Computational Methods for Classification of Codes

Janne I. Kokkala
Computational Methods for Classification of Codes

Janne I. Kokkala

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Electrical Engineering, at a public examination held at the lecture hall E (Otakaari 1, Espoo) of the school on 29 September 2017 at 12.

Aalto University
School of Electrical Engineering
Department of Communications and Networking
Information Theory
Supervising professor
Prof. Patric Östergård, Aalto University, Finland

Preliminary examiners
Prof. Ian Wanless, Monash University, Australia
Prof. Masaaki Harada, Tohoku University, Japan

Opponent
Dr. Markus Grasl, Max Planck Institute for the Science of Light, Germany
Abstract

One of the fundamental problems in digital communications and data storage systems is finding good coding methods that allow transmitting or storing information efficiently in situations where errors may occur. Many problems in coding theory can be described in terms of combinatorial objects. These problems can then be solved using theoretical and computational methods.

The focus of this thesis is developing algorithms for classification and existence problems involving codes and covering arrays, which are mathematically related to codes and used for example in designing tests for systems such as software. The algorithms make use of common techniques for exhaustive generation and isomorph rejection.

The new methods are applied to three problems. First, all maximum distance separable (MDS) codes over alphabets of size at most 8 with minimum distance at least 3 are classified. Three new equivalence classes of perfect one-error-correcting 8-ary MDS codes are found. Second, some small covering arrays of strength 2 are classified to assist studying the structure of such arrays. These results also settle the size of an optimal covering array in some cases. Third, the chromatic number of the square of the 8-cube, a problem that has resisted a solution since the first attempts in early 1990s, is solved using a coding-theoretical approach and the method of prescribing symmetries.

Keywords algorithm, classification, MDS code, covering array, chromatic number
Tekijä
Janne I. Kokkala

Väitöskirjan nimi
Laskennallisia menetelmiä koodien luokittelun

Julkaisija
Sähköteknikan korkeakoulu

Yksikkö
Tietoliikenne- ja tietoverkotekniikan laitos

Sarja
Aalto University publication series DOCTORAL DISSERTATIONS 159/2017

Tutkimusala
Informaatioteoria

Käsikirjoituksen pvm
12.06.2017

Väitöspäivä
29.09.2017

Julkaisuluvan myöntämispäivä
08.08.2017

Kieli
Englanti

Monografia
☑ Artikkeliväittökirja
☐ Esseeväittökirja

Tiivistelmä

Digitaalisessa viestinnässä ja tiedontalennusjärjestelmissä keskeinen ongelmia on löytää hyviä koodausmenetelmiä, joilla voidaan siirtää tai tallentaa tietoa tehokkaasti tilanteissa, joissa voi esiintyä virheitä. Monet koodaustietorion ongelmat voidaan esittää kombinatoristen objektien avulla, ja näitä ongelmia voidaan ratkoo teoreettisilla ja laskennallisilla menetelmillä.


Avainsanat
algoritmi, luokittelu, MDS-koodi, peittotaulukko, kromaattinen luku

ISBN (painettu) 978-952-60-7581-5

ISBN (pdf) 978-952-60-7580-8

ISSN-L 1799-4934

ISSN (painettu) 1799-4934

ISSN (pdf) 1799-4942

Julkaisupaikka Helsinki

Painopaikka Helsinki

Vuosi 2017

Sivumäärä 140

Preface

The work for this thesis was conducted at the Department of Communications and Networking, Aalto University. My research was supported by Aalto ELEC Doctoral school, Academy of Finland under grants no. 132122 and 289002, Nokia Foundation, and Emil Aaltonen Foundation.

I am grateful to my supervisor Patric Östergård for his guidance and encouragement throughout the years. I would also like to thank my other coauthors Denis Krotov, Karen Meagher, Reza Naserasr, Kari Nurmela, and Brett Stevens for their contribution, and the preliminary examiners Ian Wanless and Masaaki Harada for their feedback regarding this thesis.

Many other people who are too numerous to mention individually have helped me throughout my journey in all things academic, social, and otherwise. I want to thank the current and former members of the research group and other colleagues in the department for creating an enjoyable and inspiring environment. In addition, I wish to thank various doctoral students at Aalto University and my family and friends for their valuable support.

Espoo, August 25, 2017,

Janne I. Kokkala
Preface
Contents

Preface i

Contents iii

List of Publications v

Author's Contribution vii

List of Symbols ix

List of Abbreviations xi

1. Introduction 1
   1.1 Contribution ............................ 2
   1.2 Structure of this Overview .................... 2

2. Preliminaries 5
   2.1 Symmetries and Group Actions ............... 5
   2.2 Hamming Space .......................... 6
   2.3 Graphs ............................... 7

3. Codes and Related Structures 9
   3.1 Codes ................................ 9
   3.2 Perfect Codes ............................ 10
   3.3 MDS Codes ............................. 11
      3.3.1 Equivalent Structures ............... 13
   3.4 Covering Arrays .......................... 16
   3.5 Extensions of Codes and Covering Arrays ........ 17
4. Computational Classification 19
   4.1 Tools for Common Subproblems .................... 20
      4.1.1 Clique Problem .............................. 20
      4.1.2 Exact Cover .................................. 21
      4.1.3 Graph Isomorphism ............................ 21
   4.2 Generation by Subobjects ........................... 23
   4.3 Isomorph Rejection ................................ 26
      4.3.1 Constructing Objects of Certain Form .......... 26
      4.3.2 Canonical Augmentation ........................ 28
   4.4 Construction by Prescribed Symmetries .............. 29
   4.5 Validity of Computational Results ................... 31

5. Classification Results 33
   5.1 MDS Codes ......................................... 33
   5.2 Covering Arrays .................................... 35
   5.3 The Chromatic Number of the Square of the 8-Cube .. 37

6. Conclusions 39

References 41

Errata 51

Publications 53
List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.


Author’s Contribution

Publication I: “Classification of Graeco-Latin cubes”

The author designed and implemented the computational method and took part in writing the article.

Publication II: “On the classification of MDS codes”

The author designed and implemented the computational method and wrote parts of the article.

Publication III: “Further results on the classification of MDS codes”

The author designed the computational method together with the coauthor, implemented it, and wrote most the article with help from coauthors.

Publication IV: “Bounds, structure, and classification of small strength-2 covering arrays”

The author designed the computational method together with P. R. J. Östergård, implemented it, and wrote the part of the article describing the computational work with help from coauthors.
Publication V: “The chromatic number of the square of the 8-cube”

The general idea of searching by symmetries was given by the coauthor. The author designed the details of the computational method, implemented it, and wrote the article together with the coauthor.
List of Symbols

$[n]$ the set $\{1, 2, \ldots, n\}$

$(n, M, d)_q$ a $q$-ary code with length $n$, size $M$, and minimum distance at least $d$

$(n, k)_q$ a $q$-ary maximum distance separable code of length $n$ and combinatorial dimension $k$

$\mathcal{A}$ a finite alphabet of symbols

$A(n, d)$ the maximum size of a binary code of length $n$ and minimum distance $d$

$\text{Aut}(X)$ the automorphism group of $X$

$C(X)$ the set of children of $X$

$\text{CA}(N; t, k, v)$ a covering array of size $N$, strength $t$, degree $k$, order $v$, and index 1

$\text{CAN}(t, k, v)$ the covering array number for strength $t$, degree $k$, order $v$, and index 1

$d_H(x, y)$ the Hamming distance between $x$ and $y$

$\mathbb{F}_q$ the finite field of order $q$

$\text{Iso}(X, Y)$ the set of isomorphisms from $X$ to $Y$

$\mathcal{P}(X)$ the power set of $X$

$Q_n$ the $n$-dimensional hypercube graph

$Q_n^k$ the $k$th power of $Q_n$

$\text{UCA}(N; t, k, v)$ a uniform covering array of size $N$, strength $t$, degree $k$, order $v$, and index 1

$\chi(\Gamma)$ the chromatic number of the graph $\Gamma$
List of Symbols
List of Abbreviations

CA   covering array
CAN  covering array number
MDS  maximum distance separable
MOLS mutually orthogonal Latin squares
OA   orthogonal array
UCA  uniform covering array
List of Abbreviations
1. Introduction

One of the fundamental problems in digital communications and data storage systems is finding good coding methods that allow transmitting or storing information efficiently in situations where errors may occur [105]. Combinatorial coding theory approaches this problem by describing a code as a finite set of codewords that consist of symbols of a finite alphabet [59]. The main problem then is the mathematical problem of finding codes that have certain combinatorial properties. In particular, one often seeks to find codes that have a large number of codewords, which relates to transmission efficiency, and a large distance between codewords, which relates to the error-correcting capability of the code.

As is often the case in mathematics, certain codes are equivalent or related to various other structures in combinatorics. For example, maximum distance separable (MDS) codes, which are studied in this thesis, are equivalent to orthogonal arrays and sets of mutually orthogonal Latin squares, which are commonly used in designing statistical experiments [44, 73]. Moreover, covering arrays, which are used in testing systems such as software [48, 75], are a generalization of orthogonal arrays and thus closely related to MDS codes.

With the increase of computing power, computational methods [42] have become common in the study of existence and classification of combinatorial structures. Among the most famous theorems proved using a computer are the four-color theorem [6, 7] and the nonexistence of projective planes of order 10 [53]; there still are no known proofs of these theorems that do not require a computer to check.

The general motivation for doing computational work to solve single cases of combinatorial problems is twofold. In some cases, previously un-
Introduction

known objects that are found computationally may be directly useful in applications. On the other hand, the results of a computer search, such as a full catalog of certain objects, can be studied to assist theoretical development.

1.1 Contribution

In this work we develop computational methods for three related problems: classification of MDS codes over small alphabets, especially perfect MDS codes, classification of small covering arrays, and a graph coloring problem that has applications in network theory and is mathematically related to binary codes.

In Publications I, II, and III, we complete the classification of MDS codes over alphabets of size at most 8 and minimum distance at least 3. The work is done in parts, where smaller MDS codes are classified first and the results are then used in classification of larger codes. Each step requires a different computational method due to the differences in the sizes and numbers of the objects. The most significant result is the construction of three new perfect one-error-correcting 8-ary MDS codes.

In Publication IV, we develop a similar method for covering arrays and classify small covering arrays. The goal is to improve bounds for the optimal sizes of covering arrays and create a catalog of small covering arrays to assist studying the structure of covering arrays.

Finally, in Publication V, we determine the chromatic number of the square of the 8-cube, which had been an open problem despite many efforts since the early 1990s [108]. The problem is solved by approaching it from a coding-theoretical point of view and constructing a coloring of the graph using classification results of binary codes and the method of prescribing symmetries.

1.2 Structure of this Overview

This overview is structured as follows. In Chapter 2, we review some basic mathematical concepts. In Chapter 3, we describe codes and related structures that are studied in this thesis, and in Chapter 4, we give an
overview of the computational methods that are relevant to this work. In Chapter 5, we present the classification results of this work and discuss how they relate to earlier research. We conclude this overview by discussing the contribution of this thesis and possible future work in Chapter 6.
2. Preliminaries

In this chapter, we briefly introduce some common mathematical concepts used in this thesis.

2.1 Symmetries and Group Actions

In combinatorics, it is common to say that two objects are isomorphic if one can be obtained from the other by some simple operation. The exact definition of isomorphism depends on the problem. The usual requirement is that isomorphic objects share some important properties. For some objects there may be various notions of isomorphism that come for different applications; in that case other terms for isomorphism are used as well. For example, the term equivalence is used for unrestricted codes and will be introduced in Section 3.1.

The operations that maintain isomorphism of objects, called isomorphisms, form a group that acts on the set of objects that are considered. Here we present the basic properties of group actions that are used in this thesis to describe symmetry of combinatorial objects; for a more formal introduction to group theory and group actions, see for example [88, Chapter 3].

Definition 1. Let $G$ be a group and $S$ a set. A function $\phi : G \times S \to S$ is a group action of $G$ on $S$ if $\phi(e, x) = x$ for the identity element $e \in G$ and all $x \in S$, and $\phi(gh, x) = \phi(g, \phi(h, x))$ for all $g, h \in G$ and all $x \in S$.

Often the group action $\phi$ is clear from the context and we use the notation $gx = \phi(g, x)$ for all $g \in G$ and $x \in S$. A group action of $G$ on $S$ also
Preliminaries

induces a group action of $G$ on $\mathcal{P}(S)$: For all $g \in G$ and $X \in \mathcal{P}(S)$, we have

$$gX = \{gx : x \in X\}.$$  \hspace{1cm} (2.1)

For a subgroup $H \leq G$ and an element $x \in S$, the orbit of $x$ under $H$ is the set

$$Hx = \{hx : h \in H\},$$

and for an element $x \in S$, the stabilizer of $x$ in $G$ is the subgroup

$$G_x = \{g \in G : gx = x\}.$$

The orbit–stabilizer theorem [88, Theorem 3.19] states that

$$|Gx| = |G|/|G_x|.$$  

In the following, $\Omega$ is a set of combinatorial objects, and $G$ is the group of isomorphisms acting on $\Omega$. When the objects are subsets of some set $U$, that is, $\Omega \subseteq \mathcal{P}(U)$, the isomorphisms can be defined as operations on the elements of $U$, which gives a group action on $U$. The action on $\Omega$ is then given by (2.1).

For objects $X$ and $Y$, we denote by $\text{Iso}(X,Y)$ the set of all isomorphisms that map $X$ to $Y$, that is,

$$\text{Iso}(X,Y) = \{g \in G : gX = Y\}.$$  

An isomorphism that maps an object $X$ onto itself is called an automorphism of $X$. The stabilizer of $X$ in $G$ is the set $\text{Iso}(X,X)$, and is called the automorphism group of $X$, also denoted by $\text{Aut}(X)$. A subgroup $H \leq \text{Aut}(X)$ is called a group of automorphisms of $X$. For an object $X$, the set of objects isomorphic to $X$ is the orbit of $X$ under $G$ and is called the equivalence class of $X$. By the orbit–stabilizer theorem, the size of the equivalence class of $X$ is $|G|/|\text{Aut}(X)|$.

2.2 Hamming Space

For a finite set $A$ and a positive integer $n$, the set of $n$-tuples of elements of $A$ is denoted by $A^n$. An element $x \in A^n$ can be written elementwise as $x = (x_1, x_2, \ldots, x_n)$. In this thesis, the indices $1, \ldots, n$ are called coordinates and the elements of $A$ are called symbols. The Hamming distance
Preliminaries

between two elements $x$ and $y$ of $A^n$, denoted by $d_H(x, y)$, is the number of positions in which they differ, that is,

$$d_H(x, y) = |\{i : x_i \neq y_i\}|. \tag{2.2}$$

The set $A_n$ coupled with the Hamming distance $d_H$ forms a metric space called the Hamming space. A bijection $f : A^n \to A^n$ that maintains the Hamming distance between any pair of elements, that is, for all $x, y \in A^n$,

$$d_H(x, y) = d_H(f(x), f(y)),$$

is called an isometry of the Hamming space. The isometries form a group that acts on $A^n$ and is called the isometry group of the Hamming space. All isometries of the Hamming space can be constructed as permutations of coordinates and permutations of symbols at each coordinate separately [42, Section 3.2.1], that is, all isometries are of the form

$$(x_1, x_2, \ldots, x_n) \mapsto (\sigma_1(x_{\pi^{-1}(1)}), \sigma_2(x_{\pi^{-1}(2)}), \ldots, \sigma_n(x_{\pi^{-1}(n)}))$$

for a permutation $\pi$ of $[n]$ and permutations $\sigma_i$ of $A$.

2.3 Graphs

A graph is a pair $\Gamma = (V, E)$ where $V$ is a set of vertices and

$$E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$$

is a set of edges. Two vertices $u, v \in V$ of $\Gamma$ are called adjacent if the graph contains an edge $\{u, v\}$. A clique in $\Gamma$ is a set $V' \subseteq V$ such that each pair of vertices in $V'$ is adjacent.

A coloring of a graph $\Gamma = (V, E)$ is a function $c : V \to [k]$ for some $k$; the values of $c$ are called colors. A coloring is called proper if $c(u) \neq c(v)$ for all pairs of adjacent vertices $u$ and $v$. The chromatic number of a graph $\Gamma$, denoted by $\chi(\Gamma)$, is the smallest number of colors for which a proper coloring exists. For a graph $(V, E)$ and its coloring $c$, the triple $(V, E, c)$ is called a colored graph.

Two graphs, $\Gamma_1 = (V_1, E_1)$ and $\Gamma_2 = (V_2, E_2)$, are isomorphic if there is a bijective map from $V_1$ to $V_2$ that maps every adjacent pair of vertices into an adjacent pair and every non-adjacent pair into a non-adjacent pair.
Preliminaries

definition can also be generalized for colored graphs by requiring that the map maintain the colors of the vertices.

The distance between vertices $u$ and $v$ is the smallest $d$ for which there exists a tuple $(v_0, v_1, \ldots, v_d)$ of vertices with $u = v_0$ and $v = v_d$ such that for all $i \in [d]$, the vertices $v_{i-1}$ and $v_i$ are adjacent. If such a tuple does not exist, the distance is undefined.

The $k$th power of a graph $\Gamma$, denoted by $\Gamma^k$, is the graph that has edges between the pairs of vertices whose distance in $\Gamma$ is at most $k$. In particular, the 2nd power of a graph $\Gamma$ is called the square of the graph.

To conclude this section, we present a family of graphs that is related to Hamming spaces and binary codes and is studied in Publication V. The $n$-dimensional hypercube, or $n$-cube, denoted by $Q_n$, is the graph with the vertex set $\{0, 1\}^n$ that has an edge between a pair of vertices if their Hamming distance is exactly 1. The $k$th power of the $n$-cube, $Q_n^k$, thus has an edge between a pair of vertices if their Hamming distance is at most $k$. 
3. Codes and Related Structures

3.1 Codes

In combinatorics, a code $C$ of length $n$ and size $M$ over an alphabet $A$ is a subset of $A^n$ of size $M$. The elements of $C$ are called codewords, and the elements of $A^n$ are called words. A code over an alphabet of size $q$ is called a $q$-ary code. An important parameter of a code is its minimum distance, the smallest Hamming distance between any two different codewords. A $q$-ary code with length $n$, size $M$, and minimum distance at least $d$ is called an $(n, M, d)_q$ code.

The minimum distance is an important measure of the quality of the code. A code with minimum distance at least $d$ can detect $d - 1$ errors: If at most $d - 1$ elements of a codeword change during transmission, one can detect that an error has occurred. A code with minimum distance at least $2t + 1$ can correct $t$ errors: If at most $t$ elements of a codeword change during transmission, the original codeword can be restored because there is a unique codeword that has Hamming distance at most $t$ to the received codeword.

If $q$ is a prime power and the alphabet is the finite field $\mathbb{F}_q$, a code $C$ of length $n$ is called linear if it is a subspace of the vector space $\mathbb{F}_q^n$. A linear code is usually described in terms of a $k \times n$ generator matrix whose rows form a basis of the corresponding vector space, or an $(n-k) \times n$ parity check matrix $H$ for which the code is the set of codewords $c$ for which $cH^T = 0$.

A code that is not linear is called nonlinear. In cases where both linear and nonlinear codes are discussed, the codes are called unrestricted. The codes in this thesis are unrestricted unless otherwise mentioned.
In the unrestricted case, we often treat the codes as subsets of the Hamming space \((A^n, d_H)\). If there is an isometry of the Hamming space that maps a code \(C\) to another code \(C'\), the codes are called equivalent. As explained in Section 2.2, such an isometry is given by a permutation of coordinates followed by separate permutations of symbols at each coordinate. If there is only one \((n, M, d)_q\) code up to equivalence for given parameters, the code is said to be unique. There also exist other notions of isomorphism for different cases. For example, for linear codes, it is required that an isomorphism maintain linearity of codes.

Codes can be constructed from other codes in many different ways. Let \(C\) be an \((n, M, d)_q\) code. Shortening \(C\) means removing a coordinate and retaining all codewords that have a given symbol at that coordinate. Puncturing \(C\) means removing a coordinate and retaining all codewords; note that the number of codewords may decrease if two codewords of \(C\) differ only in that coordinate. Extending \(C\) means adding a coordinate. The new symbol in each codeword is not restricted in this definition of extending, but often extending is used to create a new code with larger minimum distance. Augmenting \(C\) means adding new codewords. Lengthening \(C\) means extending it so that the new symbol is the same in each codeword and then augmenting it.

A central problem in coding theory is maximizing the number of codewords \(M\) given the other parameters \(n, d, q\). For binary codes, that is, \(q = 2\), the maximum size of a code with length \(n\) and minimum distance \(d\) is denoted by \(A(n, d)\). Brouwer maintains a list of best currently known bounds for \(A(n, d)\) for small parameters [16].

### 3.2 Perfect Codes

Let \(C\) be an \((n, M, 2t + 1)_q\) code, that is, a \(t\)-error-correcting code. For each codeword, there are exactly \(\sum_{k=0}^{t} \binom{n}{k} (q-1)^k\) words in \(A^n\) that have Hamming distance at most \(t\) to that codeword. The fact that a word in \(A^n\) can be within Hamming distance \(t\) of at most one codeword yields the Hamming bound, which was first introduced by Hamming [32] for binary codes,

\[
M \leq \frac{q^n}{\sum_{k=0}^{t} \binom{n}{k} (q-1)^k}.
\]
Codes and Related Structures

Codes that are \( t \)-error-correcting and attain this bound are called \textit{perfect} or \textit{\( t \)-perfect}. The trivial perfect codes are codes containing the whole space, codes with only one codeword, and odd-length binary codes equivalent to the code that contains the all-zero and the all-one codeword. For linear codes, \( q \) is a prime power and all perfect codes are known [99]. A nontrivial linear perfect one-error-correcting code must have the parameters

\[
\left( \frac{q^m - q}{q - 1}, \frac{q^m - q}{(q - 1)^2}, 3 \right)_q
\]

for \( m \geq 2 \). Such codes exist for all \( m \geq 2 \) and prime powers \( q \) and are called Hamming codes. The parity check matrix of a Hamming code is the \( m \times \left( \frac{q^m - q}{q - 1} \right) \) matrix whose columns are all \( m \)-tuples over \( \mathbb{F}_q \) that have 1 as first nonzero element [59, Section 7.3]. Other linear perfect codes are the \((23, 2^{12}, 7)_2\) and \((11, 3^6, 5)_3\) Golay codes [30]. All nontrivial linear perfect codes are equivalent to one of these codes.

For the unrestricted case, the problem remains open. When \( q \) is a prime power, there are no perfect codes with other parameters than those for linear codes [99], but there exist perfect codes with the same parameters as the linear codes that are not equivalent to the linear codes; see for example [81] for computational classification of the \((15, 2^{11}, 3)_2\) codes. However, no nontrivial perfect codes over non prime power alphabets are known, and the existence is an open problem in general; see [34] for an overview.

### 3.3 MDS Codes

Another general upper bound for the size of an \((n, M, d)_q\) code is the Singleton bound,

\[
M \leq q^{n-d+1}.
\]

This can be proved by observing that no two codewords may agree in the \( n - d + 1 \) first coordinates and there are only \( q^{n-d+1} \) possible tuples of length \( n - d + 1 \).

Codes attaining this bound are called \textit{maximum distance separable} (MDS). MDS codes were first studied explicitly by Singleton in 1964 [93]. A well-known example of MDS codes is the family of Reed-Solomon codes [86], which are linear MDS codes and are used in many applications, including DVDs [38], space transmission [106, Chapter 3], and DSL technologies [31].
It is common to characterize an MDS code by the length $n$ and the value $k = n - d + 1$. An $(n, q^k, n - k + 1)_q$ code is called an $(n, k)_q$ MDS code. If the code is linear, $k$ is the dimension of the vector space corresponding to the code, but the parameter $k$ is convenient also for unrestricted codes and is then called the \textit{combinatorial dimension}.

The central property of MDS codes is that given any $k$ coordinates, all $k$-tuples of symbols over the alphabet occur in those coordinates in exactly one codeword; moreover, all codes with this property are MDS codes. From this it is easy to see that puncturing an $(n, k)_q$ MDS code with $n > k$ gives an $(n - 1, k)_q$ MDS code and shortening an $(n, k)_q$ MDS code with $n \geq k > 1$ gives an $(n - 1, k - 1)_q$ MDS code. Some $(n, k)_q$ MDS codes can also be extended to $(n + 1, k)_q$ MDS codes; in this case the codes are called \textit{extendable}.

Since the introduction of MDS codes, most research has focused on linear codes. Nevertheless, the unrestricted case has also received some attention, in part because of the equivalence of unrestricted MDS codes to various other combinatorial structures. Below we discuss some results about linear and unrestricted MDS codes.

MDS codes with $k = n$ or $k = 1$ are called trivial. MDS codes with $n = k + 1$, which correspond to Latin hypercubes, exist for all $k$ and $q$. However, in general the existence of MDS codes with given parameters is an open problem. For prime powers $q$, the MDS conjecture, first given for linear MDS codes by Segre in 1955 [90] in terms of $n$-arcs in finite geometries, is commonly believed to be true.

\textbf{Conjecture 1 (MDS conjecture).} Let $q$ be a prime power. Nontrivial $(n, k)_q$ MDS codes exist if and only if $n = k + 1$ or $n \leq q + 1$, with the exception that when $q$ is a power of 2, $(q + 2, 3)_q$ and $(q + 2, q - 1)_q$ MDS codes exist.

Linear codes with these parameters are known to exist [59, Section 11.5]. In 2012, the MDS conjecture was proved for linear codes by Ball [8] when $q$ is a prime, and by Ball and De Beule [9] when $q$ is a power of a prime $p$ and $k \leq 2p - 2$. There are also other cases for which the conjecture has been proved for linear codes; see [35] for an overview.

When $q$ is not a prime power, the existence of $(n, k)_q$ MDS codes for $n \leq q + 1$ is not true in general. A construction by Tolhuizen [100] implies that if an $(n, k)_q$ and an $(n, k)_r$ MDS code exist, then an $(n, k)_{qr}$ MDS code
exists; this gives an infinite family of MDS codes over non prime power alphabets. However, it is known for example that $(4, 2)_6$ and $(9, 2)_{10}$ MDS codes do not exist (See Section 3.3.1). For general $q$, a conjecture similar to the MDS conjecture can be formulated for the upper bound of the length of an MDS code with $n \geq k + 2$.

**Conjecture 2.** If a nontrivial $(n, k)_q$ MDS code exists, then $n = k + 1$ or $n \leq q + 1$, with the exception that when $q$ is a power of 2, $(q + 2, 3)_q$ and $(q + 2, q - 1)_q$ MDS codes exist.

Conjecture 2 has been proved for various sets of parameters $k$ and $q$. For $k \geq q$, it follows from the Hamming bound for 1-error-correcting codes, and for $k = 2$, it follows from the well-known upper bound for the size of a set of MOLS. For $q \leq 6$, the conjecture was proved by Maneri and Silverman [60, Theorem 4 (2)]. Bruen and Silverman [19] proved it in the case $k = 3$ when $q$ is not divisible by 4, and Alderson [2] in the case $k = 4$ when $q$ is even but not divisible by 36. See [37, Section 5] for a list of other sets of parameters for which Conjecture 2 has been proved.

In addition to these results, there has been some other research about unrestricted MDS codes. Tolhuizen [100] noted that some earlier results of MDS codes over finite field alphabets also hold when $q$ is not a prime power. Alderson showed in 2005 [2] that a $(q + k - 2, k)_q$ MDS code is extendable if and only if $q$ is even. Alderson showed in 2006 [3] that the $(6, 3)_4$ and $(5, 3)_4$ MDS codes are unique. Alderson, Bruen, and Silverman [4] showed in 2007 that certain long linear MDS codes cannot be extended to unrestricted MDS codes that are not equivalent to a linear code. Some new bounds for the length of an MDS code were derived independently by Yang, Zhang, and Wang in 2011 [107], and Alderson and Huntemann in 2014 [5], using the concept of partition weight enumerator, introduced earlier for linear MDS codes [27].

### 3.3.1 Equivalent Structures

Unrestricted MDS codes are equivalent to many other structures that are studied in combinatorics, and many results that apply to MDS codes are published using other terminology. Here we introduce two of them; Latin squares and related structures, and orthogonal arrays.
A Latin square of order $q$ is a $q \times q$ array with elements from an alphabet $A$ of size $q$ such that in each row and each column, every symbol occurs exactly once. See [44] for a recently updated overview of Latin squares and various applications of them. A Latin rectangle is a $r \times q$ array with $r \leq q$ with elements from an alphabet of size $q$ such that every symbol occurs exactly once in every row and at most once in every column. A pair of Latin squares of the same order is called orthogonal if each pair of symbols occurs exactly once when the squares are superimposed. A set of Latin squares where each pair of squares is orthogonal is called a set of mutually orthogonal Latin squares (MOLS). A set of two MOLS is also called a Graeco–Latin square. If Latin squares $L$ and $M$ are orthogonal, $M$ is called an orthogonal mate of $L$.

Sets of MOLS have been widely studied; see for example [1] for a general overview. It is known that a set of MOLS of order $q$ can contain at most $q-1$ Latin squares, and that bound can be reached when $q$ is a prime power. Euler conjectured in the 18th century that Graeco–Latin squares do not exist when $q \equiv 2 \pmod{4}$ [28]. For order 2 this is trivial, and for order 6 it was proved by Tarry in 1900 [97, 98], but for all other orders a construction of a Graeco–Latin square was found in 1960 [14]. However, the maximum size of a set of MOLS for orders that are not prime powers is an open problem in general. Shrikhande [91] showed that a set of $q-3$ MOLS of order $q$ exists if and only if a set of $q-1$ MOLS of order $q$ exists. For order 10, a set of 9 MOLS does not exist, which follows from the nonexistence of projective planes of order 10, proved using a computer search [53]. From these it follows that 7 MOLS of order 10 do not exist. On the other hand, the largest known set of MOLS of order 10 has only 2 squares, which leaves the existence of a set of $r$ MOLS of order 10 for $3 \leq r \leq 6$ open.

There exist various generalizations of Latin squares to more than two dimensions. In this work, we use the one that has direct relation to MDS codes [18, 25, 55]. A Latin hypercube of order $q$ and dimension $k$ is a $k$-dimensional $q \times q \times \cdots \times q$ array with elements from an alphabet $A$ of size $q$ such that if any $k-1$ coordinates are fixed, every symbol occurs exactly once.

There is a one-to-one correspondence between ordered sets of $r$ MOLS of order $q$ (with $r \geq 1$) and $(r + 2, 2)_q$ MDS codes: When indexing the
coordinates of the squares with the symbols of the alphabet $A$, for a set of MOLS $(L_1, L_2, \ldots, L_r)$ the set

$$C = \{(x, y, L_1(x, y), L_2(x, y), \ldots, L_r(x, y)) : x, y \in A\} \quad (3.3)$$

is an $(r + 2, 2)_q$ MDS code, and conversely, for an $(r + 2, 2)_q$ MDS code $C$, there is a unique ordered set of MOLS $(L_1, L_2, \ldots, L_r)$ for which (3.3) holds. Similarly, Latin hypercubes of order $q$ and dimension $k$ correspond to $(k + 1, k)_q$ MDS codes: For a Latin hypercube $L$, the corresponding code is

$$C = \{ (x_1, x_2, \ldots, x_k, L(x_1, x_2, \ldots, x_k)) : x \in A^k \}. \quad (3.4)$$

In [92] and Publication I, a pair of 3-dimensional Latin hypercubes is called a Graeco–Latin cube if, when the cubes are superimposed, each 2-dimensional subarray obtained by fixing any coordinate is a Graeco–Latin square. A Graeco–Latin cube of order $q$ corresponds to an $(5, 3)_q$ MDS code.

There exist various notions of equivalence for Latin squares, Latin hypercubes, and MOLS. Of interest in this work is the most general one, which corresponds to the notion of equivalence for MDS codes: Two Latin hypercubes or two sets of MOLS are called paratopic if the corresponding MDS codes are equivalent. Counting and classifying Latin squares for small orders has a long history including many published errors; see [63] and references therein. More recent results include counting the number Latin squares of order 11 by McKay and Wanless [68], and counting the number of equivalence classes of Latin squares under different notions of equivalence by McKay, Meynert, and Myrvold [63] for orders 9 and 10, and by Hulpke, Kaski, and Östergård [36] for order 11.

Sets of MOLS for orders up to 9 have been classified up to different notions of equivalence by Egan and Wanless [26]. Small Latin hypercubes have been classified by McKay and Wanless [69]. Potapov and Krotov studied asymptotics and bounds for the number of Latin hypercubes [84].

An orthogonal array of degree $k$, order $v$, strength $t$, size $N$, and index $\lambda = N/v^t$, denoted by $OA_\lambda(t, k, v)$ or $OA(N, k, v, t)$, is an $N \times k$ array over an alphabet of size $v$ where in each $N \times t$ subarray, each $t$-tuple of symbols occurs exactly $\lambda$ times. An $OA_1(t, k, v)$ is equivalent to an $(k, t)_v$ MDS code. For a general overview of orthogonal arrays, see [33]. Orthogonal arrays are used for example in software testing [85, Section 18.6.4]. However,
orthogonal arrays do not exist for all parameters $\lambda$, $t$, $k$, and $v$. In the next section, we introduce a useful generalization that does not have this limitation.

### 3.4 Covering Arrays

A *covering array* of degree $k$, order $v$, strength $t$, size $N$, and index $\lambda$, denoted by $\text{CA}_\lambda(N; t, k, v)$ is an $N \times k$ array over an alphabet of size $v$ where in each $N \times t$ subarray, each $t$-tuple of symbols occurs at least $\lambda$ times [22]. Covering arrays differ from orthogonal arrays in that they allow a $t$-tuple to occur more than $\lambda$ times, and unlike orthogonal arrays, a covering array exists for every $t$, $k$, $v$, and $\lambda$. The index $\lambda$ is often 1, in which case the parameter is omitted in the notation. Similarly to codes, two covering arrays are called *equivalent* if one can be obtained from another by a permutation of rows and columns followed by a permutation of symbols in each column separately.

Covering arrays are used in testing interactions of various parameters in systems such as software [48, 75]. Each column corresponds to one parameter, a symbol in that column corresponds to a value of that parameter, and each row corresponds to a test run. Then for all sets of $t$ parameters, all possible value combinations are tested in at least one run. To minimize resource requirements, the target is often to minimize the size $N$ of the covering array given the parameters $t$, $k$, and $v$.

The smallest $N$ for which a $\text{CA}(N; t, k, v)$ exists is denoted by $\text{CAN}(t, k, v)$, called the *covering array number* for the parameters $t$, $k$, and $v$. Covering arrays that attain this bound are called *optimal*. There have been various efforts to find bounds and exact values for $\text{CAN}(t, k, v)$ for small parameters. The results include various theoretical constructions [20, 21, 72], non-constructive existence proofs [89], stochastic searches [77, 103], and exhaustive searches [70, 102]. Colbourn maintains a list [23] of currently best known upper bounds for $\text{CAN}(t, k, v)$ that are realized by explicit constructions of covering arrays.

A $\text{CA}(N; t, k, v)$ is *uniform* if in every column, each symbol occurs $\lceil N/v \rceil$ or $\lfloor N/v \rfloor$ times. A uniform covering array is denoted by $\text{UCA}(N; t, k, v)$.
Observations of known covering arrays lead Meagher and Stevens to the following conjecture.

**Conjecture 3** ([71, Conjecture 1]). *If a CA(N;2,k,v) exist, there also exists a UCA(N;2,k,v).*

### 3.5 Extensions of Codes and Covering Arrays

Here we briefly discuss an observation that is central to the computational approach in Publications III and IV and in various studies of MOLS. Theorems 1 and 2 are converses of each other.

**Theorem 1.** Let \( C \) be an \((n,M,d)q\) code and let \( C' \) be an \((n+1,M,d+1)q\) code that is obtained by extending \( C \). For each \( v \in A \), let \( C'_v \) be the code obtained from \( C' \) by removing the last symbol and retaining the codewords that have symbol \( v \) in that coordinate. Then the codes \( C'_v \) are \((n,M_v,d+1)q\) codes for some \( M_v \) and form a partition of \( C \).

**Theorem 2.** Let \( C \) be an \((n,M,d)q\) code and let \( C'_v, v \in A, \) be \((n,M_v,d+1)q\) codes for some \( M_v \) that form a partition of \( C \). Let \( C' = \bigcup_{v \in A} C'_v v \), where \( C'_v v \) denotes the code obtained by adding a symbol \( v \) at the end of each codeword of \( C'_v \). Then \( C' \) is an \((n+1,M,d+1)q\) code.

Setting \( M = q^k \) for \( k = n - d + 1 \) in these theorems implies that extending an \((n,k)q\) code \( C \) into an \((n+1,k)q\) MDS code is equivalent to partitioning \( C \) into \((n,k-1)q\) MDS codes, up to a permutation of symbols in the last coordinate.

For \( n = 3, k = 2 \), this is equivalent to the common method of finding orthogonal mates for a given Latin square, used in computer search already in 1963 by Parker [82]. A *transversal* of a Latin square of order \( q \) is a set of \( q \) cells, one from each row and each column, that contains every symbol exactly once. A transversal then corresponds to an \((3,1)q\) MDS code. Consider mutually orthogonal Latin squares \( L \) and \( M \). For a given symbol, the cells that contain that symbol in the square \( M \) form a transversal of \( L \). An orthogonal mate \( M \) of \( L \) thus corresponds to a partition of the set of cells of \( L \) into transversals.

For covering arrays, Theorems 3 and 4 give a similar result which shows that extending a \( \text{CA}(N;t,k,v) \) named \( C \) to a \( \text{CA}(N;t,k+1,v) \) by adding
a column is equivalent to partitioning the multiset of the rows of $C$ into covering arrays of strength $t - 1$, up to permutation of rows and a permutation of symbols in the new coordinate.

**Theorem 3.** Let $C$ be a $\text{CA}(N; t, k, v)$ and let $C'$ be a $\text{CA}(N; t, k + 1, v)$ that is obtained by adding a column to $C$. For each $v \in A$, let $C'_v$ be the covering array obtained from $C'$ by removing the last column and retaining the rows that have symbol $v$ at that column. Then each $C'_v$ is a $\text{CA}(N_v; t - 1, k, v)$ for some $N_v$ and the multisets of rows of $C'_v$ form a partition of the multiset of rows of $C$.

**Theorem 4.** Let $C$ be a $\text{CA}(N; t, k, v)$ and let $C'_v$ for each $v \in A$ be a $\text{CA}(N_v; t - 1, k, v)$ for some $N_v$ such that the multisets of rows of $C'_v$ form a partition of the multiset of rows of $C$. Let $C'$ be the covering array obtained by taking every row from every $C'_v$ with the symbol $v$ added at the end. Then $C'$ is a $\text{CA}(N; t, k + 1, v)$.
4. Computational Classification

Two common problems in combinatorics are determining whether an object with certain properties exists and classifying all such objects up to isomorphism. There are two main ways to approach these problems computationally. One is to attempt to settle the existence problem by finding at least one object, and the other is to generate all such objects up to isomorphism. It is often faster to try to construct at least one object using methods that use various heuristics to guide the search [47, Chapter 5]. However, most such methods can never conclude that an object does not exist, and they can rarely be applied when a full classification is desired. Therefore, exhaustive generation is often preferable whenever it is computationally feasible.

In this work, we focus mostly on full classification by exhaustive generation. Full classification is carried out in Publications I, II, and III for MDS codes over small alphabets and in Publication IV for small strength-2 covering arrays. The method used in Publication V lies between the two main approaches, as we solve an existence problem by restricting our search to objects that have certain symmetries and performing a full classification of those objects.

This chapter reviews computational classification methods that are relevant to this work. We present the general methods and discuss their application for the classification problems in this work. In computational work one can very often reduce subproblems to other known problems and use existing software; we begin by discussing that in Section 4.1. In Section 4.2, we present the general principle of constructing objects by considering their subobjects, and in Section 4.3, we discuss methods for speeding up the search by rejecting isomorphic objects. In Section 4.4, we
present the general approach for generating objects with nontrivial symmetries. Finally, in Section 4.5 we discuss methods for gaining confidence in the validity of computational results.

4.1 Tools for Common Subproblems

Many subproblems that occur in classification problems can be described in terms of well-known problems. In many cases, existing software for these problems can be used to solve the subproblems. In this section, we present three common problems and software libraries that are used in this work: Cliquer for finding cliques in graphs, libexact for solving instances of the exact cover problem, and nauty for detecting graph isomorphism. For a discussion of these and similar problems from a complexity theory point of view, see for example [29].

4.1.1 Clique Problem

The problem of finding cliques of given size in a graph is a well-known hard problem; the decision version of the problem is NP-complete. In this work, we use the library Cliquer [76] to find cliques in a graph.

An example of the problem occurring in combinatorial search is finding an \((n, M, d)_q\) code with fixed parameters. The problem is formulated as follows. The vertices in the graph are the elements of the Hamming space \(\mathcal{A}^n\). There is an edge between two vertices if the Hamming distance between the corresponding words is at least \(d\). In the graph, every clique of size \(M\) corresponds to an \((n, M, d)_q\) code and vice versa. Due to the large isometry group of the Hamming space, solving this problem to find all \((n, M, d)_q\) codes is often not feasible, but the formulation is useful for example if a subset \(C'\) of the code \(C\) is fixed; then the vertices of the graph are all words in \(\mathcal{A}^n\) that have a Hamming distance at least \(d\) to every codeword in \(C'\) and a clique in the graph corresponds to the vertices in \(C \setminus C'\). This approach is used in Publication I to finish the construction of a code after some codewords have been fixed. For other examples of reducing combinatorial problems to the problem of finding cliques, see [79].
4.1.2 Exact Cover

Another common problem is the exact cover problem: Given a finite set $X$ and a family $\mathcal{F}$ of subsets of $X$, find all subsets of $\mathcal{F}$ that are partitions of $X$. The decision version of this problem is NP-complete. We use the library libexact [43] to solve instances of the exact cover problem. The library also works for the version of the problem where $X$ and $\mathcal{F}$ are multisets. In Section 3.5, we discuss the equivalence between finding partitions and extending MDS codes and covering arrays. That approach is used in Publication III for MDS codes: To extend and $(n,k)_q$ MDS code $C$ into an $(n+1,k)_q$ MDS code, we first find all subsets of $C$ that are $(n,k-1)_q$ MDS codes and then use libexact to find all partitions of $C$ into such subsets.

A similar approach with some modifications is used in Publication IV to generate covering arrays column by column. When extending a CA($N;2,k,v$), the exact cover formulation is as follows: $X$ is the multiset of rows of the array, and the elements of $\mathcal{F}$ are the subsets of $X$ that correspond to covering arrays of strength 1 (see Section 3.5). Because the set $\mathcal{F}$ is large, using libexact directly to find all subsets of $\mathcal{F}$ of size $v$ that are partitions of $X$ is computationally infeasible. Instead, we consider the significantly smaller set $\mathcal{F}'$ which contains all elements of $\mathcal{F}$ for which no proper subset is in $\mathcal{F}$, and use libexact to find all partitions of $X$ that are subsets of $\mathcal{F}' \cup \{\{x\} : x \in X\}$ and contain at least $v$ elements of $\mathcal{F}'$. For each such partition, we then add the elements from the singleton sets to the elements of $\mathcal{F}'$ in all possible ways. Because the number of singleton sets in the partitions is small, libexact still performs well in most cases.

4.1.3 Graph Isomorphism

The computational problem of detecting whether two graphs are isomorphic is not believed to be NP-complete but a polynomial time algorithm is not known either; see [66] for a recent overview of the graph isomorphism problem. The library nauty [65, 66] is used in this work. For a given colored graph $\Gamma = (V,E,c)$, nauty computes a canonical labeling, that is, a bijection $V \rightarrow [\vert V \vert]$ such that for two colored graphs $\Gamma_1 = (V_1,E_1,c_1)$ and $\Gamma_2 = (V_2,E_2,c_2)$, the graphs induced by the canonical labelings are equal if and only if $\Gamma_1$ and $\Gamma_2$ are isomorphic. The library can also output other information about a graph, such as the automorphism group.
Often the problem of detecting whether two combinatorial objects are isomorphic can be reduced to the graph isomorphism problem by representing the objects as colored graphs; see for example [47, Section 7.4.2] and [42, Section 3.3.2]. Here, we describe the standard reduction for unrestricted codes that is used in this work, introduced in [80] for binary codes.

For an \((n, M, d)\_q\) code \(C\), the corresponding graph \(\Gamma\) is of order \(nq + M\) and is as follows. First, \(\Gamma\) contains \(n\) copies of the complete graph of order \(q\), named \(\Gamma_1, \Gamma_2, \ldots, \Gamma_n\) and colored with the first color. The vertices of \(\Gamma_i\) for each \(i\) are labeled as \((i, a)\) for each symbol \(a\). Further, \(\Gamma\) contains \(M\) vertices colored with the second color, labeled by the codewords of \(C\). For each codeword \(c \in C\) and each coordinate \(i \in [n]\), \(\Gamma\) has an edge from \(c\) to \((i, c_i)\). Two codes, \(C\) and \(C'\), are now equivalent if and only if the corresponding graphs are isomorphic. Further, the automorphism group of a code \(C\) corresponds to the automorphism group of the graph. An example is illustrated in Figure 4.1 for the \((3,3,2)\_3\) codes \(C = \{000, 120, 211\}\) and \(C' = \{001, 120, 211\}\), which are equivalent. The code \(C'\) can be obtained from \(C\) by swapping the first two coordinates and then swapping the symbols 0 and 1 in the last coordinate. Similarly, the graph corresponding to \(C'\) can be obtained from the graph corresponding to \(C\) by swapping \(\Gamma_1\) and \(\Gamma_2\), then swapping the vertices labeled \((3,0)\) and \((3,1)\), and finally mapping the vertices labeled 000, 120, and 211 to the vertices labeled 001, 211, and 120, respectively.

In Publication IV, we treat rows of a covering array as codewords and use the same graph construction as for codes. However, because a covering array may have rows that occur multiple times, the automorphism groups of the graphs contain additional automorphisms that permute the vertices that correspond to rows that are equal. For the graphs that occur in Publication IV, this does not make a significant difference in the speed of \textit{nauty}, and the additional automorphisms can be removed from the \textit{nauty} output afterwards. Another option, which may be necessary for larger covering arrays with more duplicate rows, would be to have just one vertex for a set of equal rows that is colored with a new color corresponding to the multiplicity of the row.
4.2 Generation by Subobjects

Almost all algorithms that exhaustively generate combinatorial objects do it in a step-by-step manner by generating smaller objects and extending them to larger ones; see for example [42, Section 4.1] for a general discussion. Here, we discuss the general approach and its application to generating codes and covering arrays.

As an example, consider the generation of Latin squares of order $n$ in the following naive row-by-row manner. In the first step we consider all possible first rows, which are the permutations of the alphabet. In the next phase, we consider objects that are obtained by adding a second row to each of them, thus giving us $2 \times n$ Latin rectangles, and similarly in the $k$th phase we extend the $(k-1) \times n$ Latin rectangles to $k \times n$ Latin rectangles.

To discuss the general setting, we formalize this as follows. The objects that we wish to construct in the search are called full objects, other subobjects that are considered by the search are called partial objects. The search starts from a root object $X_0$, which is often a trivial object such as the empty set or a $0 \times n$ Latin rectangle.

A single step in the search takes as input a partial object $X$ and finds a set $C(X)$ of objects that can be obtained from $X$. Such a step may itself require using a nontrivial algorithm. In the example above, $X$ is a $k \times n$
Latin rectangle and $C(X)$ is the set of all $(k + 1) \times n$ Latin rectangles obtained by adding a row to $X$. The relationship between the objects can be illustrated in a search tree, a tree whose root is the root object, and the children of every node $X$ are the objects in $C(X)$. An example of a part of the search tree for Latin squares of order 3 is illustrated in Figure 4.2.

An object $X$ that we eventually wish to construct occurs in the search tree if there exists a tuple $(X_0, X_1, \ldots, X_n)$ where $X_0$ is the root of the tree, $X_n = X$, and for each $i$, we have $X_{i+1} \in C(X_i)$.

We assume that every nonroot object occurs at most once in the search tree. For all nonroot objects $X$ that occur in the search tree, the parent of $X$ is the unique $X'$ for which $X \in C(X')$.

A simple algorithm utilizing this idea is backtrack search: As an input the procedure gets an object $X$ and recursively calls itself for all objects in $C(X)$. Whenever the procedure finds a full object, that object is reported. Calling the procedure with $X_0$ as input generates all objects in the search tree. The performance of a backtrack search depends highly on the choice of the set of children. Often it is advisable to design the method so that $|C(X)|$ is minimized at each step; in the example of constructing Latin squares, one might at some steps extend a partial object by filling a column instead of adding a row. In addition, if it can be detected that the input cannot lead to a full object, the branch can be pruned from the search tree immediately. See [64] for other rules of thumb and an example utilizing them.

We will now discuss how the search is formulated in the publications in this work. In addition to what is described here, the methods utilize isomorph rejection, which is discussed in the next section.
The methods in Publications I and II generate \((k + 3, k + 1)_q\) MDS codes starting from \((k + 2, k)_q\) MDS codes; Publication I contains the case \(k = 2\), and Publication II contains the case \(k \geq 3\). In both methods, the first step adds the words that have a 0 in the first coordinate, which corresponds to a \((k + 2, k)_q\) MDS code, and the second step adds the remaining words that have a 0 in the second coordinate, which corresponds to another \((k + 2, k)_q\) MDS code. The advantage of this approach for the second step is that \(q^{k-1}\) of the \(q^k\) words of the \((k + 2, k)_q\) MDS code are already included in the code, which limits the number of children. The method in Publication I then proceeds by adding words that have a 0 in one of the remaining coordinates in each step and finally adding the rest of the words in the last step. In Publication II, on the other hand, the method proceeds by considering the shortened codes obtained by removing the third coordinate. This difference illustrates the importance of choosing the rule for extending partial objects: While the methods solve similar problems, the fastest approach to take is dictated by the specifics of the problem; using one approach for the other problem may yield larger search trees.

In Publication III, MDS codes are classified starting from \((k+2, k)_q\) MDS codes and extending them coordinate by coordinate. One search step then considers all the ways to obtain an \((n + 1, k)_q\) MDS code from an \((n, k)_q\) MDS code by adding a coordinate. The main computational obstacle for this problem is not the size of the search tree but conducting the single step of finding extensions of an \((n, k)_q\) MDS code \(C\), which correspond to partitions of \(C\) into \((n, k - 1)_q\) MDS codes (see Section 3.5). This problem is solved by using the results obtained in the classification of \((n - 1, k - 1)_q\) codes to find all \((n, k - 1)_q\) MDS codes that are subsets of \(C\) and then finding partitions of \(C\).

Similarly, in Publication IV we classify \(CA(N; 2, k, v)\)s starting from \(CA(N; 2, 2, v)\)s, and one step of the search extends covering arrays with \(k\) columns to covering arrays with \(k + 1\) columns by adding a column. For some \(N, v\), a full classification for all \(k\) would not be computationally feasible, and we perform a classification only for \(CA(N; 2, k, v)\)s for a fixed \(k\). We can reject all arrays with \(k' < k\) columns for which we can determine that they cannot be extended to a \(k\)-column array; this also decreases the time required by a single step, because it need not generate all extensions. More details of this method can be found in the publication.
4.3 Isomorph Rejection

Exhaustive generation of all objects of certain type is often computationally infeasible and not even desired. Instead, constructing one object from each equivalence class is enough. The computational method for detecting isomorphic objects depends on the problem, and for the case of codes and covering arrays, one method is discussed in Section 4.1.3. In this section, we discuss ways to perform isomorph rejection to cut the search tree and decrease the time requirement of the search.

The simplest way for isomorph rejection is to perform the full exhaustive search and at the end select only one object from each equivalence class of constructed objects. When using this method, the search tree has some branches for which all full objects in the branch are rejected. Therefore, the search can be speeded up by pruning those branches early. In the following, we discuss some ways to achieve this.

A straightforward way to prune the search tree is to perform isomorph rejection for partial objects. Often the search method is formulated in such a way that if two partial objects $X$ and $X'$ are isomorphic, every child of $X$ is isomorphic to a child of $X'$. This implies that the subtrees starting from isomorphic nodes contain isomorphic objects, and all except one of the subtrees can be pruned. In the simplest implementation, the search tree is traversed in a breadth-first manner, and at each level of the search tree, all objects are tested for isomorphism and only one object from each equivalence class is accepted for the next level of the search. Publication III uses this approach: Equivalence class representatives of $(n,k)_q$ MDS codes are extended into $(n+1,k)_q$ MDS codes in all possible ways, and then isomorph rejection is performed on all $(n+1,k)_q$ MDS codes obtained.

4.3.1 Constructing Objects of Certain Form

A more detailed way to implement isomorph rejection for partial objects is to define a canonical form for all objects. A canonical form of an object is a representative of the equivalence class such that each object in the equivalence class has the same canonical form. If the canonical form is defined in such a way that a child of an object $X$ can be in canonical form only if the object $X$ itself is in canonical form, then we may prune
non-canonical partial objects from the search tree. This method is called *orderly generation*. When using orderly generation, the search tree can be pruned even further if it can be detected that even though a partial object is in canonical form, it cannot lead to a full object in canonical form.

Often detecting canonical forms is a computationally tedious task. In these cases, one can find a good compromise by noticing that every equivalence class of full objects contains an object that satisfies certain conditions, but contrary to a canonical form, there may be more than one such object in the equivalence class. For example, a Latin square is always paratopic to a Latin square that has the first row and first column in lexicographic order; such a Latin square is said to be reduced. Figure 4.3 illustrates a search tree of reduced Latin squares of order 4. The rightmost two $2 \times 4$ arrays in the second level are paratopic (one can be obtained from the other by swapping the symbols 2 and 3 and the rightmost two columns), and thus yield paratopic full objects, but the approach still cuts most of the search tree while remaining computationally simple.

The approach of constructing only objects of a certain form is utilized in Publications I and II. In both publications, the construction of an $(n, k)_q$ MDS code $C$ starts by fixing some shortened codes of $C$ that are $(n - 1, k - 1)_q$ MDS codes and are constructed using an ordered list $\mathcal{L}$ of equivalence class representatives of $(n - 1, k - 1)_q$ MDS codes. One requirement for a code to be constructed is that the shortened code $C'$ that is fixed in the first step is an equivalence class representative that occurs in $\mathcal{L}$, and the shortened codes that are fixed in the following steps are
equivalent to $C'$ or a code that occurs in $\mathcal{L}$ after $C'$. The advantage of this approach is that in most branches of the search tree the algorithm need not consider the full list $\mathcal{L}$ when fixing new shortened codes. In this way the search tree can be pruned enough to make the algorithm run in feasible time for the problems that occur in Publications I and II. If orderly generation was used instead, the time gained by pruning the search tree a bit more would be small compared to the time required for performing a canonicity check after every step.

### 4.3.2 Canonical Augmentation

The methods described above require either detection of canonical forms or a full equivalence check of objects on some levels. These tasks may be time-consuming computationally. Canonical augmentation, introduced by McKay [61, 67], seeks to avoid these problems by requiring that a non-root object is obtained from its parent in a canonical way, rather than being of canonical form. We describe here the full version of canonical augmentation; there is also a simplified method, called weak canonical augmentation and described for example in [42, Section 4.2.3].

There are different ways to formulate canonical augmentation. Our presentation is similar to the one in [42, Section 4.2.3]. For the formal treatment, we require that the same group of isomorphisms act on objects on all levels of the search tree. For example when generating $k$-column covering arrays column by column, the smaller $k'$-column covering arrays with $k' < k$ can be considered as $k$-column covering arrays that have empty cells at the last $k - k'$ columns. However, in practice, the method can often be used without defining such a group action explicitly.

A necessary condition for being able to use canonical augmentation is that if two objects $X$ and $Y$ are isomorphic, then their children are isomorphic by an isomorphism between $X$ and $Y$, that is, for every $Z \in C(X)$, there is an isomorphism $g \in \text{Iso}(X, Y)$ for which $gZ \in C(Y)$.

For each nonroot object $X$, we define a **canonical parent** $m(X)$ such that if two nonroot objects $X$ and $Y$ are isomorphic, then $gm(X) = m(Y)$ for some $g \in \text{Iso}(X, Y)$. An object $X$ is said to be **generated by canonical augmentation** if its parent is in the same orbit as $m(X)$ under $\text{Aut}(X)$. Objects that are not generated by canonical augmentation are rejected,
and then isomorph rejection is carried out for the remaining objects. In fact, two objects that are generated by canonical augmentation can be isomorphic only if they are have the same parent and are isomorphic by an automorphism of their parent, which simplifies the last isomorph check by restricting the group of possible isomorphisms. Canonical augmentation generates representatives of all equivalence classes if the canonical parent is defined in such a way that for each object, there is an isomorphic object that is generated by canonical augmentation.

In practice, detecting whether an object is generated by canonical augmentation is usually faster by other means than constructing the canonical parent and checking for isomorphism explicitly. Often the canonical parent function \( m \) is not even explicitly defined, and instead the condition for an object to be generated by canonical augmentation is defined.

Canonical augmentation is used in Publication IV for covering arrays. A covering array \( C \) is said to be generated by canonical augmentation if the new column is in a certain orbit of \( \text{Aut}(C) \) when considering the action of \( \text{Aut}(C) \) on the columns. An important feature of the definition is that \( C \) can be generated by canonical augmentation only if all symbols occur in the new column less than \( \mu \) times, where \( \mu \) is a bound that depends on the parent of \( C \). This gives a significant speedup for the part of the search that finds extensions, as extensions where a symbol occurs more than \( \mu \) times are not generated at all. In addition, isomorph rejection needs to be carried out only among children of the same array \( C' \), and because the automorphism group of \( C' \) is usually very small, it can be done with a brute force search over \( \text{Aut}(C') \) in time that is negligible compared to the full search.

### 4.4 Construction by Prescribed Symmetries

A recurring problem in computational combinatorics is generating objects that have a nontrivial automorphism group, instead of generating all objects. Often a full classification is not feasible but one is interested in a partial classification, and requiring for example that the automorphism group be nontrivial restricts the search space enough to make the search computationally feasible. Further, many existence problems have been
solved in the positive by prescribing a group of automorphisms of the object to be searched; see for example [15, 17, 40, 50] for results on binary codes.

In the general setting, we have the group $G$ and we wish to generate objects $X$ that have a nontrivial automorphism group $\text{Aut}(X) \leq G$. The problem can be simplified by the following two observations.

**Theorem 5** ([88, Theorem 4.2]). Let $H$ be a group and $p$ a prime. If $p$ divides $|H|$, then $H$ contains a subgroup of order $p$.

**Theorem 6.** Let $X$ be an object with automorphism group $\text{Aut}(X)$. For all $g \in G$,

$$\text{Aut}(gX) = g\text{Aut}(X)g^{-1}. \quad (4.1)$$

Theorem 5 implies that an object $X$ with nontrivial automorphism group has a group $H$ of automorphisms that is of prime order. Theorem 4.1 implies that if $H'$ is conjugate to $H$, that is there is an isomorphism $g \in G$ for which $H' = gHg^{-1}$, and there is an object $X$ for which $H \leq \text{Aut}(X)$, then there is an isomorphic object $X'$ for which $H' \leq \text{Aut}(X')$. The problem of constructing objects with nontrivial automorphism groups now simplifies to constructing objects that have $H$ as automorphism group for one $H$ of each conjugacy class of subgroups of $G$ of prime order.

When a group $H$ of automorphisms is prescribed, the search usually becomes a lot faster. For example for codes, if the word $c$ is in the code, then all words in its orbit under $H$ are in the code as well, which reduces the search space.

Publication V deals with the problem of coloring the graph $Q_{2}^{8}$ with 13 colors, which is equivalent to partitioning the Hamming space $\{0, 1\}^{8}$ into 13 $(8, M, 3)_{2}$ codes. It was an open problem whether such a partition exists. Because $A(8, 3) = 20$, it is a straightforward observation that the codes must have between 16 and 20 codewords. All earlier attempts of finding such a partition or proving its nonexistence had been deemed computationally infeasible. Limiting the search to partitions with nontrivial automorphism groups allows a computational search in reasonable time.

To find a partition with $H$ as a group of automorphisms, the main observation is that if the partition contains a code $C$, then it also contains the code $hC$ for all $h \in H$. Therefore, instead of finding a subset of 13 codes from a set of all $(8, M, 3)_{2}$ codes with $M \geq 16$, one may consider the sig-
nificantly smaller set of orbits of such codes under $H$, in fact only orbits for which all codes are pairwise disjoint, and find a subset such that the union of those orbits forms a partition. Further, it can be shown that in this case, an automorphism of a partition must be an automorphism of at least one single code in that partition, and the number of groups $H$ to be considered can be limited using classification results for $(8, M, 3)_2$ codes.

### 4.5 Validity of Computational Results

Whenever a program is run to solve a problem, there is a question of how trustworthy its output is. Even with a correct algorithm, there may be a bug in the implementation, or a hardware error may occur while the program is running. Testing a program in small cases against a simpler and slower program helps in development but may not say much about the correctness of the program for large cases. Using existing well-tested libraries for solving common subproblems reduces the probability of an implementation bug, but even a correct program may give a wrong result due to a hardware error; for example, random memory errors are a concern in modern large-scale systems [94].

A simple way of gaining confidence that the results are correct is running two independent implementations of the algorithm, preferably by two different authors, and checking that the results agree. This was used for example in [26,80] among other methods. However, this approach doubles the resource requirements, including the time used for programming.

A method that requires only little extra resources is checking the results of the program output by double counting. A simple example is given in [46] where representatives from each paratopy class of Latin squares of order 8 were generated. The total number of Latin squares of order 8 was calculated using the obtained paratopy class representatives and the orbit–stabilizer theorem and then compared to an earlier result obtained by other methods.

However, in many cases, counting the total number of objects by other means than a full classification is not computationally feasible, so checking against a known result is not possible. Another way to employ double counting is to check the internal consistency of the program output. This
Computational Classification

approach was first proposed by Lam and Thiel in 1989 [52], and has many variations depending on how the search is structured.

A straightforward example occurs in [41], where Steiner triple systems of order 19, denoted briefly by STS(19), were classified starting from a set of nonisomorphic subobjects called seeds. The total number of STS(19) was computed from the set of generated equivalence class representatives, and also from the total number of ways to extend the seeds to an STS(19), which was obtained during the search. The numbers obtained in this way are large; for example, the total number of STS(19) is approximately $1.3 \times 10^{27}$. Therefore, it is highly unlikely that the counts are equal if the results are wrong; it would require either that several errors cancel each other out by chance or that some equivalence classes of STS(19) are completely missing from the search tree but all STS(19) from all other equivalence classes are constructed.

In a more complicated search setting, such as orderly generation, such a direct approach may not be possible if the search step is optimized in such a way that all possible extensions of an object are not generated in the search. Nevertheless, often the total number of objects satisfying some condition can be calculated using some numbers that can be computed during the search without significant additional resource requirements. Sometimes this requires studying the generated objects to detect subobjects that are isomorphic to objects that should occur in the previous level of the search tree. For example in [51], projective planes of order 9 were classified starting from Latin squares of order 8. The consistency check was then performed by studying the generated equivalence classes representatives to find the Latin squares that they could be generated from.

Internal consistency check of the computational results by double counting was used in all publications in this thesis. In Publications I and II, this required searching the generated codes for certain subobjects but even in these cases the additional computational time required for the consistency check was negligible compared to the full search.
5. Classification Results

5.1 MDS Codes

Perfect one-error-correcting MDS codes are \((q + 1, q - 1)_q\) MDS codes. The only linear codes with these parameters up to equivalence are the Hamming codes. For unrestricted codes, the classification of perfect one-error-correcting MDS codes is nontrivial. The \((3,1)_2\) and \((4,2)_3\) MDS codes are unique, which is quite straightforward to show by hand. Alderson proved in 2006 that the \((5,3)_4\) and \((6,3)_4\) MDS codes are unique [3]. For \(q = 6\), no nontrivial MDS codes with \(n - k \geq 2\) exist, which follows from the nonexistence of \((4,2)_6\) MDS codes. For prime powers \(q \geq 9\), a construction by Lindström [56] gives a \((q + 1, q - 1)_q\) MDS code that is not equivalent to the Hamming code.

In Publications I, II, and III, we classify all unrestricted \((n,k)_q\) MDS codes with \(q = 5, 7, 8\) and \(n \geq k + 2\). We show that all \((n,k)_q\) MDS codes for \(q = 5, 7\) and \(n - k \geq 2\) and \(k \geq 2\) are equivalent to linear codes; this result follows from the computational classification of \((5,3)_5\) and \((5,3)_7\) MDS codes in Publication I and a theoretical proof in Publication II. In particular, this implies that the perfect 5-ary and 7-ary MDS codes are unique. Another significant result in Publication II is the computational construction of three inequivalent perfect one-error-correcting 8-ary MDS codes that are not equivalent to linear codes. In Publication III, we finish the classification of 7-ary and 8-ary MDS codes with \(k \geq 3\), \(n - k \geq 2\), which settles the MDS conjecture for \(q = 7, 8\).

The numbers of equivalence classes for MDS codes are given in Tables 5.1, 5.2, and 5.3 for \(q = 5, 7, 8\); references to earlier work and Pub-
Table 5.1. Number of equivalence classes of \((n,k)\) MDS codes with \(n-k \geq 2\)

<table>
<thead>
<tr>
<th>(n \setminus k)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[62]₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[26]₁</td>
<td>I₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[26]₁</td>
<td>a,₁₁</td>
<td>c,₁₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2. Number of equivalence classes of \((n,k)\) MDS codes with \(n-k \geq 2\)

<table>
<thead>
<tr>
<th>(n \setminus k)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[62]₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[26]₁</td>
<td>I₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[26]₁</td>
<td>III₁</td>
<td>c,₁₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>[26]₁</td>
<td>III₁</td>
<td>III₁</td>
<td>c,₁₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>[26]₁</td>
<td>III₁</td>
<td>III₁</td>
<td>III₁</td>
<td>c,₁₁</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>b,₁₁</td>
<td>b,₁₁</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

All other results of this work in these tables are computational. The unmarked entries with a 0 follow from the previously proved cases of the MDS conjecture (see Section 3.3).

These results complete the classification of all \((n,k)\) MDS codes with \(q \leq 8\) and \(n \geq k + 2\). The only other nontrivial MDS codes for \(q \leq 8\) are \((k + 1, k)\) MDS codes, which correspond to Latin hypercubes. Some
Table 5.3. Number of equivalence classes of \((n,k)\) MDS codes with \(n - k \geq 2\)

<table>
<thead>
<tr>
<th>(n \setminus k)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>5</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>6</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>7</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>8</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>9</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>10</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>11</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
</tbody>
</table>

of them have been classified in [69]. In all except a few of the remaining cases, the known lower bound for the number of equivalence classes is so large that a computational classification by constructing equivalence class representatives is probably not feasible; see Publication III, Section 5 for details.

5.2 Covering Arrays

Most computational studies of covering arrays have focused on finding new covering arrays with stochastic methods, but some classification results exist also. An overview of various computational methods is given in [101]. Covering arrays of order 2 have been classified for example in [49,70], and for orders 2 and larger in [24,102]. For strength-2 covering arrays of order at least 3 in particular, there are relatively few results. The case \(CA(v^2; 2, k, v)\) corresponds to sets of MOLS, \(CA(11; 2, k, 3)\) and \(CA(12; 2, k, 3)\) were classified in [24], and \(CA(13; 2, k, 3)\) in [102].

In Publication IV, we classify small covering arrays of strength 2 and orders 3 to 6. Numbers of equivalence classes of covering arrays and uniform covering arrays are shown in Table 5.4 with references to the earliest known publication of the result. For the cases marked with *, a \(CA(N; 2, k + 1, v)\) does not exist, so the arrays have the maximum number of columns for the given \(N\) and \(v\). For the cases marked with †, a \(UCA(N; 2, k + 1, v)\) does not exist but it is not known whether a non-
Classification Results

<table>
<thead>
<tr>
<th>N</th>
<th>k</th>
<th>v</th>
<th>#CA</th>
<th>#UCA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>10</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>11</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>3</td>
<td>13</td>
<td>11</td>
<td>[24]</td>
</tr>
<tr>
<td>*</td>
<td>12</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>[102]</td>
</tr>
<tr>
<td>*</td>
<td>13</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>14</td>
<td>10</td>
<td>3</td>
<td>4490</td>
<td>4117</td>
</tr>
<tr>
<td>*</td>
<td>17</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>18</td>
<td>5</td>
<td>4</td>
<td>201</td>
<td>67</td>
</tr>
<tr>
<td>*</td>
<td>19</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>20</td>
<td>6</td>
<td>4</td>
<td>25760</td>
<td>745</td>
</tr>
<tr>
<td>†</td>
<td>21</td>
<td>7</td>
<td>4</td>
<td>1005</td>
<td>15</td>
</tr>
<tr>
<td>*</td>
<td>26</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>*</td>
<td>27</td>
<td>6</td>
<td>5</td>
<td>11603</td>
<td>3463</td>
</tr>
<tr>
<td>*</td>
<td>28</td>
<td>6</td>
<td>5</td>
<td>75720</td>
<td>344</td>
</tr>
<tr>
<td>*</td>
<td>29</td>
<td>7</td>
<td>5</td>
<td>281</td>
<td>258</td>
</tr>
<tr>
<td>†</td>
<td>30</td>
<td>7</td>
<td>5</td>
<td>18863</td>
<td>15</td>
</tr>
<tr>
<td>*</td>
<td>37</td>
<td>4</td>
<td>6</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>*</td>
<td>38</td>
<td>4</td>
<td>6</td>
<td>8865</td>
<td>6215</td>
</tr>
<tr>
<td>*</td>
<td>39</td>
<td>5</td>
<td>6</td>
<td>289</td>
<td>158</td>
</tr>
<tr>
<td>*</td>
<td>40</td>
<td>5</td>
<td>6</td>
<td>(\geq 388)</td>
<td>(\geq 128)</td>
</tr>
<tr>
<td>†</td>
<td>41</td>
<td>6</td>
<td>6</td>
<td>(\geq 16)</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4. Number of equivalence classes of strength-2 covering arrays

uniform \(CA(N; 2, k+1, v)\) exists. In the last two cases, only covering arrays that satisfy a certain necessary condition to be extended with another column were classified, so the obtained numbers are lower bounds for the total number of equivalence classes.

The new classification results also settle \(CAN(2, 7, 4) = 21\), \(CAN(2, 5, 6) = 39\), \(CAN(2, 6, 6) = 41\), and \(CAN(2, k, 3) = 15\) for \(k = 11, \ldots, 20\) by showing that no smaller covering arrays with the same \(t, k, v\) exist; the existence of arrays attaining these bounds is known. The case \(CAN(2, 5, 6) = 39\) was also recently settled independently by Torres-Jimenez [96]. Known values and bounds of \(CAN(2, k, v)\) for small \(v\) and \(k\) are given in Table 5.5. Refer-
Classification Results

<table>
<thead>
<tr>
<th>$k \backslash v$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9</td>
<td>16</td>
<td>25</td>
<td>[95]37 [95]</td>
</tr>
<tr>
<td>5</td>
<td>[95]11 [95]</td>
<td>16</td>
<td>25</td>
<td>IV39 [77]</td>
</tr>
<tr>
<td>6</td>
<td>[95]12 [95]</td>
<td>[95]19 [95]</td>
<td>25</td>
<td>IV41 [77]</td>
</tr>
<tr>
<td>7</td>
<td>[95]12 [95]</td>
<td>IV21 [95]</td>
<td>[95]29 [95]</td>
<td>IV41–42 [77]</td>
</tr>
</tbody>
</table>

Table 5.5. Bounds for CAN(2, $k$, $v$)

ences for the lower and upper bounds are given in the left and right side superscript, respectively. Unmarked entries are cases that are equivalent to sets of MOLS. In addition to the entries in the table, it was also known earlier that CAN(2, $k$, 3) ≤ 15 for $k = 11, \ldots, 20$ [77], and Publication IV shows that the bound is tight.

The results indicate that at least for small strength-2 covering arrays, most equivalence classes of optimal arrays are uniform. This gives some credibility to Conjecture 3.

5.3 The Chromatic Number of the Square of the $8$-Cube

Studying colorings of $Q^k_n$ and in particular the chromatic numbers of $Q^k_n$ is related to the more general problem of chromatic numbers of a certain family of graphs, called cube-like graphs (see [83] and [39, Section 9.7]), and also to a scalability problem of certain optical networks [104], and has gained wide interest; see for example [45, 54, 74, 78, 87, 108]. Asymptotically, $\chi(Q^2_n) \sim n$ and $\chi(Q^3_n) \sim 2n$ [78]. The maximum size of an $(n, M, d)_2$ code, $A(n, d)$, gives a lower bound

$$\chi(Q^d_n) \geq \left\lceil \frac{2^n}{A(n, d)} \right\rceil. \tag{5.1}$$

For each $n$, there is a construction [57] that gives a coloring of $Q^2_n$ with $2^\lceil(n+1)/2\rceil$ colors, so

$$\chi(Q^2_n) \leq 2^\lceil(n+1)/2\rceil. \tag{5.2}$$

The smallest open case of $\chi(Q^2_n)$ has been $n = 8$, for which (5.1) gives $\chi(Q^2_8) \geq 13$, and 14-colorings of $Q^2_8$ were found in the early 1990s by
Classification Results

<table>
<thead>
<tr>
<th>$n$</th>
<th>$A(n, 3)$</th>
<th>$\chi(Q_n^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>[11]$^2$</td>
<td>4$^{[57]}$</td>
</tr>
<tr>
<td>5</td>
<td>[11]$^4$</td>
<td>8$^{[57]}$</td>
</tr>
<tr>
<td>6</td>
<td>[11]$^8$</td>
<td>8$^{[57]}$</td>
</tr>
<tr>
<td>7</td>
<td>[32]$^{16}$</td>
<td>8$^{[57]}$</td>
</tr>
<tr>
<td>8</td>
<td>[12]$^{20}$</td>
<td>$13V^{[57]}$</td>
</tr>
<tr>
<td>9</td>
<td>[10]$^{40}$</td>
<td>13–14$^{[54]}$</td>
</tr>
<tr>
<td>10</td>
<td>[80]$^{72}$</td>
<td>15–16$^{[57]}$</td>
</tr>
<tr>
<td>11</td>
<td>[80]$^{144}$</td>
<td>15–16$^{[57]}$</td>
</tr>
<tr>
<td>12</td>
<td>[11]$^{256}$</td>
<td>16$^{[57]}$</td>
</tr>
<tr>
<td>13</td>
<td>[11]$^{512}$</td>
<td>16$^{[57]}$</td>
</tr>
<tr>
<td>14</td>
<td>[11]$^{1024}$</td>
<td>16$^{[57]}$</td>
</tr>
<tr>
<td>15</td>
<td>[32]$^{2048}$</td>
<td>16$^{[57]}$</td>
</tr>
</tbody>
</table>

Table 5.6. Bounds for $\chi(Q_n^2)$

Hougardy [108] and Royle [39, Section 9.7]. Recently, Lauri [54] found a 14-coloring of $Q_9^2$. In Publication V, we settle that $\chi(Q_8^2) = 13$ by constructing partitions of $\{0, 1\}^8$ into 13 one-error-correcting codes.

Known cases and bounds for $\chi(Q_n^2)$ for small $n$ are given in Table 5.6 with references for $A(n, 3)$ and for the upper bounds given in superscript.
6. Conclusions

In this thesis we have developed computational methods for generating and classifying codes and related structures. We have completed the classification of unrestricted MDS codes for alphabets of size at most 8 and minimum distance at least 3. The results include showing that the perfect 5-ary and 7-ary one-error-correcting MDS codes are unique, and construction of three new equivalence classes of perfect 8-ary MDS codes. In addition, using similar methods, we have classified many small strength-2 covering arrays. While covering arrays with new parameters were not found, the classification results settle the sizes of optimal covering arrays for some parameters and improve some bounds. Finally, we have settled the long-standing open problem of the chromatic number of the square of the 8-cube graph by using existing classification results for codes and the method of prescribing automorphism groups.

This thesis contributes to two main research directions. Firstly, the classification results can be used directly for understanding the behavior of the objects in question. Studying the nonlinear perfect MDS codes may open way to new theoretical constructions of nonlinear codes, and the coloring of the square of the 8-cube may help in understanding colorings of hypercube graphs and other cube-like graphs. In addition, the complete catalogs of small covering arrays can be used to assist in forming conjectures on the properties of optimal covering arrays, which can serve as a stepping stone to useful theoretical observations also for large covering arrays. Publication V also directly demonstrates the usefulness of classification results by using classification of binary codes as a starting point for solving another problem.
The second research direction is using the methods developed in this thesis for other related problems and further development of the algorithms used in this thesis. For example, orthogonal arrays with $\lambda > 2$ or covering arrays with $t > 2$ were not studied here but the ideas used in this thesis may be directly applicable to these cases. In addition, most of the best known covering arrays have been found using stochastic search methods [23], which were not studied in this thesis. Coupling stochastic methods with the algorithms used in this thesis to prune the search tree may yield new small covering arrays. Another interesting open problem related to this work is the existence of a set of 3 MOLS of order 10. In addition, while the method of prescribing automorphisms is not new, it was again proven useful in this work, and will likely continue to be useful in research.
References


References


[28] L. Euler, Recherches sur une nouvelle espece de quarres magiques (1782).


Errata

Publication III

The nonexistence of a $(10, 4)_8$ MDS code is erroneously reported as a new result in Table 4. The result was proved by Alderson [T. L. Alderson, Extending MDS codes, *Ann. Comb.* 9, 125–135 (2005)].