{6}^{3}$	ψ	$\mathbf{6}^{4}$	*0*	$\mathbf{6}^{4}$	ζ	$\mathbf{6}^{4}$	ζ	$\mathbf{6}^{4}$	ζ
(14,7,4,3)	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ,κ	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,8,4,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,7,5,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ,κ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,8,5,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*5*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,5,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*7*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,8,6,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*5*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,6,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*6*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,7,5,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ,κ	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,8,5,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*7*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,5,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*10*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,8,6,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*6*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,6,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*5*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*17*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,7,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*12*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,8,6,5)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*6*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,6,5)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*5*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*16*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,7,5)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*13*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(14,9,7,6)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*13*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,8,4,2)	$\mathbf{5}^{1}$	ψ,ω	$\mathbf{5}^{3}$	ψ,ω	$\mathbf{6}^{3}$	ψ	$\mathbf{6}^{4}$	ψ	$\mathbf{6}^{6}$	*0*	$\mathbf{6}^{3}$	ψ	$\mathbf{6}^{4}$	ψ	$\mathbf{6}^{6}$	*0*	$\mathbf{6}^{6}$	ζ	$\mathbf{6}^{6}$	ζ
(15,8,4,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*0*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*1*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,8,5,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*7*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,9,5,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*10*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,10,5,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*8*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,8,6,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*6*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,9,6,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*3*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*9*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,10,6,3)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*2*	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*8*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,8,5,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*8*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,9,5,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*4*	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*14*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ
(15,10,5,4)	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	*5*	$\mathbf{12}^{5}$	ψ,ω	$\mathbf{12}^{5}$	ψ	$\mathbf{12}^{5}$	*14*	$\mathbf{12}^{5}$	ζ	$\mathbf{12}^{5}$	ζ

λ / y	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(15,8,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	3	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	7	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,9,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	6	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	29	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	33	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,9,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	22	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	6	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	35	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	4	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	26	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,8,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	3	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	7	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,9,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	6	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	29	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	6	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	35	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,9,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	24	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	6	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	42	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	31	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,9,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	23	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	6	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	41	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	4	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	31	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,10,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	31	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,5,3)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	1	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,6,3)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	1	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,5,4)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	1	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,6,4)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	2	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,7,4)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	1	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,6,5)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	2	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,7,5)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	2	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(13,9,7,6)	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	1	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,5,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	7	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	5	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,5,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	9	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	15	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	16	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	11	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	15	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	15	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	12	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	17	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	12	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(14,10,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ζ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	12	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	9	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	30	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	30	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	23	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	32	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	38	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	39	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,9,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	26	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	39	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ

λ / y	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(15,11,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,9,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*24*	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,9,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*25*	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(15,11,9,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*27*	$\mathbf{12}^5$	ζ	$\mathbf{12}^5$	ζ
(16,8,4,2)	$\mathbf{5}^1$	ψ,ω	$\mathbf{5}^3$	ψ,ω	$\mathbf{6}^3$	ψ	$\mathbf{6}^4$	ψ	$\mathbf{6}^6$	ψ	$\mathbf{6}^3$	ψ	$\mathbf{6}^4$	ψ	$\mathbf{6}^6$	ψ	$\mathbf{6}^6$	*0*	$\mathbf{6}^6$	ζ
(16,8,4,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*0*	$\mathbf{12}^5$	ζ
(16,8,5,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*7*	$\mathbf{12}^5$	ζ
(16,9,5,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*11*	$\mathbf{12}^5$	ζ
(16,10,5,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*10*	$\mathbf{12}^5$	ζ
(16,8,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*5*	$\mathbf{12}^5$	ζ
(16,9,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*10*	$\mathbf{12}^5$	ζ
(16,10,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*11*	$\mathbf{12}^5$	ζ
(16,11,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*11*	$\mathbf{12}^5$	ζ
(16,12,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*9*	$\mathbf{12}^5$	ζ
(16,8,5,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*8*	$\mathbf{12}^5$	ζ
(16,9,5,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*12*	$\mathbf{12}^5$	ζ
(16,10,5,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*14*	$\mathbf{12}^5$	ζ
(16,8,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*6*	$\mathbf{12}^5$	ζ
(16,9,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*28*	$\mathbf{12}^5$	ζ
(16,10,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ
(16,11,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ
(16,12,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*34*	$\mathbf{12}^5$	ζ
(16,9,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*22*	$\mathbf{12}^5$	ζ
(16,10,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*41*	$\mathbf{12}^5$	ζ
(16,11,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*39*	$\mathbf{12}^5$	ζ
(16,12,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*37*	$\mathbf{12}^5$	ζ
(16,10,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*30*	$\mathbf{12}^5$	ζ
(16,11,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*31*	$\mathbf{12}^5$	ζ
(16,12,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*28*	$\mathbf{12}^5$	ζ
(16,8,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*6*	$\mathbf{12}^5$	ζ
(16,9,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*32*	$\mathbf{12}^5$	ζ
(16,10,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*41*	$\mathbf{12}^5$	ζ
(16,11,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*43*	$\mathbf{12}^5$	ζ
(16,12,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*39*	$\mathbf{12}^5$	ζ
(16,9,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*26*	$\mathbf{12}^5$	ζ
(16,10,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*46*	$\mathbf{12}^5$	ζ
(16,11,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*47*	$\mathbf{12}^5$	ζ
(16,12,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*46*	$\mathbf{12}^5$	ζ
(16,10,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ
(16,11,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*48*	$\mathbf{12}^5$	ζ
(16,12,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*46*	$\mathbf{12}^5$	ζ
(16,11,9,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*41*	$\mathbf{12}^5$	ζ
(16,12,9,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*50*	$\mathbf{12}^5$	ζ
(16,12,10,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*34*	$\mathbf{12}^5$	ζ

y \ λ	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time	$CAK_\lambda^{y\#}$	time
(16,9,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*25*	$\mathbf{12}^5$	ζ
(16,10,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*47*	$\mathbf{12}^5$	ζ
(16,11,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*49*	$\mathbf{12}^5$	ζ
(16,12,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*45*	$\mathbf{12}^5$	ζ
(16,10,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ
(16,11,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*54*	$\mathbf{12}^5$	ζ
(16,12,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*51*	$\mathbf{12}^5$	ζ
(16,11,9,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*42*	$\mathbf{12}^5$	ζ
(16,12,9,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*50*	$\mathbf{12}^5$	ζ
(16,12,10,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*36*	$\mathbf{12}^5$	ζ
(16,10,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*37*	$\mathbf{12}^5$	ζ
(16,11,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*52*	$\mathbf{12}^5$	ζ
(16,12,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*49*	$\mathbf{12}^5$	ζ
(16,11,9,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*40*	$\mathbf{12}^5$	ζ
(16,12,9,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*50*	$\mathbf{12}^5$	ζ
(16,12,10,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*38*	$\mathbf{12}^5$	ζ
(16,11,9,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*39*	$\mathbf{12}^5$	ζ
(16,12,9,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*52*	$\mathbf{12}^5$	ζ
(16,12,10,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*37*	$\mathbf{12}^5$	ζ
(16,12,10,9)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*32*	$\mathbf{12}^5$	ζ
(17,9,5,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*11*
(17,10,5,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*9*
(17,9,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*10*
(17,10,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*9*
(17,11,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*10*
(17,12,6,3)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*7*
(17,9,5,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*15*
(17,10,5,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*14*
(17,9,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*31*
(17,10,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*34*
(17,11,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*
(17,12,6,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*35*
(17,9,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*22*
(17,10,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*
(17,11,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*40*
(17,12,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*41*
(17,13,7,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*37*
(17,10,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*30*
(17,11,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*30*
(17,12,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*32*
(17,13,8,4)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*26*
(17,9,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*30*
(17,10,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*36*
(17,11,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*36*
(17,12,6,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*34*

λ / y	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(17,9,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*23*
(17,10,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*45*
(17,11,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*48*
(17,12,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*48*
(17,13,7,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*48*
(17,10,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*37*
(17,11,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*48*
(17,12,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*53*
(17,13,8,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*50*
(17,11,9,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*38*
(17,12,9,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*46*
(17,13,9,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*46*
(17,12,10,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*41*
(17,13,10,5)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*35*
(17,9,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*25*
(17,10,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*48*
(17,11,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*53*
(17,12,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*53*
(17,13,7,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*48*
(17,10,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*37*
(17,11,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*53*
(17,12,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*57*
(17,13,8,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*50*
(17,11,9,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*41*
(17,12,9,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*54*
(17,13,9,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*45*
(17,12,10,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*42*
(17,13,10,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*52*
(17,13,11,6)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*33*
(17,10,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*35*
(17,11,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*57*
(17,12,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*54*
(17,13,8,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*53*
(17,11,9,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*40*
(17,12,9,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*59*
(17,13,9,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*51*
(17,12,10,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*40*
(17,13,10,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*51*
(17,13,11,7)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*34*
(17,11,9,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*36*
(17,12,9,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*58*
(17,13,9,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*59*
(17,12,10,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*42*
(17,13,10,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*52*
(17,13,11,8)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*37*

y \ λ	(12,6,3,1)		(12,6,2,1)		(12,5,2,1)		(11,5,2,1)		(10,5,2,1)		(12,4,2,1)		(11,4,2,1)		(10,4,2,1)		(9,4,2,1)		(8,4,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time	$\mathsf{CAK}_\lambda^{y\#}$	time
(17,12,10,9)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ	$\mathbf{12}^5$	*42*
(17,13,10,9)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*47*
(17,13,11,9)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*35*
(17,13,11,10)	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	ψ,ω	$\mathbf{12}^5$	*37*

Binary Balanced CAs of Strength $t = 5$

Table 25: $N = 52^{*}$, $t = 5$, $v = 2$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 52$ rows, strength $t = 5$ and a binary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(26,13,6,3,1)		(26,12,6,3,1)		(25,12,6,3,1)		(24,12,6,3,1)		(26,13,6,2,1)		(26,12,6,2,1)		(25,12,6,2,1)		(24,12,6,2,1)	
	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	CAK$_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(26,13,7,4,2)	$\boldsymbol{\emptyset}$	0	$\emptyset$	κ	$\emptyset$	ζ,κ	$\emptyset$	ζ,κ	$\mathbf{6}^{1}$	0	6^{1}	κ	6^{1}	ζ,κ	6^{1}	ζ,κ
(26,13,7,4,3)	$\boldsymbol{\emptyset}$	0	$\emptyset$	κ	$\emptyset$	ζ,κ	$\emptyset$	ζ,κ	$\mathbf{6}^{12}$	0	6^{12}	κ	6^{12}	ζ,κ	6^{12}	ζ,κ
(26,14,7,4,2)	$\emptyset$	ω	$\boldsymbol{\emptyset}$	0	$\emptyset$	ζ	$\emptyset$	ζ	6^{1}	ω	$\mathbf{6}^{6}$	0	6^{6}	ζ	6^{6}	ζ
(26,14,8,4,2)	$\emptyset$	ω	$\mathbf{5}^{1}$	0	5^{1}	ζ	5^{1}	ζ	6^{1}	ω	$\mathbf{6}^{11}$	0	6^{11}	ζ	6^{11}	ζ
(26,14,7,4,3)	$\emptyset$	ω	$\mathbf{6}^{1}$	0	6^{1}	ζ	6^{1}	ζ	6^{12}	ω	$\mathbf{7}^{16}$	1	7^{16}	ζ	7^{16}	ζ
(26,14,8,4,3)	$\emptyset$	ω	$\mathbf{6}^{3}$	0	6^{3}	ζ	6^{3}	ζ	6^{12}	ω	$\mathbf{8}^{1}$	1	8^{1}	ζ	8^{1}	ζ
(26,14,8,5,3)	$\emptyset$	ω	$\mathbf{6}^{5}$	0	6^{5}	ζ	6^{5}	ζ	7^{1}	ω	$\mathbf{8}^{4}$	4	8^{4}	ζ	8^{4}	ζ
(26,14,8,5,4)	$\emptyset$	ω	$\mathbf{6}^{7}$	0	6^{7}	ζ	6^{7}	ζ	7^{1}	ω	$\mathbf{8}^{5}$	4	8^{5}	ζ	8^{5}	ζ
(27,14,7,4,2)	$\emptyset$	ψ,ω	$\emptyset$	ψ	$\mathbf{6}^{1}$	0	6^{1}	ζ	6^{1}	ψ,ω	6^{6}	ψ	$\mathbf{6}^{7}$	0	6^{7}	ζ
(27,14,8,4,2)	$\emptyset$	ψ,ω	5^{1}	ψ	$\mathbf{6}^{1}$	0	6^{1}	ζ	6^{1}	ψ,ω	6^{11}	ψ	$\mathbf{6}^{20}$	1	6^{20}	ζ
(27,15,8,4,2)	$\emptyset$	ψ,ω	5^{1}	ψ,ω	$\mathbf{6}^{1}$	0	6^{1}	ζ	6^{1}	ψ,ω	6^{11}	ψ,ω	$\mathbf{6}^{28}$	0	6^{28}	ζ
(27,14,7,4,3)	$\emptyset$	ψ,ω	6^{1}	ψ	$\mathbf{6}^{2}$	0	6^{2}	ζ	6^{12}	ψ,ω	7^{16}	ψ	$\mathbf{7}^{17}$	1	7^{17}	ζ
(27,14,8,4,3)	$\emptyset$	ψ,ω	6^{3}	ψ	$\mathbf{6}^{14}$	0	6^{14}	ζ	6^{12}	ψ,ω	8^{1}	ψ	$\mathbf{8}^{1}$	4	8^{1}	ζ
(27,15,8,4,3)	$\emptyset$	ψ,ω	6^{3}	ψ,ω	$\mathbf{6}^{15}$	0	6^{15}	ζ	6^{12}	ψ,ω	8^{1}	ψ,ω	$\mathbf{8}^{1}$	3	8^{1}	ζ
(27,14,8,5,3)	$\emptyset$	ψ,ω	6^{5}	ψ	$\mathbf{6}^{46}$	0	6^{46}	ζ	7^{1}	ψ,ω	8^{4}	ψ	$\mathbf{8}^{4}$	41	8^{4}	ζ
(27,15,8,5,3)	$\emptyset$	ψ,ω	6^{5}	ψ,ω	$\mathbf{6}^{105}$	0	6^{105}	ζ	7^{1}	ψ,ω	8^{4}	ψ,ω	$\mathbf{8}^{4}$	88	8^{4}	ζ
(27,15,9,5,3)	$\emptyset$	ψ,ω	6^{5}	ψ,ω	$\mathbf{6}^{139}$	1	6^{139}	ζ	7^{1}	ψ,ω	8^{4}	ψ,ω	$\mathbf{8}^{4}$	95	8^{4}	ζ
(27,15,9,6,3)	$\emptyset$	ψ,ω	6^{5}	ψ,ω	$\mathbf{6}^{146}$	0	6^{146}	ζ	7^{1}	ψ,ω	8^{6}	ψ,ω	$\mathbf{8}^{7}$	81	8^{7}	ζ
(27,14,8,5,4)	$\emptyset$	ψ,ω	6^{7}	ψ	$\mathbf{6}^{54}$	0	6^{54}	ζ	7^{1}	ψ,ω	8^{5}	ψ	$\mathbf{8}^{5}$	44	8^{5}	ζ
(27,15,8,5,4)	$\emptyset$	ψ,ω	6^{7}	ψ,ω	$\mathbf{6}^{126}$	1	6^{126}	ζ	7^{1}	ψ,ω	8^{5}	ψ,ω	$\mathbf{8}^{5}$	104	8^{5}	ζ
(27,15,9,5,4)	$\emptyset$	ψ,ω	6^{7}	ψ,ω	$\mathbf{6}^{171}$	0	6^{171}	ζ	7^{1}	ψ,ω	8^{5}	ψ,ω	$\mathbf{8}^{5}$	120	8^{5}	ζ
(27,15,9,6,4)	$\emptyset$	ψ,ω	6^{7}	ψ,ω	$\mathbf{7}^{1}$	1	7^{1}	ζ	7^{1}	ψ,ω	8^{7}	ψ,ω	$\mathbf{8}^{8}$	158	8^{8}	ζ
(27,15,9,6,5)	$\emptyset$	ψ,ω	6^{7}	ψ,ω	$\mathbf{7}^{1}$	0	7^{1}	ζ	7^{1}	ψ,ω	8^{7}	ψ,ω	$\mathbf{8}^{8}$	160	8^{8}	ζ
(28,14,7,4,2)	$\emptyset$	ψ,ω	$\emptyset$	ψ	6^{1}	ψ	$\mathbf{6}^{1}$	0	6^{1}	ψ,ω	6^{6}	ψ	6^{7}	ψ	$\mathbf{6}^{7}$	0
(28,14,8,4,2)	$\emptyset$	ψ,ω	5^{1}	ψ	6^{1}	ψ	$\mathbf{6}^{1}$	0	6^{1}	ψ,ω	6^{11}	ψ	6^{20}	ψ	$\mathbf{6}^{21}$	0
(28,15,8,4,2)	$\emptyset$	ψ,ω	5^{1}	ψ,ω	6^{1}	ψ	$\mathbf{6}^{1}$	0	6^{1}	ψ,ω	6^{11}	ψ,ω	6^{28}	ψ	$\mathbf{6}^{33}$	0
(28,16,8,4,2)	$\emptyset$	ψ,ω	5^{1}	ψ,ω	6^{1}	ψ,ω	$\mathbf{6}^{1}$	0	6^{1}	ψ,ω	6^{11}	ψ,ω	6^{28}	ψ,ω	$\mathbf{6}^{38}$	1
(28,14,7,4,3)	$\emptyset$	ψ,ω	6^{1}	ψ	6^{2}	ψ	$\mathbf{6}^{2}$	0	6^{12}	ψ,ω	7^{16}	ψ	7^{17}	ψ	$\mathbf{7}^{17}$	1
(28,14,8,4,3)	$\emptyset$	ψ,ω	6^{3}	ψ	6^{14}	ψ	$\mathbf{6}^{15}$	0	6^{12}	ψ,ω	8^{1}	ψ	8^{1}	ψ	$\mathbf{8}^{1}$	4

y \ λ	(26,13,6,3,1)		(26,12,6,3,1)		(25,12,6,3,1)		(24,12,6,3,1)		(26,13,6,2,1)		(26,12,6,2,1)		(25,12,6,2,1)		(24,12,6,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T
(28,15,8,4,3)	$\emptyset$	ψ,ω	$\mathbf{6}^3$	ψ,ω	$\mathbf{6}^{15}$	ψ	$\mathbf{6}^{17}$	*0*	$\mathbf{6}^{12}$	ψ,ω	$\mathbf{8}^1$	ψ,ω	$\mathbf{8}^1$	ψ	$\mathbf{8}^1$	*5*
(28,16,8,4,3)	$\emptyset$	ψ,ω	$\mathbf{6}^3$	ψ,ω	$\mathbf{6}^{15}$	ψ,ω	$\mathbf{6}^{17}$	*0*	$\mathbf{6}^{12}$	ψ,ω	$\mathbf{8}^1$	ψ,ω	$\mathbf{8}^1$	ψ,ω	$\mathbf{8}^1$	*5*
(28,14,8,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ	$\mathbf{6}^{46}$	ψ	$\mathbf{6}^{50}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^4$	ψ	$\mathbf{8}^4$	ψ	$\mathbf{8}^4$	*40*
(28,15,8,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{105}$	ψ	$\mathbf{6}^{135}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	ψ	$\mathbf{8}^4$	*141*
(28,16,8,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{105}$	ψ,ω	$\mathbf{6}^{137}$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	*115*
(28,15,9,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{139}$	ψ	$\mathbf{6}^{201}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	ψ	$\mathbf{8}^4$	*149*
(28,16,9,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{139}$	ψ,ω	$\mathbf{6}^{218}$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	*150*
(28,16,10,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{139}$	ψ,ω	$\mathbf{6}^{220}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	ψ,ω	$\mathbf{8}^4$	*154*
(28,15,9,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{146}$	ψ	$\mathbf{6}^{215}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ,ω	$\mathbf{8}^7$	ψ	$\mathbf{8}^7$	*119*
(28,16,9,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{146}$	ψ,ω	$\mathbf{6}^{239}$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^7$	*128*
(28,16,10,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{146}$	ψ,ω	$\mathbf{6}^{246}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^7$	*120*
(28,14,8,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ	$\mathbf{6}^{54}$	ψ	$\mathbf{6}^{59}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^5$	ψ	$\mathbf{8}^5$	ψ	$\mathbf{8}^5$	*45*
(28,15,8,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ψ	$\mathbf{6}^{164}$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	ψ	$\mathbf{8}^5$	*140*
(28,16,8,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ψ,ω	$\mathbf{6}^{168}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	*140*
(28,15,9,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{171}$	ψ	$\mathbf{6}^{248}$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	ψ	$\mathbf{8}^5$	*204*
(28,16,9,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{171}$	ψ,ω	$\mathbf{6}^{273}$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	*209*
(28,16,10,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{171}$	ψ,ω	$\mathbf{6}^{276}$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	ψ,ω	$\mathbf{8}^5$	*206*
(28,15,9,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ	$\mathbf{7}^2$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*306*
(28,16,9,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^3$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*371*
(28,16,10,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^4$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*324*
(28,16,10,7,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^5$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*388*
(28,15,9,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ	$\mathbf{7}^2$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*308*
(28,16,9,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^3$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*370*
(28,16,10,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^4$	*0*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*389*
(28,16,10,7,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^8$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*365*
(28,16,10,7,6)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^8$	*1*	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*355*
(26,13,7,5,3)	$\emptyset$	ω	$\emptyset$	κ,ω	$\emptyset$	ζ,κ,ω	$\emptyset$	ζ,κ,ω	$\mathbf{7}^1$	*0*	$\mathbf{7}^1$	κ	$\mathbf{7}^1$	ζ,κ	$\mathbf{7}^1$	ζ,κ
(26,13,7,5,4)	$\emptyset$	ω	$\emptyset$	κ,ω	$\emptyset$	ζ,κ,ω	$\emptyset$	ζ,κ,ω	$\mathbf{7}^1$	*0*	$\mathbf{7}^1$	κ	$\mathbf{7}^1$	ζ,κ	$\mathbf{7}^1$	ζ,κ
(26,14,7,5,3)	$\emptyset$	ω	$\mathbf{6}^1$	ω	$\mathbf{6}^1$	ζ,ω	$\mathbf{6}^1$	ζ,ω	$\mathbf{7}^1$	ω	$\mathbf{8}^2$	*2*	$\mathbf{8}^2$	ζ	$\mathbf{8}^2$	ζ
(26,14,8,6,3)	$\emptyset$	ω	$\mathbf{6}^5$	ω	$\mathbf{6}^5$	ζ,ω	$\mathbf{6}^5$	ζ,ω	$\mathbf{7}^1$	ω	$\mathbf{8}^6$	*3*	$\mathbf{8}^6$	ζ	$\mathbf{8}^6$	ζ
(26,14,7,5,4)	$\emptyset$	ω	$\mathbf{6}^1$	ω	$\mathbf{6}^1$	ζ,ω	$\mathbf{6}^1$	ζ,ω	$\mathbf{7}^1$	ω	$\mathbf{8}^2$	*2*	$\mathbf{8}^2$	ζ	$\mathbf{8}^2$	ζ
(26,14,8,6,4)	$\emptyset$	ω	$\mathbf{6}^7$	ω	$\mathbf{6}^7$	ζ,ω	$\mathbf{6}^7$	ζ,ω	$\mathbf{7}^1$	ω	$\mathbf{8}^7$	*3*	$\mathbf{8}^7$	ζ	$\mathbf{8}^7$	ζ
(26,14,8,6,5)	$\emptyset$	ω	$\mathbf{6}^7$	ω	$\mathbf{6}^7$	ζ,ω	$\mathbf{6}^7$	ζ,ω	$\mathbf{7}^1$	ω	$\mathbf{8}^7$	*3*	$\mathbf{8}^7$	ζ	$\mathbf{8}^7$	ζ
(27,14,7,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^2$	ω	$\mathbf{6}^2$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^2$	ψ	$\mathbf{8}^2$	*1*	$\mathbf{8}^2$	ζ

λ / y	(26,13,6,3,1)		(26,12,6,3,1)		(25,12,6,3,1)		(24,12,6,3,1)		(26,13,6,2,1)		(26,12,6,2,1)		(25,12,6,2,1)		(24,12,6,2,1)	
	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T	$\mathsf{CAK}_\lambda^{y\#}$	T
(27,14,8,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{46}$	ω	$\mathbf{6}^{46}$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ	$\mathbf{8}^7$	*31*	$\mathbf{8}^7$	ζ
(27,15,8,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{105}$	ω	$\mathbf{6}^{105}$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ,ω	$\mathbf{8}^7$	*67*	$\mathbf{8}^7$	ζ
(27,14,7,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^2$	ω	$\mathbf{6}^2$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^2$	ψ	$\mathbf{8}^2$	*1*	$\mathbf{8}^2$	ζ
(27,14,8,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{54}$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ	$\mathbf{8}^8$	*36*	$\mathbf{8}^8$	ζ
(27,15,8,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ω	$\mathbf{6}^{126}$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	*81*	$\mathbf{8}^8$	ζ
(27,15,9,7,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ω	$\mathbf{7}^1$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	*125*	$\mathbf{8}^8$	ζ
(27,14,8,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{54}$	ω	$\mathbf{6}^{54}$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ	$\mathbf{8}^8$	*40*	$\mathbf{8}^8$	ζ
(27,15,8,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ω	$\mathbf{6}^{126}$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	*81*	$\mathbf{8}^8$	ζ
(27,15,9,7,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ω	$\mathbf{7}^1$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	*112*	$\mathbf{8}^8$	ζ
(27,15,9,7,6)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ω	$\mathbf{7}^1$	ζ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	*110*	$\mathbf{8}^8$	ζ
(28,14,7,5,3)	$\emptyset$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^2$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^2$	ψ	$\mathbf{8}^2$	ψ	$\mathbf{8}^2$	*2*
(28,14,8,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{46}$	ψ,ω	$\mathbf{6}^{50}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ	$\mathbf{8}^7$	ψ	$\mathbf{8}^7$	*32*
(28,15,8,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{105}$	ψ,ω	$\mathbf{6}^{135}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ,ω	$\mathbf{8}^7$	ψ	$\mathbf{8}^7$	*97*
(28,16,8,6,3)	$\emptyset$	ψ,ω	$\mathbf{6}^5$	ψ,ω	$\mathbf{6}^{105}$	ψ,ω	$\mathbf{6}^{137}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^6$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^7$	*87*
(28,14,7,5,4)	$\emptyset$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^2$	ψ,ω	$\mathbf{6}^2$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^2$	ψ	$\mathbf{8}^2$	ψ	$\mathbf{8}^2$	*2*
(28,14,8,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{59}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*38*
(28,15,8,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ψ,ω	$\mathbf{6}^{164}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*123*
(28,16,8,6,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ψ,ω	$\mathbf{6}^{168}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*91*
(28,15,9,7,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^2$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*224*
(28,16,9,7,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^3$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*252*
(28,16,10,8,4)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^5$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*265*
(28,14,8,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{54}$	ψ,ω	$\mathbf{6}^{59}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*36*
(28,15,8,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ψ,ω	$\mathbf{6}^{164}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*116*
(28,16,8,6,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{6}^{126}$	ψ,ω	$\mathbf{6}^{168}$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*103*
(28,15,9,7,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^2$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*208*
(28,16,9,7,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^3$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*253*
(28,16,10,8,5)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^8$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*270*
(28,15,9,7,6)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^2$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ	$\mathbf{8}^8$	*220*
(28,16,9,7,6)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^3$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*274*
(28,16,10,8,6)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^8$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*248*
(28,16,10,8,7)	$\emptyset$	ψ,ω	$\mathbf{6}^7$	ψ,ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^8$	ω	$\mathbf{7}^1$	ψ,ω	$\mathbf{8}^7$	ψ,ω	$\mathbf{8}^8$	ψ,ω	$\mathbf{8}^8$	*258*

Ternary Balanced CAs

Ternary Balanced CAs of Strength $t = 2$

Table 26: $N = 9^*$, $t = 2$, $v = 3$ – Results for the classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 9$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(3,1)	$\mathbf{4}^1$	*0*

Table 27: $N = 10$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 10$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(4,2)	$\mathbf{4}^2$	*0*

Table 28: $N = 11^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 11$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(4,2)	$\mathbf{5}^3$	*0*
(5,2)	$\mathbf{5}^3$	*0*
(5,3)	$\mathbf{5}^3$	*0*

Table 29: $N = 12^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 12$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)		(3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(4,2)	$\mathbf{7}^1$	*0*	$\mathbf{7}^1$	ζ
(5,2)	$\mathbf{7}^1$	ψ	$\mathbf{7}^1$	*0*
(6,2)	$\mathbf{7}^1$	ψ	$\mathbf{7}^1$	*1*
(5,3)	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^1$	*0*
(6,3)	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^1$	*0*
(6,4)	$\mathbf{7}^1$	ψ,ω	$\mathbf{7}^1$	*1*

Table 30: $N = 13^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 13$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)		(3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(5,2)	$\mathbf{6}^{5079}$	*6*	$\mathbf{6}^{5315}$	*6*
(5,3)	$\mathbf{9}^{4}$	*177*	$\mathbf{9}^{4}$	*277*
(6,2)	$\mathbf{6}^{5079}$	ψ	$\mathbf{6}^{5335}$	*6*
(6,3)	$\mathbf{9}^{4}$	ψ	$\mathbf{9}^{4}$	*348*
(7,3)	$\mathbf{9}^{4}$	ψ	$\mathbf{9}^{4}$	*357*
(6,4)	$\mathbf{9}^{4}$	ψ,ω	$\mathbf{9}^{4}$	*335*
(7,4)	$\mathbf{9}^{4}$	ψ,ω	$\mathbf{9}^{4}$	*296*
(7,5)	$\mathbf{9}^{4}$	ψ,ω	$\mathbf{9}^{4}$	*301*

Table 31: $N = 14^*$, $t = 2$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 14$ rows, strength $t = 2$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)		(3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(5,2)	$\mathbf{7}^{370}$	*38*	$\mathbf{7}^{370}$	ζ
(6,2)	$\mathbf{7}^{381}$	*44*	$\mathbf{7}^{381}$	*43*
(5,3)	t.o.		-	ζ
(6,3)	-		t.o.	
(6,4)	-		-	
(7,3)	-	ψ	-	
(8,3)	-	ψ	-	
(7,4)	-	ψ	-	
(8,4)	-	ψ	-	
(7,5)	-	ψ,ω	-	
(8,5)	-	ψ,ω	-	
(8,6)	-	ψ,ω	-	

Ternary Balanced CAs of Strength $t = 3$

Table 32: $N = 27^*$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 27$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(9,3,1)	$\mathbf{4}^{1}$	*0*

Table 33: $N = 28$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 28$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where $\mathtt{T}$ denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	$\mathtt{T}$
(10,4,2)	$\mathbf{4}^{2}$	*0*

Table 34: $N = 29$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 29$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where $\mathtt{T}$ denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	$\mathtt{T}$
(10,4,2)	$\mathbf{4}^{7}$	*0*
(11,4,2)	$\mathbf{4}^{11}$	*0*
(11,5,2)	$\mathbf{4}^{15}$	*0*
(11,5,3)	$\mathbf{4}^{19}$	*0*

Table 35: $N = 30$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 30$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where $\mathtt{T}$ denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	$\mathtt{T}$	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	$\mathtt{T}$
(10,4,2)	$\mathbf{4}^{5}$	*0*	$\mathbf{4}^{5}$	ζ
(11,4,2)	$\mathbf{4}^{5}$	ψ	$\mathbf{4}^{33}$	*0*
(12,4,2)	$\mathbf{4}^{5}$	ψ	$\mathbf{4}^{36}$	*0*
(11,5,2)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{88}$	*0*
(12,5,2)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{110}$	*1*
(12,6,2)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{114}$	*0*
(11,5,3)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{117}$	*0*
(12,5,3)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{159}$	*0*
(12,6,3)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{176}$	*0*
(12,6,4)	$\mathbf{4}^{5}$	ψ,ω	$\mathbf{4}^{182}$	*0*

Table 36: $N = 31$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 31$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where $\mathtt{T}$ denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	$\mathtt{T}$	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	$\mathtt{T}$
(11,4,2)	$\mathbf{4}^{28}$	*0*	$\mathbf{4}^{75}$	*0*

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(11,5,2)	$\mathbf{4}^{113}$	*0*	$\mathbf{4}^{424}$	*0*
(11,5,3)	$\mathbf{4}^{142}$	*0*	$\mathbf{4}^{585}$	*0*
(12,4,2)	$\mathbf{4}^{28}$	ψ	$\mathbf{4}^{87}$	*0*
(12,5,2)	$\mathbf{4}^{113}$	ψ	$\mathbf{4}^{1008}$	*1*
(13,5,2)	$\mathbf{4}^{113}$	ψ	$\mathbf{4}^{1049}$	*0*
(12,6,2)	$\mathbf{4}^{113}$	ψ,ω	$\mathbf{4}^{1037}$	*1*
(13,6,2)	$\mathbf{4}^{113}$	ψ,ω	$\mathbf{4}^{1088}$	*1*
(12,5,3)	$\mathbf{4}^{142}$	ψ	$\mathbf{4}^{1718}$	*2*
(13,5,3)	$\mathbf{4}^{142}$	ψ	$\mathbf{4}^{1827}$	*1*
(12,6,3)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2035}$	*1*
(13,6,3)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2303}$	*2*
(13,7,3)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2329}$	*1*
(12,6,4)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2091}$	*2*
(13,6,4)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2397}$	*1*
(13,7,4)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2448}$	*2*
(13,7,5)	$\mathbf{4}^{142}$	ψ,ω	$\mathbf{4}^{2457}$	*1*

Table 37: $N = 32$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 32$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(11,4,2)	$\mathbf{4}^{88}$	*0*	$\mathbf{4}^{88}$	ζ
(12,4,2)	$\mathbf{4}^{130}$	*0*	$\mathbf{4}^{174}$	*0*
(11,5,2)	$\mathbf{4}^{605}$	*0*	$\mathbf{4}^{605}$	ζ
(12,5,2)	$\mathbf{4}^{2829}$	*2*	$\mathbf{4}^{7593}$	*5*
(12,6,2)	$\mathbf{4}^{2950}$	*2*	$\mathbf{4}^{7887}$	*5*
(11,5,3)	$\mathbf{4}^{864}$	*0*	$\mathbf{4}^{864}$	ζ
(12,5,3)	$\mathbf{4}^{4566}$	*3*	$\mathbf{4}^{15279}$	*10*
(12,6,3)	$\mathbf{4}^{5257}$	*4*	$\mathbf{4}^{19299}$	*14*
(12,6,4)	$\mathbf{4}^{5321}$	*4*	$\mathbf{4}^{19713}$	*12*
(13,5,2)	$\mathbf{4}^{2829}$	ψ	$\mathbf{4}^{8981}$	*7*
(14,5,2)	$\mathbf{4}^{2829}$	ψ	$\mathbf{4}^{9006}$	*6*
(13,6,2)	$\mathbf{4}^{2950}$	ψ	$\mathbf{4}^{9635}$	*6*
(14,6,2)	$\mathbf{4}^{2950}$	ψ	$\mathbf{4}^{9699}$	*5*
(13,5,3)	$\mathbf{4}^{4566}$	ψ	$\mathbf{4}^{19301}$	*13*
(14,5,3)	$\mathbf{4}^{4566}$	ψ	$\mathbf{4}^{19393}$	*13*
(13,6,3)	$\mathbf{4}^{5257}$	ψ	$\mathbf{4}^{30440}$	*18*
(14,6,3)	$\mathbf{4}^{5257}$	ψ	$\mathbf{4}^{31360}$	*22*
(13,7,3)	$\mathbf{4}^{5257}$	ψ,ω	$\mathbf{4}^{30922}$	*21*
(14,7,3)	$\mathbf{4}^{5257}$	ψ,ω	$\mathbf{4}^{32109}$	*22*
(14,8,3)	$\mathbf{4}^{5257}$	ψ,ω	$\mathbf{4}^{32129}$	*20*
(13,6,4)	$\mathbf{4}^{5321}$	ψ	$\mathbf{4}^{32002}$	*21*
(14,6,4)	$\mathbf{4}^{5321}$	ψ	$\mathbf{4}^{33106}$	*22*
(13,7,4)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{33122}$	*21*
(14,7,4)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{34843}$	*22*
(14,8,4)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{34933}$	*20*

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(13,7,5)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{33212}$	*21*
(14,7,5)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{34994}$	*20*
(14,8,5)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{35124}$	*19*
(14,8,6)	$\mathbf{4}^{5321}$	ψ,ω	$\mathbf{4}^{35136}$	*20*

Table 38: $N = 33^*$, $t = 3$, $v = 3$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 33$ rows, strength $t = 3$ and a ternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(11,3,1)		(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}^{\#}}$	T
(11,4,2)	$\mathbf{6}^{1}$	*0*	$\mathbf{6}^{1}$	ζ	$\mathbf{6}^{1}$	ζ
(11,5,2)	$\mathbf{6}^{1}$	*1*	$\mathbf{6}^{1}$	ζ	$\mathbf{6}^{1}$	ζ
(11,5,3)	$\mathbf{6}^{1}$	*0*	$\mathbf{6}^{1}$	ζ	$\mathbf{6}^{1}$	ζ
(12,4,2)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*0*	$\mathbf{6}^{1}$	*0*
(12,5,2)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*21*	$\mathbf{6}^{1}$	*29*
(13,5,2)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*28*	$\mathbf{6}^{1}$	*49*
(12,6,2)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*20*	$\mathbf{6}^{1}$	*27*
(13,6,2)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*28*	$\mathbf{6}^{1}$	*48*
(12,5,3)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*44*	$\mathbf{6}^{1}$	*66*
(13,5,3)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*66*	$\mathbf{6}^{1}$	*127*
(12,6,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*51*	$\mathbf{6}^{1}$	*86*
(13,6,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*106*	$\mathbf{6}^{1}$	*251*
(13,7,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*103*	$\mathbf{6}^{1}$	*266*
(12,6,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*51*	$\mathbf{6}^{1}$	*90*
(13,6,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*99*	$\mathbf{6}^{1}$	*260*
(13,7,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*100*	$\mathbf{6}^{1}$	*271*
(13,7,5)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*95*	$\mathbf{6}^{1}$	*311*
(14,5,2)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*45*
(15,5,2)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*46*
(14,6,2)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*49*
(15,6,2)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*47*
(14,5,3)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*130*
(15,5,3)	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*123*
(14,6,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*306*
(15,6,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*285*
(14,7,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*326*
(15,7,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*341*
(14,8,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*300*
(15,8,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*316*
(15,9,3)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*281*
(14,6,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*289*
(15,6,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*296*
(14,7,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*332*
(15,7,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ	$\mathbf{6}^{1}$	*376*
(14,8,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*362*
(15,8,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*321*
(15,9,4)	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	ψ,ω	$\mathbf{6}^{1}$	*298*

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(11,3,1)		(10,3,1)		(9,3,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(14,7,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*343*
(15,7,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ	$\mathbf{6}^1$	*342*
(14,8,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*332*
(15,8,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*333*
(15,9,5)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*333*
(14,8,6)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*370*
(15,8,6)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*339*
(15,9,6)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*300*
(15,9,7)	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	ψ,ω	$\mathbf{6}^1$	*290*

Quaternary Balanced CAs of Strength $t = 2$

Table 39: $N = 16$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 16$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ *T*, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(4,1)	$\mathbf{5}^1$	*0*

Table 40: $N = 17$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 17$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ *T*, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(5,2)	$\mathbf{5}^4$	*0*

Table 41: $N = 18$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 18$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ *T*, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(5,2)	$\mathbf{5}^{67}$	*0*
(6,2)	$\mathbf{5}^{134}$	*0*
(6,3)	$\mathbf{5}^{201}$	*1*

Table 42: $N = 19$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 19$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(5,2)	$\mathbf{6}^4$	*4*
(6,2)	$\mathbf{6}^4$	*18*
(7,2)	$\mathbf{6}^4$	*19*
(6,3)	$\mathbf{6}^4$	*63*
(7,3)	$\mathbf{6}^4$	*70*
(7,4)	$\mathbf{6}^4$	*71*

Table 43: $N = 20$, $t = 2$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 20$ rows, strength $t = 2$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(5,1)		(4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(5,2)	$\mathbf{6}^{745}$	*713*	$\mathbf{6}^{745}$	ζ
(6,2)	$\mathbf{6}^{745}$	ψ	t.o.	
(7,2)	$\mathbf{6}^{745}$	ψ	-	
(8,2)	$\mathbf{6}^{745}$	ψ	-	
(6,3)	$\mathbf{6}^{745}$	ψ,ω	-	
(7,3)	$\mathbf{6}^{745}$	ψ,ω	-	
(8,3)	$\mathbf{6}^{745}$	ψ,ω	-	
(7,4)	$\mathbf{6}^{745}$	ψ,ω	-	
(8,4)	$\mathbf{6}^{745}$	ψ,ω	-	
(8,5)	$\mathbf{6}^{745}$	ψ,ω	-	

Table 44: $N = 64$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 64$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(16,4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(16,4,1)	$\mathbf{6}^1$	*7*

Table 45: $N = 65$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 65$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(16,4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(17,5,2)	$\mathbf{6}^{3}$	*2*

Table 46: $N = 66$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 66$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(16,4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(17,5,2)	$\mathbf{6}^{41}$	*5*
(18,5,2)	$\mathbf{6}^{77}$	*11*
(18,6,2)	$\mathbf{6}^{122}$	*17*
(18,6,3)	$\mathbf{6}^{157}$	*20*

Table 47: $N = 67$, $t = 3$, $v = 4$ – Classification of $(\boldsymbol{\lambda}, \boldsymbol{y})$-balanced CAs with $N = 67$ rows, strength $t = 3$ and a quaternary alphabet in the format $\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$ T, where T denotes the time (in seconds).

$\boldsymbol{y}$ \ $\boldsymbol{\lambda}$	(16,4,1)	
	$\mathsf{CAK}_{\boldsymbol{\lambda}}^{\boldsymbol{y}\#}$	T
(17,5,2)	$\mathbf{6}^{354}$	*20*
(18,5,2)	$\mathbf{6}^{11508}$	*178*
(19,5,2)	$\mathbf{6}^{11790}$	*152*
(18,6,2)	$\mathbf{6}^{42041}$	*474*
(19,6,2)	$\mathbf{6}^{49460}$	*500*
(19,7,2)	$\mathbf{6}^{49622}$	*475*
(18,6,3)	$\mathbf{6}^{60080}$	*570*
(19,6,3)	$\mathbf{6}^{83435}$	*736*
(19,7,3)	$\mathbf{6}^{87877}$	*796*
(19,7,4)	$\mathbf{6}^{88409}$	*879*