Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 94
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 94
Espoo 2005 HUT-TCS-A94

ALGORITHMS FOR CLASSIFICATION OF COMBINATORIAL OBJECTS

Petteri Kaski

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 94
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 94
Espoo 2005 HUT-TCS-A94

ALGORITHMS FOR CLASSIFICATION OF COMBINATORIAL OBJECTS

Petteri Kaski

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 15th of June, 2005, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FI-02015 TKK
Tel. +358-0-451 1
Fax. +358-0-451 3369
E-mail: lab@tcs.hut.fi

ISBN 951-22-7711-5
ISSN 1457-7615

Multiprint Oy
Helsinki 2005

ABSTRACT: A recurrently occurring problem in combinatorics is the need to completely characterize a finite set of finite objects implicitly defined by a set of constraints. For example, one could ask for a list of all possible ways to schedule a football tournament for twelve teams: every team is to play against every other team during an eleven-round tournament, such that every team plays exactly one game in every round. Such a characterization is called a *classification* for the objects of interest. Classification is typically conducted up to a notion of structural equivalence (isomorphism) between the objects. For example, one can view two tournament schedules as having the same structure if one can be obtained from the other by renaming the teams and reordering the rounds.

This thesis examines algorithms for classification of combinatorial objects up to isomorphism. The thesis consists of five articles—each devoted to a specific family of objects—together with a summary surveying related research and emphasizing the underlying common concepts and techniques, such as backtrack search, isomorphism (viewed through group actions), symmetry, isomorph rejection, and computing isomorphism. From an algorithmic viewpoint the focus of the thesis is practical, with interest on algorithms that perform well in practice and yield new classification results; theoretical properties such as the asymptotic resource usage of the algorithms are not considered.

The main result of this thesis is a classification of the Steiner triple systems of order 19. The other results obtained include the nonexistence of a resolvable 2-$(15, 5, 4)$ design, a classification of the one-factorizations of k-regular graphs of order 12 for $k \leq 6$ and $k = 10, 11$, a classification of the near-resolutions of 2-$(13, 4, 3)$ designs together with the associated thirteen-player whist tournaments, and a classification of the Steiner triple systems of order 21 with a nontrivial automorphism group.

KEYWORDS: classification algorithm, isomorphism, isomorph rejection, near-resolvable design, one-factorization, orderly algorithm, regular graph, resolvable design, Steiner triple system

CONTENTS

PREFACE

This thesis is the result of studies and research carried out at the Laboratory for Theoretical Computer Science of Helsinki University of Technology (TKK) from 2001 to 2005.

A number of people have contributed to this thesis, either directly or indirectly. First, I would like to thank Professor Patric Östergård, my thesis instructor, for guiding me into the intricacies of combinatorial classification—and the world of science in general—through extensive collaboration and advice, not to mention our joint monograph project on classification algorithms. I am grateful to Professor Ilkka Niemelä, the laboratory head, for the opportunity to work at the Laboratory for Theoretical Computer Science and for creating the encouraging environment at the laboratory. To Professor Pekka Orponen I am much obliged for his acting as my thesis supervisor and for collaboration in the NAPS Algorithmics project, not to mention granting a more than occasional peek into an extensive personal library. I also would like to thank everyone at the laboratory for the congenial working atmosphere; in the context of this thesis I am especially indebted to Harri Haanpää for our collaboration and to Tommi Junttila for contributing to my understanding of isomorphism algorithms.

This research has been funded by the Helsinki Graduate School in Computer Science and Engineering (HeCSE) and by the Academy of Finland. Further financial support in the form of personal grants from the Foundation of Technology (Tekniikan Edistämissäätiö) and the Nokia Foundation is gratefully acknowledged.

Finally, I would like to thank my parents and friends for their support and encouragement along the years.

Otaniemi, June 2005

Petteri Kaski

1 INTRODUCTION

Combinatorics [24, 151] is commonly defined as the branch of mathematics studying finite arrangements of certain basic objects—such as elements or subsets of a given finite set—where an arrangement is required to meet certain constraints. This is obviously a rather imprecise definition, but it accurately reflects the variety in the types of objects studied. For the skeptical, a browse through the *Handbook of Combinatorics* [77] or the *CRC Handbook of Combinatorial Designs* [34] provides ample evidence of this variety.

Depending on the difficulty of satisfying the given constraints, combinatorial objects range from elementary objects—such as permutations and partitions—to objects with more elaborate structure such as graphs with various regularity properties [12, 18, 71], designs [8, 34], and codes [30, 157, 201].

For elementary objects, the main interest is typically to *enumerate* the objects; this is the goal of *enumerative combinatorics* [227]. Also of interest is to sample the objects uniformly at random [97] and to develop ranking and unranking functions for the objects relative to some prescribed order relation [127].

For objects with more elaborate structure, the primary problem is that of *existence*; that is, determining whether the constraints can be satisfied, and if so, providing explicit constructions for objects meeting the constraints. A more ambitious goal is to completely characterize all the objects specified by the constraints. Typically such a characterization assumes the form of a *classification* that partitions the objects into a collection of classes such that:

(1.1) the structure of the objects in each class is evident; for example, an explicit construction is provided for one object from each class; and

(1.2) the objects within a class are identical in structure—or *isomorphic*—with respect to the properties under study; for example, it is common to view two objects as isomorphic if one can be obtained from the other by appropriately renaming and/or reordering the basic objects that make up an object.

For the purpose of this introduction, combinatorial objects and combinatorial classification are perhaps best briefly illustrated by the following practical example. Indeed, many combinatorial objects arise as solutions to everyday problems.

> Eight football teams are to play a tournament where every team plays once against every other team. The task is to produce a tournament schedule consisting of (seven) rounds, such that every team plays one game in each round.

To characterize all such tournament schedules, it is convenient to view any two tournament schedules as identical in structure (isomorphic) if one can be obtained from the other by renaming the teams and reordering the rounds. Relative to this notion of isomorphism, the possible tournament schedules are partitioned into six isomorphism classes. Representatives from the classes

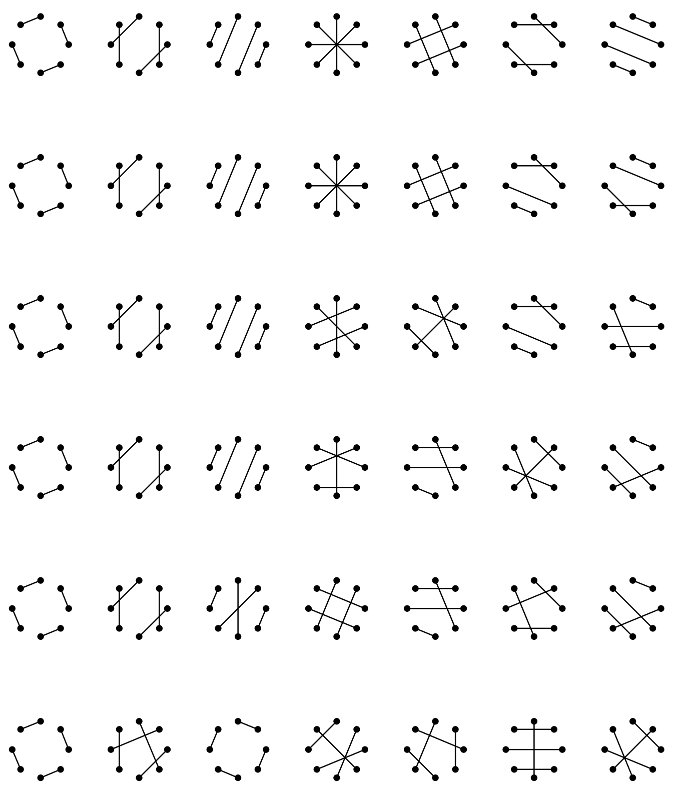
Figure 1.1: The six possible tournament schedules for eight teams

are depicted in Fig. 1.1, where the vertices represent the teams and the edges represent the games; each row gives one tournament schedule of seven rounds.

The tournament schedules in Fig. 1.1 immediately provoke a number of questions. Namely, it is straightforward to check that each tournament schedule meets the constraints, but how can we be certain that the classification is complete; that is, that it contains all possible isomorphism classes? Furthermore, even if the classification is complete, are the six tournament schedules really pairwise nonisomorphic? There are lots of possibilities to rename the teams and to reorder the rounds, so this is surely not obvious. How was the classification obtained in the first place?

Combinatorial classification is laborious. Except in rare cases where nonexistence or uniqueness up to isomorphism can be established by a clever argument, there appears to be no other way to obtain a classification of combinatorial objects with nontrivial structure than to systematically try out all the possibilities of meeting the constraints. Thus, combinatorial classification is fundamentally connected with algorithms and computation. Unlike a traditional mathematical proof, which can—at least in principle—be checked without much effort by an expert, a classification usually requires significant computational effort to produce and to verify.

This thesis examines algorithms for classification of combinatorial objects up to isomorphism, or *classification algorithms*. Motivation for research into classification algorithms is twofold.

First, the classification results obtained are in themselves of mathematical interest. A classification can be used to test conjectures and to gain further insight into the structure of the objects being studied. For example, the classification of the Steiner triple systems of order 19 [P2]—the main result of this thesis—resulted in the discovery of nonisomorphic Steiner triple systems with equivalent point codes [105]. In addition, the objects often have direct practical applications (see for example [36, 39])—such as in scheduling a tournament. Classification techniques and exhaustive search can also sometimes be used to settle the nonexistence of an object when a traditional nonexistence proof is lacking. Perhaps the most notable such result to date is the nonexistence of a projective plane of order 10 [135].

Second, from a computer science point of view, the relevance of research into classification algorithms lies in the design of exhaustive search algorithms for hard combinatorial problems involving symmetry. Typically constructing the objects to be classified involves finding all solutions to an instance of a hard combinatorial problem, such as the maximum clique problem or the exact cover problem. In addition, techniques for computing isomorphism are required to eliminate symmetry from the search space and to remove isomorphic objects from the output of the algorithm.

The viewpoint adopted in this thesis is practical, as opposed to theoretical in the sense of complexity theory [66, 92, 196]. The present focus is on developing classification algorithms with good enough practical performance to yield new classification results for objects of combinatorial interest, whereas theoretical properties such as asymptotic resource usage of the algorithms in relation to the size of the objects being classified (cf. [99]) are not investigated. Theoretical investigations in this direction can be found in [72] and the references therein. A closely related theoretical line of study is that of counting, approximate counting, and random sampling of combinatorial objects [73, 97].

In practice, the available finite computing resources place an upper bound on what can be achieved with algorithmic tools. There are two main obstacles in this respect.

An intrinsic obstacle is the often occurring explosive growth in the number of nonisomorphic objects as a function of the object size—if there are more nonisomorphic objects than the total number of instructions all the processors allocated for the study can together execute in a hundred years, then a classification is hardly practical. Returning to the example, the classification for tournament schedules for eight teams (or equivalently, *one-factorizations* [236] of the complete graph K_8) was obtained in the days of hand calculation [52] (as cited in [54]); a full exposition appears in [237]. For ten teams (396 nonisomorphic schedules [67]) and twelve teams (526915620 nonisomorphic schedules [54]), the use of a computer is required for a complete classification. For fourteen teams and beyond, a complete classification appears to be already out of reach for contemporary computers. In [54] the number of distinct one-factorizations of K_{14} is estimated to be approximately $9.876 \cdot 10^{28}$, which gives the estimate $1.133 \cdot 10^{18}$ for the number of isomorphism classes, assuming that most one-factorizations have a trivial automorphism group; asymptotic lower and upper bounds for the number of isomorphism classes appear in [23, Theorem 4.2] and [236].

Another obstacle to classification occurs essentially due to lack of understanding of the structure of the objects, which forces one to search for the objects of interest within an often considerably larger class of objects. For example, the properties required from a tournament schedule do not directly suggest how to proceed at generating all possible schedules. An obvious possibility is to proceed by augmenting a partial schedule one round of games at a time; unfortunately, with such an approach it can—and does—occur that a partial schedule cannot be extended to a full schedule (cf. [31]). A further case in point occurs when the goal is to establish the nonexistence of an object by "considering all possible ways" to construct it, whereby inevitably some searching must be done.

In essence, this thesis is about how to proceed at classification without doing too much searching and redundant computation, so that occasionally a practical algorithm and thereby a classification result is obtained.

2 STRUCTURE OF THE THESIS

This thesis consists of five articles and this five-chapter summary. Together with the summary, the following five articles constitute this thesis.

[P1] P. Kaski and P. R. J. Östergård, There exists no (15,5,4) RBIBD, *Journal of Combinatorial Designs* **9** (2001) 357–362.

[P2] P. Kaski and P. R. J. Östergård, The Steiner triple systems of order 19, *Mathematics of Computation* **73** (2004) 2075–2092.

[P3] P. Kaski and P. R. J. Östergård, One-factorizations of regular graphs of order 12, *Electronic Journal of Combinatorics* **12**(1) (2005) #R2, 25pp.

[P4] H. Haanpää and P. Kaski, The near resolvable 2-$(13, 4, 3)$ designs and thirteen-player whist tournaments, *Designs, Codes and Cryptography* **35** (2005) 271–285.

[P5] P. Kaski, Isomorph-free exhaustive generation of designs with prescribed groups of automorphisms, *SIAM Journal on Discrete Mathematics*, to appear.

The present chapter contains a brief description of the contents of this thesis and the role of the author in articles with more than one author. Chapter 3 surveys general techniques employed in classification algorithms. Chapter 4 reviews classification techniques employed in the classification of designs and codes. Chapter 5 presents some conclusions and topics for future work.

2.1 SUMMARY OF THE ARTICLES IN THE THESIS

The contents of the articles in this thesis are summarized briefly in what follows. In Chapters 3 and 4 each of the articles is placed in context with respect to related research.

[P1]: An orderly algorithm [61, 205] relying on a correspondence between resolutions of 2-designs and certain error-correcting codes [219] is used to establish the nonexistence of a resolvable 2-$(15, 5, 4)$ design; or, equivalently, the nonexistence of a $(14, 15, 10)_3$ error-correcting code.

[P2]: The Steiner triple systems of order 19 are classified up to isomorphism by employing a combination of an algorithm for the exact cover problem [116] and generation by canonical augmentation [170].

[P3]: Three techniques for classifying one-factorizations of regular graphs are developed. The first two techniques are based on viewing a one-factorization of a k-regular graph of order $2n$ as an equireplicate $(k, 2n, k - 1)_n$ code. The first technique uses an orderly algorithm analogous to the one in [P1] to construct the codes. The second technique relies on coordinate-by-coordinate construction; isomorph rejection is achieved by a combination of generation by canonical augmentation and orderly generation. The third algorithm is based on viewing a one-factorization of the complete graph K_{2n} as a certain triple system on $4n - 1$ points, and then employing techniques analogous to those used in [P2] to arrive at a classification algorithm. The developed techniques are then used to classify the one-factorizations of k-regular graphs of order 12 for $k \leq 6$ and $k = 10, 11$. For $k = 11$, the results obtained corroborate the results in [54].

[P4]: A correspondence between near resolutions of 2-designs and certain types of codes is introduced. An orderly algorithm utilizing this correspondence is then developed to classify up to isomorphism the near resolutions of 2-$(13, 4, 3)$ designs. Based on the classification of near resolutions, the thirteen-player whist tournaments are classified. As a consequence of the classification, the nonexistence of a triplewhist tournament for thirteen players is established.

[P5]: The classification of designs with prescribed groups of automorphisms is studied in the case where a prescribed group H has a large index in its normalizer $N_G(H)$, where G is the isomorphism-inducing group. A technique based on generation by canonical augmentation analogous to the seed-based approach in [P2, P3] is developed for coping with normalizer-induced symmetry in the search space and for performing isomorph rejection without the need to keep a record of the isomorphism class representatives encountered. As an application, a classification of the Steiner triple systems of order 21 with a nontrivial automorphism group is produced.

2.2 CONTRIBUTIONS OF THE AUTHOR

This section summarizes the contributions of the author to the articles constituting this thesis.

The author of this thesis is the sole author of [P5], and has significantly contributed to writing the articles [P1, P2, P3, P4].

In [P1, P2, P3] the algorithm designs are joint work of the author of this thesis and the coauthor. The author of this thesis is responsible for implementing the algorithms and carrying out the searches.

In [P4] the algorithm used to classify the near resolutions of 2-$(13, 4, 3)$ designs and its implementation are due to the author of this thesis. The algorithms used to produce the classification of thirteen-player whist tournaments from the near resolutions and the subsequent analysis of the classified tournaments are due to the coauthor.

3 CLASSIFICATION ALGORITHMS

This chapter reviews the general techniques and mathematical concepts used in the design of classification algorithms; the motivation is to provide a common framework for the algorithms in [P1, P2, P3, P4, P5]. Except for the somewhat modified treatment of generation by canonical augmentation [170] in Sect. 3.4.3, none of the material in the present chapter is new. The material has been either compiled from the cited references, or can be considered "folklore".

3.1 CLASSIFICATION PROBLEMS

In an abstract setting, combinatorial classification is concerned with an implicitly given finite set Γ equipped with an equivalence relation $\cong$ called *isomorphism*. The *classification problem* associated with $(\Gamma, \cong)$ is to output exactly one element from every equivalence class defined by $\cong$ on Γ, or conclude that Γ is empty. Here it is assumed that all elements of Γ admit a natural finite representation; for example, as finite binary strings. A *classification algorithm* for $(\Gamma, \cong)$ is an algorithm that solves the classification problem for $(\Gamma, \cong)$ and then halts. Alternatively to classification, one often speaks of *isomorph-free exhaustive generation*.

From a practical viewpoint, classification problems exhibit varying characteristics depending on the required structure of the objects and the isomorphism relation; cf. [16, 170]. In terms of structure, classification problems range roughly between two extreme types. On one hand, there exist objects with a recursive structure that enables exhaustive generation of the objects. For example, a permutation of $1, 2, \ldots, n$ (viewed as a list) reduces to a permutation of $1, 2, \ldots, n-1$ if we delete n from the list. Reversing this process, we obtain a method for listing permutations. Similarly, a tree on n vertices reduces to a tree on $n-1$ vertices if we delete a leaf node. With such "elementary" objects, it is customary to use the term *listing* instead of classification. Listing algorithms have been extensively studied; see [127, 212, 241]. On the other hand, there exist objects for which a recursive structure characterizing all the objects is lacking, or is not known. In the extreme, not a single object is known. For example, it is currently an open problem whether a 2-$(22, 8, 4)$ design exists [7, 83, 174, 192, 207]. In between these extremes there exist many intermediate cases. For example, it may be the case that numerous recursive and/or direct constructions for the objects are known, but these do not characterize all the objects. Steiner triple systems are an example of such objects; a comprehensive reference is [40]. Similarly, it may be possible to enumerate the distinct objects with reasonable effort (cf. [6, 54, 175]), but producing a classification up to isomorphism—or even enumerating the isomorphism classes (cf. [171])—appears to be much more difficult.

Another rough division into types can be made on basis of the isomorphism relation, which ranges from trivial (no two distinct objects are isomorphic) to computationally demanding. A canonical example of a computationally demanding isomorphism relation is graph isomorphism, for which currently no polynomial-time algorithm is known despite extensive effort; cf. Sect. 3.3.3.

With respect to this rough division into types, the classification problems studied in this thesis can be considered as lacking a recursive structure and having a computationally demanding isomorphism relation.

For the purpose of illustrating the techniques surveyed, the following tiny example (cf. [133]) will be considered throughout this chapter.

Example 1 The task is to classify up to isomorphism all 4×4 matrices with entries from $\{0, 1\}$ such that every row and every column contains exactly two 1s. Two matrices are regarded as isomorphic if one can be obtained from the other by an independent permutation of the rows and the columns.

3.2 EXHAUSTIVE GENERATION

A classification algorithm consists of two principal ingredients. One is a method for generating the objects of interest so that at least one object is generated from every isomorphism class; another is an accompanying method for removing isomorphic objects from consideration so as to achieve isomorph-free generation.

From the point of view of algorithm design, the main difficulty is usually in coming up with a computationally feasible way to generate the objects of interest. Once there is a way to generate at least one object from every isomorphism class, then it is often relatively straightforward—at least with the help of appropriate isomorphism invariants—to filter out isomorphic objects so that isomorph-free generation is achieved; cf. [16].

A practical exhaustive generation strategy in most cases requires at least some insight specific to the objects of interest in identifying a sequence of intermediate objects (typically, subobjects of some kind) along which exhaustive generation may be carried out. Obviously, in this respect classification algorithms are more or less specific to the objects of interest, and relatively few general techniques can be found. Techniques specific to the types of objects studied in articles [P1, P2, P3, P4, P5] are surveyed in Chapter 4.

To provide a general discussion, we require a formal model for studying exhaustive generation and search. Such a model is provided by backtrack search (Sect. 3.2.1) and search trees (Sect. 3.2.2). Once this model is available, we proceed to discuss isomorphism (Sect. 3.3) and techniques for isomorph rejection (Sect. 3.4) in a general setting.

3.2.1 Backtrack Search

Backtrack search or *backtracking* [76, 235] is an algorithmic principle that corresponds to the intuitive "step by step, try out all the possibilities"-approach to solving a finite problem. Textbooks that discuss backtrack search include [127, 204, 208].

A *partial solution* in a backtrack search is an ordered tuple $(a_1, a_2, \ldots, a_\ell)$, where the a_i are elements of a finite set U. It is assumed that every possible solution to the problem at hand can be represented by such a finite tuple. A partial solution that constitutes a solution to the problem at hand is called a *feasible solution*.

Backtrack search operates by recursively extending a partial solution one element at a time as dictated by the constraints of the problem at hand. More formally, given a partial solution $(a_1, a_2, \ldots, a_\ell)$ as input, a backtrack search procedure computes a *choice set* $A_{\ell+1} \subseteq U$, and recursively invokes itself with each possible input $(a_1, a_2, \ldots, a_\ell, a_{\ell+1})$, where $a_{\ell+1} \in A_{\ell+1}$. After all choices have been considered, the procedure returns control—or *backtracks*—to the invoking procedure. The initial invocation is made with the empty tuple "()", and feasible solutions are processed as they are discovered. To generate all feasible solutions, it is required that the choice sets satisfy the following completeness property: if $(a_1, a_2, \ldots, a_n)$ is a feasible solution, then $a_{\ell+1}$ must belong to the choice set $A_{\ell+1}$ associated with $(a_1, a_2, \ldots, a_\ell)$ for all $0 \le \ell \le n-1$. A pseudocode description of backtrack search is given as Algorithm 1.

```
procedure BTRK($\ell$: integer, $(a_1, a_2, \ldots, a_\ell)$: partial solution)
  1: if $(a_1, a_2, \ldots, a_\ell)$ is a feasible solution then
  2:     process $(a_1, a_2, \ldots, a_\ell)$
  3: end if
  4: compute $A_{\ell+1}$ based on $(a_1, a_2, \ldots, a_\ell)$ and the constraints imposed by
     the problem at hand
  5: for all $a_{\ell+1} \in A_{\ell+1}$ do
  6:     BTRK($\ell + 1$, $(a_1, a_2, \ldots, a_{\ell+1})$)
  7: end for
end procedure
procedure BACKTRACK
  8: BTRK(0, ())
end procedure
```

Algorithm 1: Backtrack search

Example 2 One possible backtrack solution to the running example is to construct the 4×4 matrices row by row so that U consists of all possible rows with two 1s; that is, $U = \{1100, 1010, 1001, 0110, 0101, 0011\}$. Given a partial solution with ℓ rows, the choice set $A_{\ell+1}$ consists of those rows in U whose addition does not violate the constraint that every column must contain at most two 1s. A partial solution is feasible if it has four rows.

Backtrack search is obviously a general principle rather than a complete solution to a problem requiring exhaustive search. The practicality of a backtrack algorithm is determined by the algorithm design choices made when transforming the abstract problem into the backtrack search framework.

In general, it is advisable to look at the constraints required from the feasible solutions, and attempt to relax these only as little as possible for purposes of generation. Furthermore, any structural properties implied by the constraints should be employed in restricting the partial solutions that need to be traversed. Often this requires mathematical insight into the objects being classified as well as experience and experimentation as to what properties can be efficiently exploited from a computational viewpoint. Further design principles for fast backtracking algorithms together with examples can be found in [172, 208]. Techniques for estimating the resource requirements of backtrack algorithms appear in [114, 170, 203, 204].

Existing backtrack algorithms for well-known combinatorial optimization problems can often be employed in solving an exhaustive generation problem or a related subproblem. Recurrently encountered optimization problems in this respect include the maximum clique problem [27, 191] (see also [15, 98, 186]), the exact cover problem [116, 127], graph coloring [98, 127, 197] (see also [96]), and the problem of solving a system of Diophantine linear equations with lower and upper bounds on the variables [1, 121, 126, 127, 155, 215, 238, 239] (see also [216, 243]). Depending on the extent of symmetry in the problem instance being solved, such an algorithm may require isomorph rejection on partial solutions to eliminate redundant computation; cf. Sects. 3.3 and 3.4.

3.2.2 Search Trees

In studying a search strategy, it is convenient to work with a static object capturing the essential structure of the search. *Search trees* (alternatively, *state space trees* [127]) constitute such a device. In essence, a search tree contains all the objects encountered in a search, with edges connecting objects related by a single search step.

Example 3 Figure 3.1 shows a search tree for the row-by-row backtrack search strategy in Example 2. Parts of the search tree have been truncated due to space limitations; the truncated subtrees are marked with three dots "...".

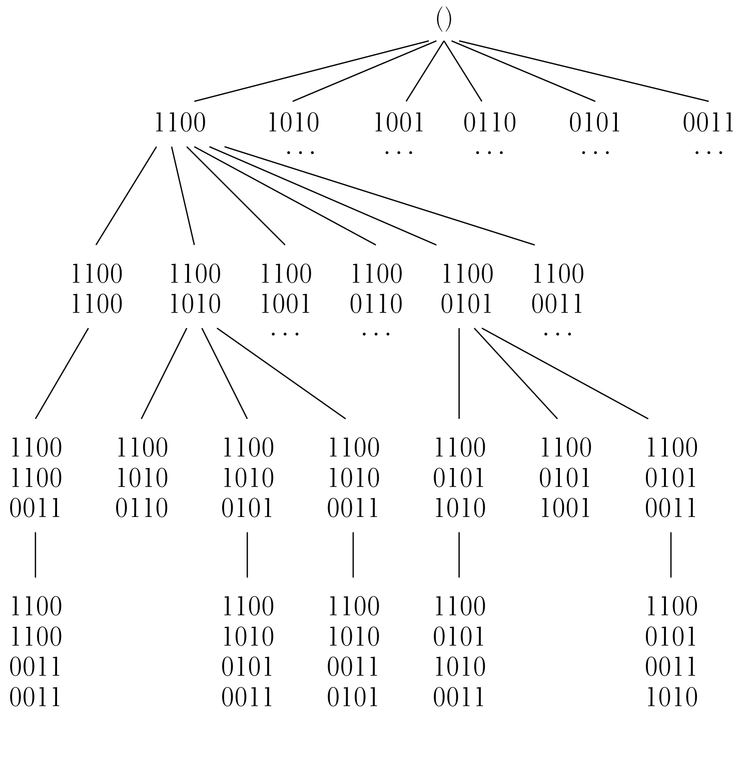

Figure 3.1: A search tree (truncated in part)

We follow [53] in graph-theoretic definitions and notation. Formally, a *search tree* is a rooted tree T, where the nodes (vertices) in the tree T are objects occurring in the search. We write $V(T)$ for the set of all nodes in T, and $r(T)$ for the root node of T. For a nonroot node $\mathcal{X} \in V(T)$, the *parent* $p(\mathcal{X})$ is the node immediately following $\mathcal{X}$ on the path from $\mathcal{X}$ to the root $r(T)$. Conversely, a nonroot node $\mathcal{Y} \in V(T)$ is a *child* of a node $\mathcal{X}$ if $p(\mathcal{Y}) = \mathcal{X}$. We write $C(\mathcal{X})$ for the set of all children of $\mathcal{X}$. Two nodes $\mathcal{Y}, \mathcal{Z} \in V(T)$ are *siblings* if $p(\mathcal{Y}) = p(\mathcal{Z})$. A node is a *leaf* if it has no children. The *level* of a node $\mathcal{X}$ in T is the length of the path connecting $\mathcal{X}$ to the root $r(T)$. For nodes $\mathcal{X}, \mathcal{Y} \in V(T)$, $\mathcal{X}$ is an *ancestor* (respectively, *descendant*) of $\mathcal{Y}$ if $\mathcal{X}$ ($\mathcal{Y}$) occurs on the path from $\mathcal{Y}$ ($\mathcal{X}$) to the root $r(T)$. For a node $\mathcal{X} \in V(T)$, the *subtree of T rooted at $\mathcal{X}$* is the rooted tree with root $\mathcal{X}$ induced by all the descendants of $\mathcal{X}$ in T.

In the context of classification algorithms, we adopt the convention that a search tree contains two types of objects: *objects of interest* (objects to be classified) and *intermediate objects*.

For purposes of description and analysis it is often convenient to work with a tree that is larger than the tree actually traversed by the search algorithm. For example, the algorithm can disregard (prune) a subtree rooted at a node because a bounding function evaluated at the node indicates that the subtree contains no objects of interest. Alternatively, an isomorphism computation can indicate that a subtree rooted at a node contains only objects that are isomorphic to objects encountered earlier, so the subtree can be pruned. In such cases working with the unpruned tree is considerably easier and often suffices to establish the properties required from the search algorithm. We adopt the convention that the term "search tree" refers to a rooted tree that contains as a subtree the tree actually traversed by the algorithm.

The execution of a search algorithm can typically be modeled by a depth-first traversal of an associated search tree; that is, starting from the root node, the algorithm recursively traverses the children of a node before returning from the node. Pseudocode for a generic depth-first traversal with pruning is given in Algorithm 2; cf. Algorithm 1. Note that we do not prescribe any order in which the children of a node are traversed—unless explicitly indicated otherwise, the structure of the algorithms is such that any order will do.

```
procedure D-F-TRAV($\mathcal{X}$: object)
 1: if pruning condition is true at $\mathcal{X}$ then
 2:     return
 3: end if
 4: process $\mathcal{X}$
 5: for all $\mathcal{Y} \in C(\mathcal{X})$ do
 6:     D-F-TRAV($\mathcal{Y}$)
 7: end for
end procedure
procedure DEPTH-FIRST-TRAVERSE($T$: search tree)
 8: D-F-TRAV($r(T)$)
end procedure
```

Algorithm 2: Depth-first traversal of a search tree

3.3 ISOMORPHISM AND SYMMETRY

The isomorphism relations associated with essentially every family of combinatorial objects encountered in practice—including all types of objects in this thesis—can be captured in a general setting either using *category theory* [156] or, what can be seen as a special case of the former, *group actions* [75, 107, 108, 139]. Of these two, category theory is perhaps more suitable for the mathematical study of transformations between different families of combinatorial objects via categories (groupoids), functors, and reconstructibility (cf. [4]), whereas group actions provide a more fixed finite framework that is more applicable for algorithms and computation (cf. [21, 130, 133, 140, 168, 170]).

Here we will base the exposition on group actions.

3.3.1 Group Actions and Isomorphism

Let G be a finite group and let Ω be a finite nonempty set. A *group action* of G on Ω is a group homomorphism $\gamma : G \to \mathrm{Sym}(\Omega)$, where $\mathrm{Sym}(\Omega)$ is the symmetric group on Ω. For brevity, we write S_n for $\mathrm{Sym}(\Omega)$ when $|\Omega| = n$ and Ω is clear from the context. For $g \in G$ and $\mathcal{X} \in \Omega$, we write simply $g\mathcal{X}$ for the image of $\mathcal{X}$ under $\gamma_g \in \mathrm{Sym}(\Omega)$ if the action γ is arbitrary or otherwise clear from the context. As is apparent from the notation, we assume that a group acts from the left; that is, $(g_1g_2)\mathcal{X} = g_1(g_2\mathcal{X})$ for all $g_1, g_2 \in G$ and $\mathcal{X} \in \Omega$. Accordingly, permutations compose from right to left; for example, $(1\,2)(2\,3) = (1\,2\,3)$ in cycle notation. A general introduction to group theory can be found in [209]. Permutation groups and group actions are discussed in [25, 55, 107, 139, 240].

For $\mathcal{X}, \mathcal{Y} \in \Omega$, we write $\mathcal{X} \cong_G \mathcal{Y}$ to indicate the existence of a $g \in G$ such that $g\mathcal{X} = \mathcal{Y}$. In this case we say that $\mathcal{X}$ and $\mathcal{Y}$ are *isomorphic* and that g is an *isomorphism* of $\mathcal{X}$ onto $\mathcal{Y}$. Equivalently, $\mathcal{X} \cong_G \mathcal{Y}$ if and only if $\mathcal{X}$ and $\mathcal{Y}$ are in the same orbit of G on Ω, where the *orbit* of an object $\mathcal{X}$ under the action of G is the set $G\mathcal{X} = \{g\mathcal{X} : g \in G\}$. An *automorphism* of $\mathcal{X}$ is an isomorphism of $\mathcal{X}$ onto itself. The *automorphism group* $\mathrm{Aut}(\mathcal{X})$ is the subgroup of G consisting of all the automorphisms of $\mathcal{X}$. Equivalently, $\mathrm{Aut}(\mathcal{X})$ is the *stabilizer* $\mathrm{Stab}_G(\mathcal{X}) = \{g \in G : g\mathcal{X} = \mathcal{X}\}$ of $\mathcal{X}$ in G. An (*isomorphism*) *invariant* is a function I of Ω such that $I(\mathcal{X}) = I(\mathcal{Y})$ whenever $\mathcal{X} \cong_G \mathcal{Y}$.

Let $(\Gamma, \cong)$ be the classification problem studied. We say that $(\Gamma, \cong)$ is *represented* by the action of G on Ω if $\Gamma \subseteq \Omega$ and the equivalence relations "$\cong$" and "$\cong_G$" coincide on Γ; that is, for all $\mathcal{X}, \mathcal{Y} \in \Gamma$ it holds that $\mathcal{X} \cong \mathcal{Y}$ if and only if $\mathcal{X} \cong_G \mathcal{Y}$. In what follows we assume that the classification problem studied can be represented by a group action. Accordingly, we drop the subscript G from "$\cong_G$" for notational convenience, whereby it is understood that in what follows "$\cong$" always refers to the equivalence relation induced by the action of G on Ω. We allow the domain Ω to be strictly larger than Γ for the purpose of accommodating the intermediate objects occurring during generation.

Example 4 Let Γ be the set of all 4×4 matrices with entries from $\{0, 1\}$

and row and column sums equal to two. Index the rows and columns of the matrices in Γ by $\{1, 2, 3, 4\}$ and $\{1', 2', 3', 4'\}$, respectively. Then, the classification problem in Example 1 is represented by the action of the direct product $G = \mathrm{Sym}(\{1, 2, 3, 4\}) \times \mathrm{Sym}(\{1', 2', 3', 4'\})$ on Γ by row and column permutation. Formally, for $\mathbf{X} \in \Gamma$ and $(g, g') \in G$, the matrix $(g, g')\mathbf{X}$ is defined by the following rule: for all $i \in \{1, 2, 3, 4\}$ and $j' \in \{1', 2', 3', 4'\}$, the entry at row i, column j' of the matrix $\mathbf{X}$ becomes the entry at row $g(i)$, column $g'(j')$ in the matrix $(g, g')\mathbf{X}$.

3.3.2 Types of Isomorphism Problems

Isomorphism problems have been extensively studied both in the context of specific types of objects and actions—a canonical example being the graph isomorphism problem [4, 74, 88, 117, 206]—and in a more general context with arbitrary permutation groups [21, 145, 146, 147, 154] and even arbitrary equivalence relations [13].

From the viewpoint of classification and group actions, an "isomorphism problem" is usually one of the following four types of problems associated with a finite permutation group G acting on a finite set Ω of combinatorial objects; cf. [21, 130, 145, 146, 147].

the isomorphism problem Given $\mathcal{X}, \mathcal{Y} \in \Omega$, decide whether $\mathcal{X} \cong \mathcal{Y}$.

automorphism group (generators) Given $\mathcal{X} \in \Omega$, compute a set of generators for $\mathrm{Aut}(\mathcal{X})$.

canonical representative Given $\mathcal{X} \in \Omega$, compute an object $\rho(\mathcal{X}) \in \Omega$ such that $\rho(\mathcal{X}) \cong \mathcal{X}$ and for all $\mathcal{X}, \mathcal{Y} \in \Omega$ it holds that $\mathcal{X} \cong \mathcal{Y}$ implies $\rho(\mathcal{X}) = \rho(\mathcal{Y})$. The object $\rho(\mathcal{X})$ is the *canonical representative* of its orbit and the *canonical form* (alternatively, *normal form*) of $\mathcal{X}$. The map $\mathcal{X} \mapsto \rho(\mathcal{X})$ is a *canonical representative map*. A variant of the canonical representative problem is the problem of *testing canonicity*; that is, for a given $\mathcal{X} \in \Omega$ deciding whether $\mathcal{X} = \rho(\mathcal{X})$ relative to some canonical representative map ρ.

canonical labeling Given $\mathcal{X} \in \Omega$, compute a $\kappa(\mathcal{X}) \in G$ such that for all $g \in G$ and $\mathcal{X} \in \Omega$ it holds that $\kappa(g\mathcal{X})g\mathcal{X} = \kappa(\mathcal{X})\mathcal{X}$. The map $\mathcal{X} \mapsto \kappa(\mathcal{X})$ is a *canonical labeling map*. The map $\mathcal{X} \mapsto \kappa(\mathcal{X})\mathcal{X}$ is the associated canonical representative map.

In addition to these problems, it is often necessary to work with permutation groups given by a set of generators; this is usually either because permutation groups are employed internally by an isomorphism algorithm or because the classification approach requires a solution to a problem involving permutation groups; say, computing the automorphism orbits of blocks in a design, where the automorphism group is represented as a permutation group on the points. A recent comprehensive treatment of permutation group algorithms is [220]; other recommended references include [20, 62, 63, 88, 100].

3.3.3 Computing Isomorphism

A long-standing open problem in the theory of computing is whether the isomorphism problem for graphs admits an algorithm that has a worst case running time bounded from above by a polynomial in the size of the input graphs; surveys and key results can be found in [3, 4, 5, 74, 88, 117, 153, 154]. For many central families of combinatorial objects—including 2-designs [42], unrestricted codes, and linear codes (given by a generator matrix) [199]—the isomorphism problem is at least as difficult as the graph isomorphism problem in the sense that a polynomial-time algorithm exists only if there exists a polynomial-time algorithm for the graph isomorphism problem. Thus, in the absence of a polynomial-time algorithm for graph isomorphism, the asymptotic worst case performance of general-purpose isomorphism algorithms remains inherently bad.

Fortunately, backtrack algorithms based on refinement of ordered partitions via invariants exhibit good practical performance on many instances. The software package *nauty* [169] is probably the fastest software implementation for practical isomorphism computations on graphs; a closely related implementation is the package *Groups & Graphs* [118]. The algorithm used by *nauty* is described in detail in [168]; a good exposition is given also in [181]. Similar algorithms are discussed in [119] and [127, Ch. 7]; the main differences between these algorithms are in how node invariants (indicator functions [168]) and discovered automorphisms are used to prune the search tree induced by individualization and refinement. All of the algorithms accept as input a *vertex-colored graph*; that is, a graph with an accompanying ordered partition of the vertices into "color classes". The algorithms output a canonical labeling and automorphism group generators for the vertex-colored graph, where the acting group is the symmetric group on the vertices. Although in practice these algorithms perform quite well, instances requiring exponential time in the size of the input graph are known [22, 181].

There are two standard approaches for computing isomorphism on objects other than graphs: either transform the object into a graph and use an algorithm for graphs, or use an algorithm specifically tailored for the objects (cf. [21]).

Most types of combinatorial objects can be transformed into a graph for purposes of computing isomorphism [86, 180]. Formally, the isomorphism-respecting properties of such transformations into graphs can be analyzed in the setting of (strongly) reconstructible functors; see [4]. The existence of such transformations is often due to the fact that the acting group inducing isomorphism for the objects in question can be concisely encoded as the automorphism group of a graph, after which the relevant structure in the objects can usually be encoded in a straightforward manner by adding edges and vertices. In practice, it is customary to employ vertex-colored graphs because this results in a more compact encoding. Examples of transformations into vertex-colored graphs can be found in [69, 167, 171, 190, 193]. Often the use of vertex invariants (see [169]) more suited for the objects at hand than the standard color-degree invariant is required for good practical performance. Invariants applicable for designs are discussed in [41, 68, 69, 70].

Objects for which tailored isomorphism algorithms have been developed include Steiner triple systems [179] (see also [32, 38, 228]), one-factorizations of connected graphs and complete multigraphs [32], Hadamard matrices [143], Latin squares [19], groups given by Cayley tables (algorithm attributed to Tarjan in [179]), and affine and projective planes [179]. Each of these algorithms is based on the existence of small distinguishing sets of subobjects (say, points of a design or rows of a Hadamard matrix), whose individualization followed by refinement with appropriate invariants produces a unique labeling for the object. Selecting the minimum object (over all labelings induced by small distinguishing sets) gives a canonical representative map. Combined with group-theoretic tools, this approach is generalized in [5] for tournaments, 2-(v, k, λ) designs with small k, λ, and symmetric designs with small λ. As a general technique, individualization and refinement of ordered partitions via invariants [168] is usually straightforward to adapt to a family of objects for which the acting group is a symmetric group or a direct product of symmetric groups; in [173] such an adaptation for designs is reported to produce an order of magnitude performance improvement compared with transforming a design into a graph. See [226] for an analysis of individualization and refinement on strongly regular graphs. An algorithm for linear codes appears in [144].

Techniques for computing isomorphism in the situation where the acting group is an arbitrary permutation group (given by a base and a strong generating set) are developed in [5, 21, 145, 146, 147]. Of these, the methods in [21, 145] are based on “traditional” backtrack search on cosets of a point stabilizer chain (see [20, 220]); the group generated by the automorphisms discovered so far is used to prune the search tree. The partition method developed in [146, 147] introduces partition refinement techniques motivated by [168] to further focus the search. A brief exposition of the partition method occurs also in [220]. The algorithm in [5] (see also [154]) applies a divide-and-conquer strategy based on orbits and systems of imprimitivity in the acting group and its subgroups.

3.4 TECHNIQUES FOR ISOMORPH REJECTION

Isomorph rejection [230] techniques serve two purposes in a classification algorithm. On one hand, to achieve isomorph-free generation for the objects of interest, isomorphic objects must be eliminated from the output of the algorithm. On the other hand, isomorph rejection is employed to eliminate redundant work caused by traversing regions of the search space that are identical for purposes of generation.

Example 5 To provide an example of redundancy, consider the search tree in Example 3. The subtrees rooted at

$$\begin{bmatrix} 1100 \\ 1010 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1100 \\ 0101 \end{bmatrix} \tag{3.1}$$

contain up to isomorphism the same objects of interest. Furthermore, the subtrees are identical in the stronger sense that one can be obtained from the other by permuting the columns of every matrix occurring in a subtree with $(1' \, 2')(3' \, 4')$; cf. Example 4. Thus, it suffices to traverse only one of the subtrees defined by (3.1).

In a general setting, we have an implicit search tree T containing at least one object from every isomorphism class of interest. The goal is to traverse a subtree with as few redundant nodes as can be efficiently detected and eliminated (compared with the cost of a redundant traversal), relative to the constraint that exactly one object from every isomorphism class of interest is output. Other design goals for isomorph rejection include parallelizability—that is, the ability to traverse disjoint subtrees independently of each other—and space-efficiency in terms of objects that need to be stored in memory during a traversal (cf. [61, 172, 170, 205]).

Redundancy is usually detected via isomorphism computations on search tree nodes. For this purpose, we will assume that the action of G on Ω applies to all objects in $V(T)$; that is, $V(T) \subseteq \Omega$. Furthermore, we assume that isomorphism is defined by the action of G on Ω. In practice, this situation is usually not difficult to achieve in a natural way.

Example 6 Consider the group action in Example 4 and the search tree in Example 3. Let Ω consist of all the 4×4 matrices with entries from $\{0, 1, ?\}$, where G acts by permuting the rows and columns as in Example 4. A matrix with $k < 4$ rows in the search tree can now be viewed as the 4×4 matrix where the entries on the last $4 - k$ rows are equal to "?". This extends in a natural way the notion of isomorphism from the objects of interest onto the entire search tree. For example, the nodes in (3.1) are isomorphic, and any isomorphism fixing rows 3 and 4 maps the subtree rooted at one node onto the subtree rooted at the other.

Isomorph rejection techniques now make certain assumptions about the structure of the search tree T in relation to the action of G on Ω. If these assumptions are satisfied, then redundant subtrees can be detected and eliminated via isomorphism computations. The precise form of the isomorphism computations and what is considered redundant depend on the technique. In general, the best isomorph rejection techniques avoid expensive isomorphism computations by taking advantage of the way in which the objects are constructed, whereby expensive computations (such as canonical labeling) are either traded for group-theoretic techniques relying on prescribed groups of automorphisms (cf. [140, 141]) or replaced with lighter computation by means of invariants in the majority of cases (cf. [17, 170, 171]).

The subsequent treatment roughly follows [16, 170] in the division of the techniques into different types.

3.4.1 Recorded Objects

Among the "folklore" techniques for isomorphism rejection is the approach of keeping a record of the isomorphism classes of objects seen so far during traversal of the search tree. If an object is isomorphic to a recorded object, then the subtree rooted at the object is pruned.

The underlying assumption with this isomorph rejection strategy is that the search tree T satisfies the following property:

(3.2) for all $\mathcal{X}, \mathcal{Y} \in V(T)$ and $\mathcal{Z} \in C(\mathcal{X})$ it holds that $\mathcal{X} \cong \mathcal{Y}$ implies there exists a $\mathcal{W} \in C(\mathcal{Y})$ with $\mathcal{Z} \cong \mathcal{W}$.

In essence (3.2) states that isomorphic objects have isomorphic children.

Example 7 The search tree in Example 3 satisfies (3.2).

With this assumption, isomorph-free exhaustive generation can be obtained by keeping a record of the objects encountered so far. Whenever an object $\mathcal{X}$ is encountered, it is tested for isomorphism against the recorded objects. If $\mathcal{X}$ is isomorphic to a recorded object, the subtree rooted at $\mathcal{X}$ is pruned. This approach is presented in Algorithm 3.

```
procedure REC-TRAV($\mathcal{X}$: object)
 1: if there exists a $\mathcal{Z} \in \mathscr{R}$ such that $\mathcal{X} \cong \mathcal{Z}$ then
 2:     return
 3: end if
 4: if $\mathcal{X}$ is an object of interest then
 5:     output $\mathcal{X}$
 6: end if
 7: for all $\mathcal{Y} \in C(\mathcal{X})$ do
 8:     REC-TRAV($\mathcal{Y}$)
 9: end for
10: $\mathscr{R} \leftarrow \mathscr{R} \cup \{\mathcal{X}\}$
end procedure
procedure RECORD-TRAVERSE($T$: search tree)
11: $\mathscr{R} \leftarrow \emptyset$
12: REC-TRAV($r(T)$)
end procedure
```

Algorithm 3: Isomorph rejection via recorded objects

Theorem 8 *Let T be a search tree that satisfies* (3.2). *If Algorithm* 3 *is invoked with input T, then a unique object is output from every isomorphism class of interest in $V(T)$.*

In practice Algorithm 3 is often implemented using a canonical representative map and a hash table (or some other data structure that allows fast searching from a large collection of objects; see [44, 115]).

Example 9 Figure 3.2 shows a subtree of the search tree in Example 3 traversed using isomorph rejection via recorded objects. Nodes indicated with "×" are isomorphic to objects encountered earlier. Note that here the subtree traversed depends on the order of traversal for the children of a node.

Isomorph rejection via recorded objects is sufficient for generating many families of combinatorial objects. Indeed, because a canonical representative map for the objects occurring in the search is often easily obtainable via transformation into graphs and isomorphism computations on graphs, this

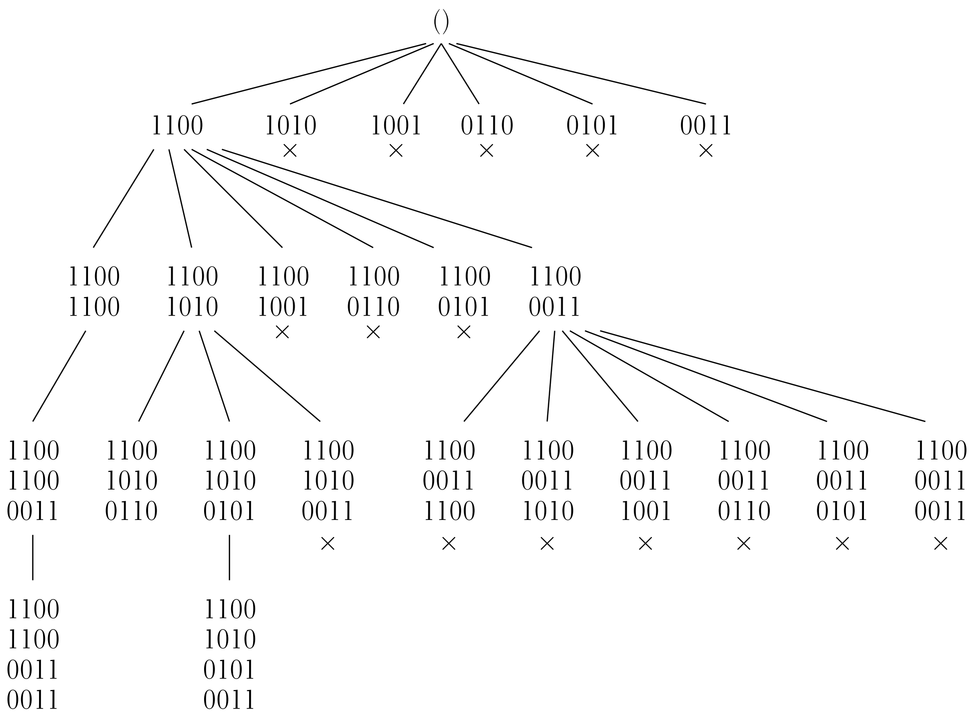

Figure 3.2: A search tree with isomorph rejection

approach is fast to implement and arguably less error-prone compared with the more advanced techniques.

There are at least three difficulties with isomorph rejection via recorded objects. Perhaps the most fundamental difficulty is the need to store the objects encountered. Especially when the number of nonisomorphic intermediate objects is large, the available storage space can quickly run out. The second difficulty is that the search cannot be easily parallelized because a search process must somehow communicate with the other search processes to find out whether an object has been already encountered. The third difficulty is that computing the canonical representative of every object encountered can be very expensive compared with the use of invariants in the more advanced techniques.

3.4.2 Generation by Canonical Representatives

Another possibility to perform isomorph rejection is to select a canonical representative for every isomorphism class, and then generate precisely these representatives, whereby it is required that the canonical representatives form a rooted subtree T_c of the search tree T with $r(T) = r(T_c)$. This is of course somewhat difficult to achieve for a nontrivial group action inducing isomorphism; in practice, the canonical representatives are extremal elements of orbits relative to a (lexicographic) order on Ω. Thus, generation by canonical representatives is often called *orderly* generation (cf. [205]), although the term is occasionally used for a larger family of algorithms (cf. [170, 210]). Generation by canonical representatives was introduced independently by Faradžev [61] and Read [205].

Formally, let $\rho : \Omega \to \Omega$ be a canonical representative map for the action of G on Ω. For completeness, it is assumed that:

(3.3) the canonical representative of every isomorphism class of interest occurs in the search tree.

Furthermore:

(3.4) the parent of every nonroot canonical representative occurring in the search tree is a canonical representative.

It follows immediately from (3.3) and (3.4) that it suffices to traverse only the canonical representatives to achieve isomorph-free exhaustive generation. This approach is formulated in Algorithm 4.

```
procedure CR-TRAV($\mathcal{X}$: object)
 1: if $\mathcal{X} \neq \rho(\mathcal{X})$ then
 2:     return
 3: end if
 4: if $\mathcal{X}$ is an object of interest then
 5:     output $\mathcal{X}$
 6: end if
 7: for all $\mathcal{Y} \in C(\mathcal{X})$ do
 8:     CR-TRAV($\mathcal{Y}$)
 9: end for
end procedure
procedure CANREP-TRAVERSE($T$: search tree)
10: CR-TRAV($r(T)$)
end procedure
```

Algorithm 4: Generation by canonical representatives

The following example illustrates generation by canonical representatives based on a lexicographic order. A general framework for group actions and lexicographic order appears in [61]; a more axiomatic framework is given in [205].

We first introduce an appropriate canonical representative map ρ for the action inducing isomorphism.

Example 10 Recall the situation in Example 6. Let $\mathbf{X}$ be a 4×4 matrix with entries from $\{0, 1, ?\}$. Associate with $\mathbf{X}$ the word $w(\mathbf{X})$ obtained by concatenating the rows of $\mathbf{X}$ in order from first to last. For example,

$$\mathbf{X} = \begin{bmatrix} 1\,1\,0\,0 \\ 1\,0\,1\,0 \\ ?\,?\,?\,? \\ ?\,?\,?\,? \end{bmatrix}, \qquad w(\mathbf{X}) = 11001010????????.$$

Let the words $w(\mathbf{X})$ be ordered lexicographically, where the alphabet is ordered by $? < 0 < 1$. Define an order for the matrices by $\mathbf{X} < \mathbf{Y}$ if and only if $w(\mathbf{X}) < w(\mathbf{Y})$. With respect to this order, let $\rho(\mathbf{X})$ be the maximum matrix obtainable from $\mathbf{X}$ by permuting the rows and columns.

Example 11 With some straightforward effort it can be checked that the search tree in Example 3 satisfies (3.3) and (3.4) with respect to the canonical representative map defined in Example 10. Here a matrix with $k < 4$ rows should be viewed as the 4×4 matrix where the entries on the last $4 - k$ rows are equal to "?". To show that $\mathbf{X}$ is canonical only if $p(\mathbf{X})$ is canonical, observe that a permutation of the rows and columns establishing noncanonicity of $p(\mathbf{X})$ also suffices to establish noncanonicity of $\mathbf{X}$ due to the lexicographic order employed.

With respect to this choice of canonical representatives, the subtree traversed by Algorithm 4 is identical to the tree depicted in Fig. 3.2, where "$\times$" now marks nodes that are not canonical.

Algorithms based on generation by canonical representatives have the convenient property that no isomorphism tests between different nodes of the search tree are required. The decision whether to accept or reject a node can be made locally, based on a canonicity test procedure that determines whether $\mathcal{X} = \rho(\mathcal{X})$ for the current node $\mathcal{X}$. Thus, the search can be efficiently parallelized because disjoint subtrees can be searched independently of each other. Furthermore, no objects need to be stored in memory for purposes of isomorph rejection.

A basic advantage with orderly generation is that it is often possible to exploit the properties of order-extremal objects of interest in pruning subtrees that cannot contain such an object.

Example 12 Let $\mathbf{X}$ be a 4×4 matrix with entries from $\{0, 1\}$ and row and column sums equal to two. Furthermore, suppose that $\mathbf{X}$ is the lexicographic maximum relative to permutation of the rows and the columns; in other words, $\rho(\mathbf{X}) = \mathbf{X}$ in Example 10. It follows from the properties of lexicographic order that the matrix $\mathbf{X}$ must have the form

$$\begin{bmatrix} 1\,1\,0\,0 \\ 1\,a\,b\,c \\ 0\,d\,e\,f \\ 0\,g\,h\,i \end{bmatrix}.$$

Namely, a matrix not of this form can be transformed by permutation of the rows and columns to a lexicographically greater matrix of this form. This observation can now be applied to prune the search tree in Example 3. For example, no descendant of the canonical matrix

$$\begin{bmatrix} 1\,1\,0\,0 \\ 0\,0\,1\,1 \\ ?\,?\,?\,? \\ ?\,?\,?\,? \end{bmatrix}$$

is a canonical matrix of interest. Thus, the subtree can be pruned.

Further examples can be found in [51, 54, 171, 177, 178, 217, 218, 225] and [P1, P4]. It should be noted that this type of order-based constraints on partial solutions can to some extent be implemented through the use of invariants in the other isomorph rejection techniques, but this is rather more tedious.

The main drawback with generation by canonical representatives is that testing canonicity relative to a lexicographic order is often computationally expensive. The typical approach for testing canonicity is to employ backtrack search on cosets of a point stabilizer chain in G to verify that $g\mathcal{X} \leq \mathcal{X}$ for all $g \in G$. Lexicographic order and discovered automorphisms can be employed to prune the associated search tree on cosets. Also the canonicity of $p(\mathcal{X})$ can be exploited to restrict the search. In many cases a useful heuristic observation is that a $g \in G$ with $g\mathcal{X} > \mathcal{X}$ is likely to establish $g\mathcal{Y} > \mathcal{Y}$ for a sibling $\mathcal{Y}$ of $\mathcal{X}$ as well (cf. [171])—in [49, 50] this observation is developed into a back-jumping strategy for the backtrack search that generates the children of a node.

3.4.3 Generation by Canonical Augmentation

Introduced by McKay [170], generation by canonical augmentation requires that an object is generated "in a canonical way", as opposed to generation by canonical representatives, which requires the object itself be canonical. The presentation that follows differs somewhat from the presentation in [170], but the central ideas are the same.

In terms of a search tree T, every nonroot object $\mathcal{Y}$ has a parent object $p(\mathcal{Y})$ from which it has been generated; consider for example the search tree in Example 3. Thus, in an abstract setting, the ordered pair $(\mathcal{Y}, p(\mathcal{Y}))$ can be viewed as capturing the augmentation by which $\mathcal{Y}$ is generated from $p(\mathcal{Y})$.

Formally, an *augmentation* is an ordered pair $(\mathcal{X}, \mathcal{Z}) \in \Omega \times \Omega$. For $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{W} \in \Omega$ we write $(\mathcal{X}, \mathcal{Z}) \cong (\mathcal{Y}, \mathcal{W})$ to indicate that there exists a $g \in G$ such that $g\mathcal{X} = \mathcal{Y}$ and $g\mathcal{Z} = \mathcal{W}$. In particular, $(\mathcal{X}, \mathcal{Z}) \cong (\mathcal{Y}, \mathcal{W})$ implies both $\mathcal{X} \cong \mathcal{Y}$ and $\mathcal{Z} \cong \mathcal{W}$, but the converse need not necessarily hold.

Generation by canonical augmentation requires that we associate with every isomorphism class of objects an augmentation by which objects in that class must be generated. Let $m : \Omega \to \Omega$ be a function that satisfies:

(3.5) for all $\mathcal{X}, \mathcal{Y} \in \Omega$ it holds that $\mathcal{X} \cong \mathcal{Y}$ implies $(\mathcal{X}, m(\mathcal{X})) \cong (\mathcal{Y}, m(\mathcal{Y}))$.

The ordered pair $(\mathcal{X}, m(\mathcal{X}))$ is the canonical augmentation associated with $\mathcal{X}$. Requirement (3.5) guarantees that the canonical augmentation is independent of the isomorphism class representative $\mathcal{X}$. We say that a nonroot object $\mathcal{Y} \in V(T)$ in the search tree is *generated by canonical augmentation* if

$$(\mathcal{Y}, m(\mathcal{Y})) \cong (\mathcal{Y}, p(\mathcal{Y})). \tag{3.6}$$

The *canonical parent* object $m(\mathcal{X})$ associated with an object $\mathcal{X}$ is typically obtained by individualizing a subobject of $\mathcal{X}$. This can be performed so that (3.5) holds with the help of a canonical labeling map κ for the action of G on Ω. Namely, given $\mathcal{X}$, first compute the canonical representative $\rho(\mathcal{X}) = \kappa(\mathcal{X})\mathcal{X}$. Then, select any subobject $\mathcal{Z} \in \Omega$ occurring in $\rho(\mathcal{X})$ so that the selection depends only on $\rho(\mathcal{X})$. Finally, put $m(\mathcal{X}) = \kappa(\mathcal{X})^{-1}\mathcal{Z}$. By definition of a canonical labeling map, $\kappa(\mathcal{Y})^{-1}\kappa(\mathcal{X})$ is an isomorphism establishing (3.5) for any two isomorphic objects $\mathcal{X} \cong \mathcal{Y}$.

Example 13 Recall the canonical representative map ρ from Example 10. Form a canonical labeling map κ by associating with every matrix $\mathbf{X}$ an isomorphism $\kappa(\mathbf{X})$ taking $\mathbf{X}$ to $\rho(\mathbf{X})$. For a given matrix $\mathbf{X}$, let the subobject $\mathbf{Z}$ associated with $\rho(\mathbf{X})$ be the matrix obtained by transforming the last non-"?" row in $\rho(\mathbf{X})$ into a "?"-row. (If all rows contain "?", then let $\mathbf{Z} = \rho(\mathbf{X})$.) Put $m(\mathbf{X}) = \kappa^{-1}(\mathbf{X})\mathbf{Z}$.

The function m can be viewed as associating with every object $\mathcal{X}$ a sequence of subobjects

$$\mathcal{X}, m(\mathcal{X}), m(m(\mathcal{X})), m(m(m(\mathcal{X}))), \ldots$$

from which $\mathcal{X}$ must be generated. Because of (3.5), this sequence is independent of the isomorphism class representatives chosen. Thus, the sequence defines a "canonical construction path" for the object $\mathcal{X}$ on the level of isomorphism classes. Traversing the search tree T can now be viewed as proceeding along the sequence in the reverse direction, where each step from subobject to object must satisfy (3.6).

Obviously, certain structure must be assumed for the search tree T to achieve exhaustive generation. The following axioms are not the weakest possible, but rather have been chosen to achieve a succinct exposition without sacrificing too much generality. For a different axiomatization, see [170].

First, for every isomorphism class occurring in T, there exists a node that is accepted in the test (3.6):

(3.7) for all nonroot $\mathcal{X} \in V(T)$, there exists a $\mathcal{Y} \in V(T)$ such that $\mathcal{X} \cong \mathcal{Y}$ and $(\mathcal{Y}, m(\mathcal{Y})) \cong (\mathcal{Y}, p(\mathcal{Y}))$.

Second, isomorphic nodes in T must have isomorphic children such that an isomorphism applies also to the parent nodes:

(3.8) for all $\mathcal{X}, \mathcal{Y} \in V(T)$ and $\mathcal{Z} \in C(\mathcal{X})$ it holds that $\mathcal{X} \cong \mathcal{Y}$ implies there exists a $\mathcal{W} \in C(\mathcal{Y})$ with $(\mathcal{Z}, \mathcal{X}) \cong (\mathcal{W}, \mathcal{Y})$.

Third, isomorphic nodes must occur at the same level of T:

(3.9) for all $\mathcal{X}, \mathcal{Y} \in V(T)$ it holds that $\mathcal{X} \cong \mathcal{Y}$ implies $\mathcal{X}$ and $\mathcal{Y}$ occur at the same level in T.

Example 14 With some straightforward effort it can be checked that the search tree in Example 3, the group action in Example 6, and the function m in Example 13 satisfy (3.7), (3.8), and (3.9).

Isomorph rejection using the test (3.6) is now based on the following observations. By (3.5), two isomorphic objects $\mathcal{Y}_1, \mathcal{Y}_2$ are both accepted in the test (3.6) only if

(3.10) $$(\mathcal{Y}_1, p(\mathcal{Y}_1)) \cong (\mathcal{Y}_2, p(\mathcal{Y}_2));$$

in particular, $p(\mathcal{Y}_1) \cong p(\mathcal{Y}_2)$. If complete isomorph rejection has been performed for all proper ancestors of $\mathcal{Y}_1, \mathcal{Y}_2$, then $p(\mathcal{Y}_1) = p(\mathcal{Y}_2) = \mathcal{X}$. Consequently, by (3.10) there exists an $a \in \mathrm{Aut}(\mathcal{X})$ such that $a\mathcal{Y}_1 = \mathcal{Y}_2$. Thus, to

```
procedure CA-TRAV($\mathcal{X}$: object)
 1: if $\mathcal{X}$ is an object of interest then
 2:     output $\mathcal{X}$
 3: end if
 4: for all $\mathscr{Y} \in \{C(\mathcal{X}) \cap \{a\mathcal{Y} : a \in \mathrm{Aut}(\mathcal{X})\} : \mathcal{Y} \in C(\mathcal{X})\}$ do
 5:     select any $\mathcal{Y} \in \mathscr{Y}$
 6:     if $(\mathcal{Y}, p(\mathcal{Y})) \cong (\mathcal{Y}, m(\mathcal{Y}))$ then
 7:         CA-TRAV($\mathcal{Y}$)
 8:     end if
 9: end for
end procedure
procedure CANAUG-TRAVERSE($T$: search tree)
10: CA-TRAV($r(T)$)
end procedure
```

Algorithm 5: Generation by canonical augmentation

achieve complete isomorph rejection, it suffices to reject $\mathrm{Aut}(\mathcal{X})$-isomorphs among the children $C(\mathcal{X})$.

Algorithm 5 is a traversal algorithm based on generation by canonical augmentation with automorphism pruning.

The following theorem is analogous to [170, Theorem 1].

Theorem 15 *Let T be a search tree that satisfies* (3.7), (3.8), *and* (3.9). *If Algorithm 5 is invoked with input T, then every isomorphism class of objects occurring in $V(T)$ contains a unique object $\mathcal{Y}$ such that the procedure* CA-TRAV *is invoked with input $\mathcal{Y}$.*

Proof. To establish uniqueness, we proceed by induction on the levels of T. The base case holds by (3.9). For the inductive step, suppose the claim (uniqueness) holds at level $\ell \geq 0$ in T. By (3.9), any two isomorphic objects in T must lie at the same level. Let $\mathcal{Y}_1, \mathcal{Y}_2 \in V(T)$ be objects at level $\ell + 1$ such that $\mathcal{Y}_1 \cong \mathcal{Y}_2$ and CA-TRAV is invoked with input $\mathcal{Y}_1, \mathcal{Y}_2$. Thus, we must have $(\mathcal{Y}_i, p(\mathcal{Y}_i)) \cong (\mathcal{Y}_i, m(\mathcal{Y}_i))$ for $i \in \{1, 2\}$. Consequently, $\mathcal{Y}_1 \cong \mathcal{Y}_2$ and (3.5) imply $(\mathcal{Y}_1, p(\mathcal{Y}_1)) \cong (\mathcal{Y}_2, p(\mathcal{Y}_2))$. Because $p(\mathcal{Y}_1), p(\mathcal{Y}_2)$ are at level ℓ, we must have $p(\mathcal{Y}_1) = p(\mathcal{Y}_2) = \mathcal{X}$ by the inductive hypothesis. By $(\mathcal{Y}_1, p(\mathcal{Y}_1)) \cong (\mathcal{Y}_2, p(\mathcal{Y}_2))$ there exists an automorphism $a \in \mathrm{Aut}(\mathcal{X})$ such that $a\mathcal{Y}_1 = \mathcal{Y}_2$. Thus, $\mathcal{Y}_1 = \mathcal{Y}_2$ by the structure of the algorithm.

It remains to establish existence. Let $\mathcal{Z} \in V(T)$. We show that there exists a $\mathcal{Y} \in V(T)$ such that $\mathcal{Y} \cong \mathcal{Z}$ and CA-TRAV is invoked with input $\mathcal{Y}$. We proceed by induction on the level of $\mathcal{Z}$. The base case follows from (3.9) and the initial invocation CA-TRAV($r(T)$). For the inductive step, suppose that the claim (existence) holds at level $\ell \geq 0$ in T. Let $\mathcal{Z}$ occur on level $\ell + 1$. By (3.7), there exists a node $\mathcal{W} \cong \mathcal{Z}$ such that $(\mathcal{W}, m(\mathcal{W})) \cong (\mathcal{W}, p(\mathcal{W}))$. By (3.9), $p(\mathcal{W})$ occurs at level ℓ because $\mathcal{Z}$ and hence $\mathcal{W}$ occur at level $\ell + 1$. Thus, by the inductive hypothesis, the procedure CA-TRAV is invoked at least once with input $\mathcal{X}$ such that $\mathcal{X} \cong p(\mathcal{W})$. It follows from (3.8) that there exists a $\mathcal{Y} \in C(\mathcal{X})$ such that $(\mathcal{Y}, \mathcal{X}) \cong (\mathcal{W}, p(\mathcal{W}))$. Let $a \in \mathrm{Aut}(\mathcal{X})$ such that $a\mathcal{Y} \in C(\mathcal{X})$. Clearly, $p(a\mathcal{Y}) = \mathcal{X} = p(\mathcal{Y})$ and $(a\mathcal{Y}, \mathcal{X}) \cong (\mathcal{Y}, \mathcal{X})$. Furthermore, (3.5) implies $(a\mathcal{Y}, m(a\mathcal{Y})) \cong (\mathcal{Y}, m(\mathcal{Y})) \cong (\mathcal{W}, m(\mathcal{W})) \cong$

$(\mathcal{W}, p(\mathcal{W})) \cong (\mathcal{Y}, p(\mathcal{Y})) \cong (a\mathcal{Y}, p(a\mathcal{Y}))$. Thus, $a\mathcal{Y}$ passes the test (3.6) for all applicable choices of $a \in \mathrm{Aut}(\mathcal{X})$. It follows that CA-TRAV is invoked with an input isomorphic to $\mathcal{Y} \cong \mathcal{W} \cong \mathcal{Z}$. □

Example 16 Consider the search tree in Example 3 and Algorithm 5. Suppose the function m from Example 13 is used. If on line 5 of the algorithm the lexicographic maximum object is selected from every $\mathrm{Aut}(\mathbf{X})$-orbit on $C(\mathbf{X})$, then the subtree traversed by the algorithm is identical to the earlier tree depicted in Fig. 3.2, where "$\times$" now marks nodes that fail the test (3.6) or are not maximal in their respective $\mathrm{Aut}(\mathbf{X})$-orbits.

Since $\mathrm{Aut}(\mathcal{X})$-isomorphs are also G-isomorphs, the $\mathrm{Aut}(\mathcal{X})$-isomorph rejection performed on lines 4 and 5 of Algorithm 5 can be replaced with traditional isomorph rejection among those children that are accepted in the test (3.6); cf. Procedure `scan2` in [170].

A fundamental advantage in an algorithm based on canonical augmentation is that it is often possible to use cheap invariants to reject or accept an object $\mathcal{Y}$ in the test (3.6). An expensive isomorphism computation is required only when the invariants fail. Examples on the use of invariants appear in [17, 170, 171, 210] and [P2, P3, P5].

Generation by canonical augmentation can be efficiently parallelized because the test (3.6) depends only on the current object $\mathcal{Y}$ and its parent $p(\mathcal{Y})$. Furthermore, the rejection of $\mathrm{Aut}(\mathcal{X})$-isomorphs is restricted to the children $C(\mathcal{X})$, so knowledge of objects encountered at other search tree nodes is not required. The memory-efficiency depends in general on the strategy chosen to reject the $\mathrm{Aut}(\mathcal{X})$-isomorphs. Often rejection via recorded objects suffices. Another possibility is to use a canonical representative map for the induced action of $\mathrm{Aut}(\mathcal{X})$ on Ω to reject children that are not canonical; here it is assumed that for all $\mathcal{Y} \in C(\mathcal{X})$ it holds that $\rho_{\mathrm{Aut}(\mathcal{X})}(\mathcal{Y}) \in C(\mathcal{X})$.

The main drawback with generation by canonical augmentation is that it is usually more laborious to implement than the previous techniques.

3.4.4 Homomorphisms of Group Actions and Localization

Kerber and Laue together with collaborators have extensively studied the use of homomorphisms of group actions and group-theoretic localization in classification [107, 108, 139, 140, 141]. This line of research has been particularly successful in the construction and classification of t-designs admitting a large prescribed group of automorphisms [9, 10, 11, 140, 215] via the Kramer-Mesner method (Sect. 4.3.1). Other successful applications include classification of chemical isomers [80] (see also [16]) and graphs with prescribed degree sequences [81].

Let G be a finite group that acts on two finite sets Ω and Π. A *homomorphism of group actions* is a map $\varphi : \Omega \to \Pi$ such that $\varphi(g\mathcal{X}) = g\varphi(\mathcal{X})$ for all $g \in G$ and $\mathcal{X} \in \Omega$. Provided that suitable homomorphisms are available, a classification problem (viewed as the problem of constructing representatives of orbits for a group action) can be solved step by step along a sequence of homomorphisms, where each step either lifts a set of orbit representatives from the image Π to the domain Ω, or (surjectively) projects from the domain Ω to the image Π; cf. [139, 140].

Basic homomorphisms of group actions are the map $\mathcal{X} \mapsto \mathrm{Stab}_G(\mathcal{X})$ taking an object $\mathcal{X} \in \Omega$ to its stabilizer (G acts on its subgroups by conjugation), the bijection $g\mathcal{X} \mapsto g\mathrm{Stab}_G(\mathcal{X})$ taking an orbit element $g\mathcal{X} \in G\mathcal{X}$ to a left coset of the stabilizer (G acts on left cosets of its subgroups by left multiplication), and the map $gH \mapsto gK$ taking a left coset onto a (larger) left coset, $H \leq K \leq G$. Also important are projection homomorphisms of the form $\pi_1 : (\mathcal{Y}, \mathcal{X}) \mapsto \mathcal{Y}$ and $\pi_2 : (\mathcal{Y}, \mathcal{X}) \mapsto \mathcal{X}$ (G acts elementwise on objects in an ordered pair).

In many cases these homomorphisms can be used to transform a problem involving an "external" action of a group G on a set Ω into a problem involving a "local" action of G on subgroups or cosets of subgroups of a related group. A good example in this respect is the algorithm *Leiterspiel* [213, 215] for computing orbit representatives for the elementwise action of a permutation group $G \leq \mathrm{Sym}(\Sigma)$ on the set of all k-subsets of a finite set Σ. The algorithm is based on the observation that an equivalent "local" problem in terms of the group $\mathrm{Sym}(\Sigma)$ is to compute representatives of orbits for the action of G on the left cosets $\mathrm{Sym}(\Sigma)/\mathrm{Stab}_{\mathrm{Sym}(\Sigma)}(E)$ by left multiplication, where $E \subseteq \Sigma$ is an arbitrary k-subset. This representative problem is then solved via homomorphisms by varying the right-hand side group along a sequence of subgroups $\mathrm{Sym}(\Sigma) = H_0, H_1, \ldots, H_{m-1}, H_m = \mathrm{Stab}_{\mathrm{Sym}(\Sigma)}(E)$ such that for all $1 \leq i \leq m$ either $H_{i-1} \leq H_i$ (projection is applied) or $H_{i-1} \geq H_i$ (lifting to preimage is applied).

Group-theoretic techniques can also be used to solve isomorphism problems for designs that are too large to handle with algorithms based on backtracking, but are known to admit a select group of automorphisms (such as a projective or affine linear group) [141]; see also [82, 140, 215]. A trivial example in this respect is a group of automorphisms $H \leq G$ that is equal to its normalizer $N_G(H) = \{g \in G : gHg^{-1} = H\}$ in G—it is easy to check that any two objects with $\mathrm{Aut}(\mathcal{X}) = \mathrm{Aut}(\mathcal{Y}) = H = N_G(H)$ satisfy either $\mathcal{X} = \mathcal{Y}$ or $\mathcal{X} \not\cong \mathcal{Y}$.

Finally, we remark that generation by canonical augmentation is closely related with techniques based on homomorphisms of group actions (cf. [140]). One step of generation by canonical augmentation can be viewed as a sequence of two projection homomorphisms $\Phi \xleftarrow{\pi_2} \Lambda \xrightarrow{\pi_1} \Psi$, where $\Lambda \subseteq \Psi \times \Phi$ and π_1 is surjective. The homomorphism $\pi_2 : (\mathcal{Y}, \mathcal{X}) \mapsto \mathcal{X}$ induces a lifting step from objects in Φ to the augmentations in Λ, and $\pi_1 : (\mathcal{Y}, \mathcal{X}) \mapsto \mathcal{Y}$ induces a projection step from Λ to the objects in Ψ. From this perspective the key idea in generation by canonical augmentation lies in the implementation of the projection step, where each object $\mathcal{Y} \in \Psi$ is associated with an $\mathrm{Aut}(\mathcal{Y})$-orbit of augmentations from which $\mathcal{Y}$ is required to originate. In more precise terms, let μ associate with every $\mathcal{Y} \in \Psi$ an $\mathrm{Aut}(\mathcal{Y})$-orbit $\mu(\mathcal{Y}) \subseteq \pi_1^{-1}(\mathcal{Y})$ such that $\mu(g\mathcal{Y}) = \{(g\mathcal{Y}, g\mathcal{X}) : (\mathcal{Y}, \mathcal{X}) \in \mu(\mathcal{Y})\}$ for all $g \in G$. When projecting from Λ to Ψ, a projection $\pi_1(\mathcal{Y}, \mathcal{X}) = \mathcal{Y}$ is accepted if and only if $(\mathcal{Y}, \mathcal{X}) \in \mu(\mathcal{Y})$. Equivalently, $\pi_1(\mathcal{Y}, \mathcal{X}) = \mathcal{Y}$ is accepted if and only if $(\mathcal{Y}, m(\mathcal{Y})) \cong (\mathcal{Y}, \mathcal{X})$ for any $(\mathcal{Y}, m(\mathcal{Y})) \in \mu(\mathcal{Y})$; cf. (3.6).

3.5 CORRECTNESS

Compared with conventional mathematical arguments, computational results are arguably more prone to errors. Namely, even if the classification algorithm is correct, it may be that the computed result is incorrect due to hardware or software errors, where the latter includes errors not only in the algorithm implementation itself, but also errors in the system programs such as the compiler, the standard libraries, and the operating system kernel. With such sources of error, it is obvious that a computed result cannot be trusted with absolute certainty. However, confidence in a computed result can be increased by employing techniques for detecting errors. An extensive discussion of errors and remedies appears in [128]. Examples of classification results in which meticulous attention has been paid to correctness include [54, 171, 173].

Arguably the two main tools for detecting errors are *double checking the result* and *consistency checking*.

Double checking the result amounts to obtaining the same classification in two independent ways, preferably with different algorithmic approaches and with different software development tools and/or computer architectures. Double checking the result was employed in the context of [P1] (partial solutions with five rows), [P2] (the seeds and the STS(19)s with a nontrivial automorphism group; cf. [P5]), [P3] (one-factorizations with $k \leq 6$), and [P4] (2-$(13, 4, 3)$ near resolutions).

Consistency checking amounts to identifying certain properties satisfied by a correct algorithm implementation and/or classification, and then verifying that these properties hold. A common approach is to employ *double counting*, whereby a quantity of interest is computed in two (essentially) independent ways, and the results are tested for equality. The orbit-stabilizer theorem ($|G\mathcal{X}| \cdot |\mathrm{Stab}_G(\mathcal{X})| = |G|$) can often be used to obtain double counting consistency checks on a backtrack algorithm with isomorph rejection; see [133] for a detailed discussion. Another possibility for consistency checking isomorph rejection is to employ hash accumulators that record the structures encountered before and after isomorph rejection tests; see [P5].

Consistency checking was employed in the context of [P2] (double counting check for complete classification), [P3] (double counting check for $k = 10, 11$), and [P5] (double counting check for complete classification, hash accumulator -based check for seed generation).

4 ALGORITHMS FOR CLASSIFICATION OF DESIGNS AND CODES

This chapter surveys the literature on classification algorithms for designs and codes. The focus is on techniques that produce a complete classification for given parameters, but also classification subject to a prescribed group of automorphisms is discussed to place the results in [P5] into context.

4.1 DESIGNS AND ERROR-CORRECTING CODES

This section briefly reviews the standard definitions for t-(v, k, λ) designs and unrestricted error-correcting codes. In what follows also other types of designs and codes are discussed to some extent; in this case the relevant standard definitions can be found in the articles surveyed or from the following references. Textbooks on designs and codes include [24, 26, 90, 150, 151, 201]. More extensive treatments can be found in [8, 30, 34, 77, 157, 202].

4.1.1 Designs

An *incidence structure* is a triple $(P, \mathcal{B}, I)$, where P and $\mathcal{B}$ are finite sets and $I \subseteq P \times \mathcal{B}$. The elements of P are called *points*, the elements of $\mathcal{B}$ are called *blocks*, and I is the *incidence relation*. A point $x \in P$ is *incident* to a block $B \in \mathcal{B}$ if $(x, B) \in I$. Similarly, a subset $W \subseteq P$ is incident to a block B if $(x, B) \in I$ for all $x \in W$. An incidence structure is *simple* if the set of incident points $\{p \in P : (p, B) \in I\}$ is unique to every block B; otherwise the incidence structure is said to have *repeated blocks*.

Two incidence structures, $\mathcal{X}$ and $\mathcal{Y}$, are *isomorphic* if there exist bijections $f_P : P(\mathcal{X}) \to P(\mathcal{Y})$ and $f_{\mathcal{B}} : \mathcal{B}(\mathcal{X}) \to \mathcal{B}(\mathcal{Y})$ such that for all $x \in P(\mathcal{X})$ and $B \in \mathcal{B}(\mathcal{X})$ it holds that $(x, B) \in I(\mathcal{X})$ if and only if $(f_P(x), f_{\mathcal{B}}(B)) \in I(\mathcal{Y})$. Such a pair of bijections $f = (f_P, f_{\mathcal{B}})$ is an *isomorphism* of $\mathcal{X}$ onto $\mathcal{Y}$. An *automorphism* of $\mathcal{X}$ is an isomorphism of $\mathcal{X}$ onto itself. The automorphism group $\mathrm{Aut}(\mathcal{X})$ is the group formed by all automorphisms of $\mathcal{X}$ with composition of mappings as the group operation.

An *incidence matrix* of an incidence structure $\mathcal{X}$ is an integer matrix $\mathbf{N} = (n_{xB})$ with rows and columns indexed by the points and blocks, respectively, such that for all $x \in P(\mathcal{X})$ and $B \in \mathcal{B}(\mathcal{X})$,

$$n_{xB} = \begin{cases} 1 & \text{if } (x, B) \in I(\mathcal{X})\text{; and} \\ 0 & \text{if } (x, B) \notin I(\mathcal{X}). \end{cases}$$

A *resolution* of an incidence structure $\mathcal{X}$ is an ordered pair $(\mathcal{X}, \mathcal{R})$, where $\mathcal{R}$ is a partition of $\mathcal{B}(\mathcal{X})$ into *parallel classes* such that every point $x \in P(\mathcal{X})$ is incident to exactly one block from every parallel class. An incidence structure is *resolvable* if it admits a resolution. Two resolutions are *isomorphic* if there exists an isomorphism $f = (f_P, f_{\mathcal{B}})$ of the incidence structures such that $f_{\mathcal{B}}$ maps one partition into parallel classes onto the other.

Let t, v, k, and λ be positive integers with $v \geq k \geq t$. A t-(v, k, λ) *design* is an incidence structure over v points such that every block is incident to exactly k points and every t-subset of points is incident to exactly λ blocks.

A design with $t = 2$ is called a (*balanced incomplete*) *block design* with parameters (v, k, λ). A standard double counting argument shows that every point of a block design is incident to r blocks and that the number of blocks is b, where

$$\lambda(v-1) = r(k-1), \qquad vr = bk. \tag{4.1}$$

Some special families of designs are as follows. A design with $\lambda = 1$ is called a *Steiner system* $S(t, k, v)$. A *Steiner triple system*—briefly, an STS(v)—is an $S(2, 3, v)$. A 2-$(n^2 + n + 1, n + 1, 1)$ design is a *projective plane* of order n. A 2-$(n^2, n, 1)$ design is an *affine plane* of order n.

4.1.2 Codes and Resolutions of Designs

Let $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and denote by $\mathbb{Z}_q^n$ the set of all words of length n over the alphabet $\mathbb{Z}_q$. A q-ary *code* of *length* n is a subset $C \subseteq \mathbb{Z}_q^n$. The *cardinality* $|C|$ of a code is the number of words in it.

For a word $x = x_1x_2\cdots x_n \in \mathbb{Z}_q^n$ and $1 \le j \le n$, we say that x_j is the symbol at *coordinate* (alternatively, *position*) j. The *distance* $d(x, y)$ between two words $x, y \in \mathbb{Z}_q^n$ is the number of positions in which they differ; that is, $d(x, y) = |\{j : x_j \ne y_j\}|$. The *minimum distance* of a code C with $|C| \ge 2$ is $d(C) = \min\{d(x, y) : x, y \in C,\ x \ne y\}$. An $(n, M, d)_q$ code is a q-ary code of length n, cardinality M, and minimum distance d. A code is *equidistant* if $d(C) = d(x, y)$ for all distinct $x, y \in C$. A q-ary code is *equireplicate* if q divides $|C|$ and every symbol occurs exactly $|C|/q$ times in every coordinate of the code.

Two codes, $C_1 \subseteq \mathbb{Z}_q^n$ and $C_2 \subseteq \mathbb{Z}_q^n$, are *equivalent* if C_1 can be transformed into C_2 by a permutation of the coordinates and by an independent permutation of the symbols $\mathbb{Z}_q$ in each coordinate. In terms of a group action, two codes are equivalent if they are on the same orbit of the product action (see for example [25, 55]) of the wreath product $\mathrm{Sym}(\mathbb{Z}_q) \wr \mathrm{Sym}(\{1, 2, \ldots, n\})$ on $\mathbb{Z}_q^n$; in what follows we use the abbreviated notation $S_q \wr S_n$. The automorphism group of a code is the stabilizer of the code with respect to this action.

The following correspondence between codes and resolutions of 2-(v, k, λ) designs, $k < v$, discovered by Semakov and Zinov'ev [219] is of particular importance in the context of this thesis. Let r and b be determined from v, k, λ by (4.1), and let $(\mathcal{X}, \mathcal{R})$ be a resolution of a 2-(v, k, λ) design. Label the parallel classes in $\mathcal{R}$ as $1, 2, \ldots, r$ and the blocks within every parallel class as $0, 1, \ldots, v/k - 1$. With respect to this labeling, define an equireplicate equidistant $(r, v, r - \lambda)_{v/k}$ code as follows. Every point $p \in P(\mathcal{X})$ defines a word $x(p) = x_1(p)x_2(p)\cdots x_r(p) \in \mathbb{Z}_{v/k}^r$, where $x_j(p)$ is the label of the block incident to p in parallel class j, $1 \le j \le r$. The claimed properties of the code $C = \{x(p) : p \in P(\mathcal{X})\}$ are easy to verify from the defining properties of the resolution $(\mathcal{X}, \mathcal{R})$.

Conversely, it follows from the generalized Plotkin bound [14, Theorem 3] and (4.1) that every $(r, v, r - \lambda)_{v/k}$ code is both equireplicate and equidistant, and thus defines a resolution of a 2-(v, k, λ) design. Furthermore, it can be checked that the equivalence classes of such codes and isomorphism classes of resolutions are in a one-to-one correspondence. This correspondence is applied in [P1]. Analogous correspondences for one-factorizations of graphs and near resolutions of designs and are applied in [P3] and [P4], respectively.

Example 17 A labeled resolution of an STS(9) and the associated $(4, 9, 3)_3$ code appear below.

	Block 0	Block 1	Block 2
Parallel class 1:	123	456	789
Parallel class 2:	159	267	348
Parallel class 3:	147	258	369
Parallel class 4:	168	249	357

$$C = \{0000, 0111, 0222, 1201, 1012, 1120, 2102, 2210, 2021\}$$

4.2 ALGORITHMS FOR DESIGNS

Most existing techniques for classification of designs can roughly be divided into two types. Either one proceeds point by point, or block by block. Following a discussion of these two somewhat "myopic" approaches, other classification techniques are surveyed.

Survey articles on computer construction and classification of designs include [69, 70, 162, 224]. Classification results for block designs are surveyed in [164, 165, 166].

4.2.1 Point-by-Point Classification

Point by point classification is perhaps easiest to discuss in terms of incidence matrices. Let $\mathbf{N}$ be an incidence matrix of a 2-(v, k, λ) design. Equivalently, if r and b are determined by (4.1), then $\mathbf{N}$ is a $v \times b$ integer matrix with entries from $\{0, 1\}$ such that

(4.2) every row contains exactly r 1s; and

(4.3) every column contains exactly k 1s; and

(4.4) the inner product (in $\mathbb{Z}$) of every pair of distinct rows is λ.

An immediate approach to constructing such matrices is via backtrack search, where every search step adds one row (that is, point of a design) to a partial $w \times b$ incidence matrix $\mathbf{N}_w$ with $0 \leq w \leq v$. Denoting by $\mathbf{j}$ a column vector of all 1s, the candidate rows that can augment $\mathbf{N}_w$ correspond to the solutions $\mathbf{x}$ of the Diophantine linear equation system

$$\mathbf{j}^T\mathbf{x} = r, \qquad \mathbf{N}_w\mathbf{x} = \lambda\mathbf{j}, \qquad \mathbf{x} \in \{0, 1\}^{b \times 1}. \tag{4.5}$$

A further constraint is that every column of $\mathbf{N}_w$ that contains k 1s must contain a 0 in the corresponding position of $\mathbf{x}$ by (4.3). Two matrices are regarded as isomorphic if one can be obtained from the other by permuting the rows and columns.

Starting with Gibbons [68, 70] and Ivanov [94], this basic approach has been successfully employed in a number of studies, including [49, 50, 51, 104, 188, 189, 195, 221, 225]. Isomorph rejection in these studies is based on orderly generation. The order employed is a lexicographic order obtained by concatenating the rows of a matrix from least recently to most recently added (cf. Example 10). The extent of isomorph rejection varies from partial to complete, where complete isomorph rejection amounts to a lexicographic maximality test relative to permutation of the rows and columns of a matrix (see for example [51]). A partial isomorph rejection strategy based on testing the maximality of a newly constructed row relative to recorded automorphism groups of the submatrices $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_w$ (acting on the columns) is developed in [68, 70] and improved in [49, 50]. A somewhat different automorphism-based partial isomorph rejection strategy is employed in [225].

In addition to basic isomorph rejection, the search space may be further restricted by (4.3) and properties of lexicographic order. Namely, the lexicographically most significant column of $\mathbf{N}_w$ not satisfying (4.3) must be augmented with a 1; otherwise no augmentation of the resulting matrix is a lexicographic maximum incidence matrix (cf. Example 12 and [51, 221, 225]).

The strategies for solving the system (4.5) include column-by-column backtracking with pruning heuristics [68, 70, 162] and using general techniques for Diophantine linear equation systems; cf. Sect. 3.2.1. In the latter case it is advisable to replace each set $\{x_{i_1}, \ldots, x_{i_s}\}$ of $\{0, 1\}$-variables corresponding to a maximal set of identical columns in $\mathbf{N}_w$ with a single variable $x_i \in \{0, 1, \ldots, s\}$ to reduce symmetry in the system. In some cases a more specialized algorithm can be used. For example, if $\lambda = 1$, then (4.5) is an instance of the exact cover problem. A recursive strategy for solving the system on $\mathbf{N}_w$ based on the solutions obtained for $\mathbf{N}_{w-1}$ is described in [94]. An automorphism-based pruning strategy for column-by-column backtrack is described in [49, 50].

Clique search can be employed to locate completions of the partial matrix $\mathbf{N}_w$ to a full incidence matrix [104, 221, 225]. Namely, if X_w is the graph with the solutions of (4.5) as vertices, and two solutions are connected by an edge in X_w if and only if their inner product is equal to λ, then the $(v - w)$-cliques in X_w correspond (up to ordering of the rows) to the completions of $\mathbf{N}_w$. Whether clique search is practical depends on the structure of the graph X_w, in particular on its order and the symmetry induced by the automorphism group of $\mathbf{N}_w$ acting on the columns. A somewhat more sophisticated clique search strategy tailored for the Steiner systems $S(2, 4, 25)$ is employed in [225].

The orderly point-by-point construction approach has been employed also for t-designs with $t > 2$ [50, 51], for group divisible designs (GDDs) [200], and for balanced ternary designs [103]. A point-by-point construction approach based on generation by canonical augmentation is used in [173].

4.2.2 Block-by-Block Classification

The obvious alternative to point-by-point construction is to proceed block by block. In this case it is convenient to view the construction of t-(v, k, λ) designs over a fixed point set P as solving the following system of Diophantine linear equations. Let $\mathbf{A} = (a_{TK})$ be an integer matrix with rows and columns indexed by t-subsets $T \subseteq P$ and k-subsets $K \subseteq P$, respectively, such that

$$a_{TK} = \begin{cases} 1 & \text{if } T \subseteq K\text{; and} \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

The solutions $\mathbf{x}$ of the system of Diophantine linear equations

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{j}, \qquad \mathbf{x} \in \{0, 1, \ldots, \lambda\}^{\binom{v}{k}\times 1} \tag{4.7}$$

correspond—up to labeling of the blocks—to the t-(v, k, λ) designs over the point set P (cf. [79, 242]). Constructing the designs block by block can thus be viewed as setting the values of the variables x_K one at a time. Two vectors, $\mathbf{x}$ and $\mathbf{x}'$, are regarded isomorphic if there exists a $g \in \mathrm{Sym}(P)$ such that $x'_{g(K)} = x_K$ for all k-subsets $K \subseteq P$.

Isomorph rejection must be employed to eliminate symmetry from the system (4.7). Furthermore, in many cases the system (4.7) is too large to be explicitly constructed in practice, let alone solved (cf. [85, 132]). The standard remedy to these difficulties is to classify up to isomorphism a set of partial solutions—called *seeds*—such that every isomorphism class of designs can be encountered as an extension of a seed. In more precise terms, a seed is a vector $\mathbf{s} \in \{0, 1, \ldots, \lambda\}^{\binom{v}{k}\times 1}$ with $\mathbf{A}\mathbf{s} \leq \lambda\mathbf{j}$. (Here "$\leq$" indicates that inequality holds in all positions of the vector.) It is required that for every solution $\mathbf{x}'$ to (4.7), there exists an isomorphic solution $\mathbf{x}$ and a seed $\mathbf{s}$ such that $\mathbf{s} \leq \mathbf{x}$. Assuming that this property holds for the seeds, a complete classification of the designs can be obtained by determining, for each seed $\mathbf{s}$ in turn, all solutions $\mathbf{y}$ of the *residual system*

$$\mathbf{A}\mathbf{y} = \lambda\mathbf{j} - \mathbf{A}\mathbf{s}, \qquad \mathbf{y} \in \{0, 1, \ldots, \lambda\}^{\binom{v}{k}\times 1}, \tag{4.8}$$

and rejecting isomorphs among the generated designs $\mathbf{x} = \mathbf{s} + \mathbf{y}$. With an appropriate choice of seeds, the residual system is considerably smaller than the original system (4.7) because many of the variables can be ignored due to the fact that they are constrained to 0.

Often a good collection of seeds is obtained by selecting a small set of points $U \subseteq P$ and considering only those blocks in a design that are incident to at least one point (or all the points) in U. The properties of a design typically induce certain structure to such sets of blocks; these structural properties can then be exploited in classifying the sets of blocks up to isomorphism.

Certainly one of the earliest applications of this approach is the manual classification of the STS(15)s [43] based on seeds consisting of all blocks incident to at least one point from a pair of points. The subsequent computer verification of this result [84] employs an analogous strategy. In [P2] the seeds induced by a block B together with all blocks incident to at least one point of B are used to classify the STS(19)s. In this case the seeds correspond up to isomorphism to sets of three pairwise edge-disjoint one-factors of the complete graph K_{16}. The projective planes of orders $8 \leq n \leq 9$ were classified using a set of seeds induced by a set of three points not incident to a common block [85, 132]. Such seeds correspond up to isomorphism to main classes of Latin squares of order $n-1$, which were classified in [120, 185, 211] for $8 \leq n \leq 9$. The classifications in [33, 64, 163] employ seeds consisting of all blocks incident to a given point. In [176] the derived STS(13)s (blocks incident to a given point) are used as seeds to classify the Steiner systems $S(3,4,14)$.

The typical approach for isomorph rejection on generated designs is to keep a record of the isomorphism classes encountered, provided that the isomorphism class representatives can be accommodated in memory. Otherwise, generation by canonical augmentation can be employed, whereby the parent object associated with a generated design is the seed from which it is generated. The papers [P2, P3] can be considered as detailed examples of such an approach.

4.2.3 Other Approaches

Compared with "myopic" generation one point or block at a time, often a more efficient approach can be obtained by resorting to a more detailed combinatorial analysis of the designs in question and/or by alternating the two basic generation strategies.

In [173] three independent approaches are used to establish the nonexistence of 4-$(12,6,6)$ designs. Common to these approaches is the extensive application of a combinatorial analysis of the 4-$(12,6,6)$ designs and related $(v-7)$-$(v,v-6,3)$ designs for $8 \leq v \leq 12$ [124]. The first approach attempts to construct the 4-$(12,6,6)$ designs point by point via generation by canonical augmentation. The second and third approach prove nonexistence based on the property that a 4-$(12,6,6)$ design and its complement form a 5-$(12,6,3)$ design [124]. Thus, it suffices to check that no 5-$(12,6,3)$ design is resolvable. The second approach constructs the 5-$(12,6,3)$ designs point by point via generation by canonical augmentation. The third approach proceeds by extension along the sequence of derived $(v-7)$-$(v,v-6,3)$ designs, $8 \leq v \leq 12$.

In [221] a classification of the 2-$(31,10,3)$ designs employs first an orderly row-by-row strategy up to 17 rows (containing two full blocks), followed by a column-by-column strategy to complete the remaining 14 rows of the incidence matrix.

In [89] the nonexistence of a 2-$(46,6,1)$ design is shown by first locating two subconfigurations, "c4" and "c5", where a "c4" must occur and a "c5" may or may not occur. Then, the search is divided into two cases based on whether a "c5" or only a "c4" occurs, which in both cases enables the incidence matrix of a putative design to be partitioned into submatrices with certain structure. It is then established by exhaustive search that these submatrices cannot be completed, from which nonexistence follows. Analogous studies that assume the existence of a subconfiguration and then produce a complete classification subject to this assumption include [106, 192, 229].

Arguably the most important algorithmic classification result to date is the nonexistence of a projective plane of order 10, announced in [135]. The nonexistence result is based on ruling out the existence of words of weights 12 [136], 15 [158] ([48]), 16 [134], and 19 [135] in the binary linear code generated by the columns (or equivalently, rows) of an incidence matrix of a putative plane of order 10. Each word weight w is excluded by first classifying up to isomorphism all the possible configurations inducing a word of weight w, and then attempting—in vain—to extend each subconfiguration to a complete plane. These nonexistence results together with results on the weight enumerator polynomial of the code of a putative plane [2, 158] show that a plane of order 10 does not exist. An exposition is given in [129]. Analogous coding-theoretic techniques have been employed to obtain progress towards settling the nonexistence of a 2-$(22, 8, 4)$ design (see [207] and [7, 83, 174, 192]).

In many cases it is possible to employ a known correspondence between a family of designs and another family of combinatorial objects. Examples of useful correspondences between different families can be found in [35]. First, the objects in the corresponding family are classified up to the associated notion isomorphism, after which the designs of interest are determined up to isomorphism from these. For example, a classification of Hadamard matrices of order $4n$ (see [93, 112, 113, 223]) can be used to obtain a classification of the 3-$(4n, 2n, n-1)$ and the 2-$(4n-1, 2n-1, n-1)$ Hadamard designs. Similarly, the affine planes of order n correspond up to isomorphism to the block automorphism orbits in projective planes of order n.

4.3 ALGORITHMS FOR DESIGNS WITH PRESCRIBED AUTOMORPHISMS

An extensive literature exists on the automorphisms of designs of various types; see [8, 29, 40, 77, 90, 138] and the references therein. To employ prescribed automorphisms in classification, it is typically necessary to first conduct a combinatorial analysis of the types of automorphisms admitted by a design with given parameters (see for example [38, 171]). A putative group (or groups) of automorphisms can then be located using the necessary conditions for automorphisms resulting from such an analysis.

Let G be the group whose action induces isomorphism, and let $H \leq G$ be a prescribed group of automorphisms. From an algorithmic point of view there are essentially two extreme cases; namely, either H is a "small" group (such as a cyclic group of prime order) or a "large" group (such as a maximal or near maximal subgroup of G). As a rule of thumb, the larger the group H, the smaller the search space for classification. Symmetry in the search space under prescribed H can in most cases be captured by restricting the action of G to the normalizer $N_G(H) = \{g \in G : gHg^{-1} = H\}$ of H in G; cf. [28, 37, 38, 111, 155, 171, 218] and [P5].

4.3.1 The Kramer-Mesner Method

Kramer and Mesner [123] observed that t-designs with a prescribed group H of automorphisms (acting on the points) can be constructed by adapting the the equation system (4.7) to this setting.

More formally, let parameters t-(v, k, λ) be fixed, let P be a fixed finite set of v points, and let $H \leq G = \mathrm{Sym}(P)$. For notational convenience, we write $\bar{E}$ for the H-orbit of a subset $E \subseteq P$. Let $\mathbf{A}_H = (a_{\bar{T}\bar{K}})$ be an integer matrix with rows and columns indexed by H-orbits of t-subsets and k-subsets of P, respectively, such that $a_{\bar{T}\bar{K}} = |\{hK : T \subseteq hK, h \in H\}|$. The solutions $\mathbf{x}$ of the system of Diophantine linear equations

$$\mathbf{A}_H\mathbf{x} = \lambda\mathbf{j}, \qquad \mathbf{x} \in \{0, 1, \ldots, \lambda\}^{|\{\bar{K}:K\subseteq P,\ |K|=k\}|\times 1} \tag{4.9}$$

correspond (up to labeling of the blocks) to the t-(v, k, λ) designs over the point set P admitting H as a group of automorphisms.

The Kramer-Mesner method has been employed to construct and classify simple t-designs with small parameters; see [9, 10, 11, 121, 125, 140, 159, 215] for classic results and state of the art. In particular, the software package DISCRETA developed at Universität Bayreuth appears to be the main tool utilized in most recent construction/classification results for large groups (typically, projective and affine linear groups) and $t > 6$; see [140, 141].

The main algorithmic phases of a classification approach employing the Kramer-Mesner method are: constructing the equation system (4.9), solving the equations, and rejecting isomorphs.

For small groups (and small parameters t, v, k, λ), constructing the matrix $\mathbf{A}_H$ and associated orbits is straightforward. For large groups and large parameters, more sophisticated techniques are required; see [121, 213, 214, 215].

A number of techniques exist for solving the system (4.9). Typically the interest is on obtaining $\{0, 1\}$-solutions; that is, simple designs. Currently the most powerful techniques appear to be based on lattice basis reduction [1, 126, 127, 142, 238, 239], other techniques include [111, 121, 155, 214, 215]; see also [69] for a brief description of the algorithm SYNTH used by Magliveras and Mathon. Depending on the normalizer $N_G(H)$, initial isomorph rejection analogous to (4.8) may be required to eliminate symmetry from the search space; cf. [37, 111, 155]. For t-designs with $t > 2$, one possibility is to employ the derived 2-designs as seeds; cf. [111, 173]. For 2-designs, sets of blocks incident to at least one point from a given small set of points can be employed as seeds; cf. [P5].

Group-theoretic techniques for isomorph rejection among generated designs are developed in [82, 140, 141, 215]. Such techniques have their main applicability for large groups and large parameters, where the designs are too large for traditional isomorphism computations. For small groups and small parameters, the typical approach is to employ isomorph rejection via recorded representatives. An approach based on generation by canonical augmentation relative to seeds classified up to $N_G(H)$-isomorphism is developed in [P5] to attack instances where there are too many representatives to be stored in memory.

4.3.2 Tactical Decompositions

Tactical decompositions were introduced by Dembowski [46, 47]. Here we focus on the application of tactical decompositions in classification under prescribed automorphisms; a more general treatment appears in [8].

Let $\mathcal{X} = (P, \mathcal{B}, I)$ be a 2-(v, k, λ) design, and let $H \leq \mathrm{Aut}(\mathcal{X}) \leq G = \mathrm{Sym}(P) \times \mathrm{Sym}(\mathcal{B})$. Let $P_1, P_2, \ldots, P_m \subseteq P$ and $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n \subseteq \mathcal{B}$ be the H-orbits on the points and blocks, respectively. Select an arbitrary block B_j from every orbit $\mathcal{B}_j$, $1 \leq j \leq n$. Let $\mathbf{T} = (t_{ij})$ be the $m \times n$ integer matrix with $t_{ij} = |\{p \in P_i : (p, B_j) \in I\}|$ for all $1 \le i \le m$ and $1 \le j \le n$; it is easy to check that the number t_{ij} is independent of the block $B_j \in \mathcal{B}_j$ selected. An analogous $m \times n$ matrix $\mathbf{U} = (u_{ij})$ is obtained by selecting a point p_i from every orbit P_i and setting $u_{ij} = |\{B \in \mathcal{B}_j : (p_i, B) \in I\}|$.

In general, a *tactical decomposition* of $\mathcal{X}$ is a partition of the points and blocks such that the matrices $\mathbf{T}$ and $\mathbf{U}$ are independent of the point and block representatives selected. However, in the context of classification under prescribed automorphisms it is a common abuse of terminology (cf. [38, 122])—which we will follow here—to call an $m \times n$ nonnegative integer matrix $\mathbf{T} = (t_{ij})$ a tactical decomposition with respect to the prescribed group $H \le G$ if it satisfies the following equations (cf. Equations (4.2)–(4.4)):

$$\sum_{j=1}^{n} t_{ij}|\mathcal{B}_j| = r|P_i| \qquad \text{for all } 1 \le i \le m, \tag{4.10}$$

$$\sum_{i=1}^{m} t_{ij} = k \qquad \text{for all } 1 \le j \le n, \tag{4.11}$$

$$\sum_{j=1}^{n} t_{i_1 j} t_{i_2 j}|\mathcal{B}_j| = \lambda|P_{i_1}||P_{i_2}| \qquad \text{for all } 1 \le i_1 < i_2 \le m, \tag{4.12}$$

$$\sum_{j=1}^{n} \binom{t_{ij}}{2}|\mathcal{B}_j| = \lambda\binom{|P_i|}{2} \qquad \text{for all } 1 \le i \le m. \tag{4.13}$$

Tactical decompositions have been employed in numerous studies aimed at constructing and classifying 2-designs with prescribed automorphisms, including [28, 38, 45, 87, 95, 122, 161, 198, 222, 231, 232, 233, 234].

The algorithmic phases of a classification approach employing tactical decompositions are: classifying the tactical decompositions (up to isomorphism induced by the action of $N_G(H)$ on the H-orbits of points and blocks), expanding the tactical decompositions into incidence matrices of designs in all possible ways, and rejecting isomorphs among the generated designs.

Classification of tactical decompositions amounts essentially to solving the system (4.10)–(4.13) up to $N_G(H)$-isomorphism. One possibility is to proceed using row-by-row backtrack; cf. Sect. 4.2.1. An orderly algorithm is employed in [38]. The expansion of a tactical decomposition can be carried out using a backtrack search, where each step replaces one entry t_{ij} of $\mathbf{T}$ with a H-invariant $|P_i| \times |\mathcal{B}_j|$ partial incidence matrix with exactly t_{ij} 1s in every column, and a complete incidence matrix $\mathbf{N}$ must satisfy Equations (4.2)–(4.4). Strategies for expanding tactical decompositions are discussed in at least [28, 38]; the extent of isomorph rejection required during expansion depends on the normalizer $N_G(H)$. Possibilities for final isomorph rejection include using recorded representatives or generation by canonical augmentation relative to the classified tactical decompositions.

4.3.3 Other Approaches

In a sequence of papers [56, 57, 58, 59, 60], Eslami, Khosrovshahi, and Tayfeh-Rezaie employ trades in classifying halvings of complete $(v-8)$-$(v, v-7, 8)$ designs with prescribed automorphisms, $9 \le v \le 14$. Let

$t \leq k \leq v$ be positive integers and let P be a fixed set of v points. An unordered pair $\{\mathcal{T}_+, \mathcal{T}_-\}$ consisting of two disjoint collections of k-subsets of P is a t-(v,k) *trade* if every t-subset of P is occurs in the same number of k-subsets in $\mathcal{T}_+$ as in $\mathcal{T}_-$. The *volume* of a trade is the number of blocks in $\mathcal{T}_+$ ($\mathcal{T}_-$). Isomorphism of trades is induced by the action of $\mathrm{Sym}(P)$. Up to sign, a trade corresponds to an integer vector $\mathbf{t}$ satisfying $\mathbf{At} = \mathbf{0}$, where $\mathbf{A}$ is the inclusion matrix (4.6). The set of all such vectors form a $\mathbb{Z}$-module, whose structure and bases are analyzed in [78, 79, 91, 109, 110]. A halving of the complete t-$(v, k, \binom{v-t}{k-t})$ design corresponds to a t-(v,k) trade of volume $\binom{v}{k}/2$. To classify such trades up to isomorphism, Eslami, Khosrovshahi, and Tayfeh-Rezaie proceed by extending a classification of $(t-1)$-$(v-1, k-1)$ trades of volume $\binom{v-1}{k-1}/2$, and rejecting isomorphs. To carry out the extension, the standard basis of trades (see [110]) is employed; this basis has the convenient property that trades of volume $\binom{v}{k}/2$ can be expressed as $\{-1, 1\}$ linear combinations of the basis vectors. Analogously to the Kramer-Mesner method, prescribed automorphisms enable the restriction of the search to automorphism orbits.

Seah and Stinson [218] (see also [217]) employ an orderly algorithm to classify one-factorizations of complete graphs K_{12} and K_{14} with prescribed automorphisms. The algorithm proceeds by extending a partial factorization by one factor orbit at a time. Intermediate isomorph rejection is based on testing whether a partial factorization is the lexicographic minimum of its orbit under the action of the normalizer.

To obtain an enumeration of the main classes and isotopy classes of Latin squares up to order 10, McKay, Meynert, and Myrvold [171] develop a classification approach for Latin squares with a nontrivial automorphism group (with respect to the action of $S_n \wr S_3$ capturing the main classes of order n). First, a combinatorial analysis is conducted to determine a minimal set Σ of prime-order elements of $S_n \wr S_3$ such that every square with a nontrivial automorphism group admits at least one automorphism conjugate to a $\sigma \in \Sigma$. Then, for each $\sigma \in \Sigma$ in turn, the main classes admitting σ as an automorphism are constructed by employing two independent techniques based on either orderly generation or generation by canonical augmentation; the acting groups used in isomorph rejection are subgroups of $S_n \wr S_3$ normalizing σ. This achieves (almost) complete isomorph rejection among the Latin squares L with $\mathrm{Aut}(L) = \langle\sigma\rangle$ because $\mathrm{Aut}(L) = \mathrm{Aut}(L') = \langle\sigma\rangle$ and $L \cong_{S_n \wr S_3} L'$ immediately imply $L \cong_{N_{S_n \wr S_3}(\sigma)} L'$. Final isomorph rejection for the stored Latin squares L with $\mathrm{Aut}(L) \neq \langle\sigma\rangle$ for all $\sigma \in \Sigma$ is carried out using a canonical representative map based on transformation into a vertex-colored graph.

4.4 ALGORITHMS FOR CODES

This section surveys classification techniques for codes. The emphasis is on unrestricted error-correcting codes, $(n, M, d)_q$ codes, and in particular on codes that correspond to resolutions of designs and one-factorizations of regular graphs (cf. [P1, P3, P4]). The classification of codes with more restricted algebraic structure—in particular, linear codes—and other types of codes—

such as covering codes—is not considered.

Analogously to classification techniques for designs in Sect. 4.2, there are essentially two ways to proceed in classifying $(n, M, d)_q$ codes up to equivalence. Either one proceeds codeword by codeword; that is, from $(n, M - 1, d)_q$ codes to $(n, M, d)_q$ codes, or coordinate by coordinate; that is, from $(n - 1, M, d - 1)_q$ codes to $(n, M, d)_q$ codes.

4.4.1 Codeword-by-Codeword Classification

For integers $q \geq 2$ and $n \geq d \geq 1$, one possibility to study codeword-by-codeword classification of $(n, M, d)_q$ codes is via Hamming graphs. The *Hamming graph* $H_q(n, d)$ has the set $\mathbb{Z}_q^n$ of words as vertices, and two vertices (words) x, y are connected by an edge if and only if $d(x, y) \geq d$. (If equidistant codes are considered, it is required that $d(x, y) = d$.) Clearly, the $(n, M, d)_q$ codes are in a one-to-one correspondence with the M-cliques in $H_q(n, d)$. Thus, codeword-by-codeword classification essentially amounts to clique search on $H_q(n, d)$. Additional constraints—such as a code being equireplicate—can also be relatively easily included if necessary.

Except possibly for the smallest parameter values, isomorph rejection on partial solutions is again a prerequisite for a practical algorithm. The group action inducing equivalence for unrestricted codes is the product action of the wreath product $S_q \wr S_n$ on $\mathbb{Z}_q^n$; equivalence computations can be carried out conveniently by transforming a code into a graph (see [193]).

A number of ways to structure a codeword-by-codeword search occur in the literature. In [193] the $(10, 72, 3)_2$ and $(11, 144, 3)_2$ codes are classified using a strategy based on extending subcodes. Let C be an $(n, M, d)_2$ code, and let C_0 (C_1) be the subcode obtained by taking all the codewords with a 0 (a 1) in the first coordinate. Up to equivalence we can assume $|C_0| \geq |C_1|$. If we remove the first coordinate from the words in C_0, the result is an $(n - 1, M', d)_2$ code with $M' \geq M/2$. Provided that we have a classification of all such codes up to equivalence, the $(n, M, d)_2$ codes can be constructed by locating—for every applicable code C_0—all $(M - |C_0|)$-cliques in the subgraph of $H_2(n, d)$ induced by words having a 1 in the first coordinate and having distance at least d to every word in C_0. (Note that [193] does not explicitly mention clique search, but the approach described is equivalent to applying the clique algorithm in [191] with the vertices appearing in lexicographic order.) After rejecting equivalent codes via transformation into graphs, a classification is obtained. This approach is generalized to mixed binary/ternary codes in [187]; also generalization to codes with $q > 3$ is possible. A similar approach is employed in [148, 149].

In [P1] an orderly codeword-by-codeword strategy combined with clique searching is used to establish the nonexistence of a $(14, 15, 10)_3$ code; that is, a resolution of a 2-$(15, 5, 4)$ design. Since such a code is necessarily both equireplicate and equidistant, these properties can be exploited to restrict the search together with properties of lexicographic minimum equireplicate codes (cf. Example 12). The canonicity test in [P1] was developed based on an idea for testing code equivalence in [101]. An orderly approach similar to [P1] is employed in [P3] and [102]. This orderly approach is generalized to "gap codes" corresponding to near resolutions of designs in [P4].

It appears that generation by canonical augmentation has not been used as an isomorph rejection strategy in published studies pertaining to codeword-by-codeword generation, although the possibility of such an approach is mentioned for example in [187]. Implemented with appropriate invariants, an approach based on generation by canonical augmentation is likely to improve considerably the efficiency of isomorph rejection compared with the use of recorded representatives. The present author (unpublished) has implemented an algorithm employing generation by canonical augmentation to double check the classification of "gap codes" corresponding to 2-$(13, 4, 3)$ near resolutions reported in [P4]. The invariant used to structure the search in this case is as follows. A "gap code" C of cardinality $1 \leq M \leq 13$ is said to have the *replication property* if there exists a coordinate such that $\lfloor M/4 \rfloor$ values occur exactly 4 times in the coordinate, and the remaining $M - 4\lfloor M/4 \rfloor$ words contain the same value in the coordinate; the latter value is allowed to be the "gap" only if $M = 13$. Given a "gap code" C, the subcode $m(C) \subseteq C$ is selected—via a canonical labeling map as in Example 13—so that if C has the replication property, then also $m(C)$ has the replication property with $|m(C)| = |C| - 1$. Accordingly, the search for words that extend a given code may be restricted to those words whose addition produces a code with the replication property; cf. Example 12 and [P4].

4.4.2 Coordinate-by-Coordinate Classification

The basic problem associated with coordinate-by-coordinate classification is that of extending an $(n-1, M, d-1)_q$ code C in all possible ways to an $(n, M, d)_q$ code by adding a new coordinate. This is equivalent to the problem of finding all vertex q-colorings of the graph G defined by $V(G) = C$ and $E(G) = \{\{x, y\} : x, y \in C,\ d(x, y) = d - 1\}$, where the "colors" $\{0, 1, \ldots, q-1\}$ indicate the symbol to be appended to each word in C; for the binary case, cf. the proof of [152, Theorem 5] and [193]. Up to equivalence, it suffices to consider only one coloring from each $S_q \times \mathrm{Aut}(C)$-orbit of q-colorings, where S_q acts by permuting the colors and $\mathrm{Aut}(C) \leq S_q \wr S_{n-1}$ acts on $C = V(G)$.

Coordinate-by-coordinate classification has been used in comparatively few studies, most of which concern the classification of codes corresponding to resolutions of designs, whereby coordinate-by-coordinate classification corresponds to classification one parallel class of the resolution at a time.

An orderly algorithm is used in [54, 217] to classify one-factorizations of the complete graph K_{2n} (equivalently, resolutions of a 2-$(2n, 2, 1)$ design; or, $(2n-1, 2n, 2n-2)_n$ codes) for $2n = 10, 12$.

In [182, 183] a backtrack algorithm that proceeds one parallel class at a time is used to classify resolutions of the 2-$(12, 4, 3)$ and 2-$(10, 5, 16)$ designs; that is, the $(11, 12, 8)_3$ and $(36, 10, 20)_2$ codes. The efficiency of both classifications is based on a careful combinatorial analysis of the possible block intersection patterns between parallel classes, which produces a set of starting configurations and enables to restrict the search. The algorithm in [183] also applies intermediate isomorph rejection based on recorded representatives and an ordering heuristic for the parallel classes.

In [P3] a combination of generation by canonical augmentation and orderly generation is applied to classify one-factorizations of k-regular graphs of order 12; that is, equireplicate $(k, 12, k-1)_6$ codes. Generation by canonical augmentation is used to perform isomorph rejection among the generated codes, whereas orderly generation is used to reject $S_6 \times \mathrm{Aut}(C)$-isomorphic 6-colorings that extend C.

4.4.3 Other Approaches

Other techniques used to classify $(n, M, d)_q$ codes corresponding to resolutions of designs include the following. An obvious possibility is to first classify the underlying designs, then resolve these in all possible ways, and reject isomorphs (see for example [194]); however, this is not very efficient if there are far more designs than there are resolvable designs. In [P3] an approach derived from [P2] and based on viewing a one-factorization of K_{2n} (that is, resolutions of the 2-$(2n, 2, 1)$ design) as a particular triple system on $4n-1$ points is used to verify the classification of one-factorizations of K_{12} obtained in [54]. In [137] a classification of the affine resolvable 2-$(27, 9, 4)$ designs is obtained by using the 2-$(13, 4, 2)$ designs as seeds.

5 CONCLUSIONS

This chapter briefly sums up the results in this thesis and presents some points for future work together with some general remarks.

The main result of this thesis is a classification of the Steiner triple systems of order 19 [P2]. The other results obtained include the nonexistence of a resolvable 2-$(15, 5, 4)$ design [P1], a classification of the one-factorizations of k-regular graphs of order 12 for $k \leq 6$ and $k = 10, 11$ [P3], a classification of the near-resolutions of 2-$(13, 4, 3)$ designs together with the associated thirteen-player whist tournaments [P4], and a classification of the Steiner triple systems of order 21 with a nontrivial automorphism group [P5].

Classification work to be conducted in the near future includes performing the classification of the one-factorizations of K_{14} with a nontrivial automorphism group, discussed in [P5]. Furthermore, a preliminary estimate indicates that a classification of the Steiner quadruple systems of order 16 (3-$(16, 4, 1)$ designs) is feasible by employing the 80 nonisomorphic STS(15)s as seeds in a block-by-block approach analogous to [P2]; cf. [176].

In general, with improvements in algorithms and computer performance, it is to be expected that computers are going to be employed in an increasing number of studies. As Gibbons [69] puts it:

> "Over the past 25 years the computer has become an indispensable ally in the search for combinatorial designs of many types. The use of clever computational techniques has not only enabled many existence and enumeration questions to be settled, but also allowed larger classes of designs to be analyzed, often leading to the formulation of conjectures which have been proved for infinite families of designs."

This paragraph is certainly as true today as it was in 1996. Especially the development of fast versatile techniques for isomorph rejection with low storage requirements—most notably, the ingenious canonical augmentation technique of McKay [170]—now enables classification on instances with isomorphism classes numbering in the billions. Also advances in exact algorithms for a number of standard combinatorial optimization problems can be exploited in classification. A general technique that warrants further investigation in a classification context is the use of linear programming relaxations in pruning parts of a search tree associated with solving a system of Diophantine linear equations; cf. [160].

As can be concluded from the preceding chapters, an algorithm design in most cases applies both combinatorial knowledge specific to the objects of interest, as well as general principles and techniques, such as backtrack search and isomorph rejection. A further characteristic in most cases associated with the design of classification algorithms is experimentation; that is, the need to assess whether a putative classification approach is practical. In this sense computer investigations into combinatorial classification can be seen as an experimental science: an experimental setup can fail, and errors can occur. Indeed, it is all too frequently the case that a consistency check reveals a subtle error in an algorithm implementation. Thus, consistency checking and double checking the result are highly recommended.

Given the experimental nature of the area, it is important to have available reliable general-purpose tools that enable rapid feasibility studies to be carried out, and, furthermore, that can be relatively easily tailored for performance. The graph isomorphism package *nauty* [169] is an excellent example of such a software tool; another example is the package Cliquer [184] for solving clique problems on graphs. More tools of this nature are in constant demand. Also important is the integration of tools with a convenient high-level programming interface—for example, the GAP system [65]—with low-level tools for computing isomorphism and solvers for combinatorial optimization problems. Such integration can be carried out either through external executables invoked by the high-level system, or through integration into the kernel of the high-level system for improved latency. Not only does such a tool decrease time required by experimentation, it also arguably reduces the probability of error in less performance-critical tasks by enabling them to be carried out in a high-level environment. Examples of successful high-level tools include the DISCRETA system developed at Universität Bayreuth and the BDX system [131] developed at Concordia University. Again, further tools are in demand.

However, it certainly appears that no amount of general-purpose tool engineering can provide the performance gain obtainable by a combinatorial insight into the structure of the objects of interest. An excellent example is the celebrated nonexistence result for projective planes of order 10 [135], which fundamentally relies on a combinatorial analysis of the weight enumerator of the code of a putative plane. Developing and applying analogous structural analysis techniques specific to the objects of interest is arguably the most rewarding topic of research associated with classification.

BIBLIOGRAPHY

[1] K. Aardal, C. A. J. Hurkens, and A. K. Lenstra, Solving a system of linear Diophantine equations with lower and upper bounds on the variables, *Math. Oper. Res.* 25 (2000), 427–442.

[2] E. F. Assmus, Jr. and H. F. Mattson, Jr., On the possibility of a projective plane of order 10, Algebraic Theory of Codes II, Air Force Cambridge Research Laboratories Report AFCRL-71-0013, Sylvania Electronic Systems, Needham Heights, Mass., 1970.

[3] L. Babai, Moderately exponential bound for graph isomorphism, *Fundamentals of Computation Theory* (F. Gécseg, Ed.), Springer-Verlag, Berlin, 1981, pp. 34–50.

[4] L. Babai, Automorphism groups, isomorphism, reconstruction, *Handbook of Combinatorics* (R. L. Graham, M. Grötschel, and L. Lovász, Eds.), Vol. II, North-Holland, Amsterdam, 1995, pp. 1447–1540.

[5] L. Babai and E. M. Luks, Canonical labeling of graphs, *Proc. Fifteenth Annual ACM Symposium on Theory of Computing*, (Boston, April 25–27, 1983), ACM Press, New York, 1983, pp. 171–183.

[6] S. E. Bammel and J. Rothstein, The number of 9×9 Latin squares, *Discrete Math.* 11 (1975), 93–95.

[7] J. A. Bate, M. Hall, Jr., and G. H. J. van Rees, Structures within $(22, 33, 12, 8, 4)$-designs, *J. Combin. Math. Combin. Comput.* 4 (1988), 115–122.

[8] T. Beth, D. Jungnickel, and H. Lenz, *Design Theory*, 2nd ed., 2 vols., Cambridge University Press, Cambridge, 1999.

[9] A. Betten, A. Kerber, A. Kohnert, R. Laue, and A. Wassermann, The discovery of simple 7-designs with automorphism group $P\Gamma L(2, 32)$, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes* (G. Cohen, M. Giusti, and T. Mora, Eds.), Springer-Verlag, Berlin, 1995, pp. 131–145.

[10] A. Betten, A. Kerber, R. Laue, and A. Wassermann, Simple 8-designs with small parameters, *Des. Codes Cryptogr.* 15 (1998), 5–27.

[11] A. Betten, R. Laue, and A. Wassermann, Simple 7-designs with small parameters, *J. Combin. Des.* 7 (1999), 79–94.

[12] N. Biggs, *Algebraic Graph Theory*, 2nd ed., Cambridge University Press, Cambridge, 1993.

[13] A. Blass and Y. Gurevich, Equivalence relations, invariants, and normal forms, *SIAM J. Comput.* 13 (1984), 682–689.

[14] G. T. Bogdanova, A. E. Brouwer, S. N. Kapralov, and P. R. J. Östergård, Error-correcting codes over an alphabet of four elements, *Des. Codes Cryptogr.* 23 (2001), 333–342.

[15] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, The maximum clique problem, *Handbook of Combinatorial Optimization* (D.-Z. Du and P. M. Pardalos, Eds.), Vol. A (supplement), Kluwer, Dordrecht, the Netherlands, 1999, pp. 1–74.

[16] G. Brinkmann, Isomorphism rejection in structure generation programs, *Discrete Mathematical Chemistry* (P. Hansen, P. Fowler, and M. Zheng, Eds.), Amer. Math. Soc., Providence, R.I., 2000, pp. 25–38.

[17] G. Brinkmann and B. D. McKay, Posets on up to 16 points, *Order* 19 (2002), 147–179.

[18] A. E. Brouwer, A. M. Cohen, and A. Neumaier, *Distance-Regular Graphs*, Springer-Verlag, Berlin, 1989.

[19] J. W. Brown, Enumeration of Latin squares with application to order 8, *J. Combin. Theory* 5 (1968), 177–184.

[20] G. Butler, *Fundamental Algorithms for Permutation Groups*, Springer-Verlag, Berlin, 1991.

[21] G. Butler and C. W. H. Lam, A general backtrack algorithm for the isomorphism problem of combinatorial objects, *J. Symbolic Comput.* 1 (1985), 363–381.

[22] J.-Y. Cai, M. Fürer, and N. Immerman, An optimal lower bound on the number of variables for graph identification, *Combinatorica* 12 (1992), 389–410.

[23] P. J. Cameron, *Parallelisms of Complete Designs*, Cambridge University Press, Cambridge, 1976.

[24] P. J. Cameron, *Combinatorics: Topics, Techniques, Algorithms*, Cambridge University Press, Cambridge, 1994.

[25] P. J. Cameron, *Permutation Groups*, Cambridge University Press, Cambridge, 1999.

[26] P. J. Cameron and J. H. van Lint, *Designs, Graphs, Codes and Their Links*, Cambridge University Press, Cambridge, 1991.

[27] R. Carraghan and P. M. Pardalos, An exact algorithm for the maximum clique problem, *Oper. Res. Lett.* 9 (1990), 375–382.

[28] V. Ćepulić, On symmetric block designs $(40, 13, 4)$ with automorphisms of order 5, *Discrete Math.* 128 (1994), 45–60.

[29] L. G. Chouinard II, R. Jajcay, and S. S. Magliveras, Finite groups and designs, *The CRC Handbook of Combinatorial Designs* (C. J. Colbourn and J. H. Dinitz, Eds.), CRC Press, Boca Raton, Fla., 1996, pp. 587–615.

[30] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein, *Covering Codes*, Elsevier, Amsterdam, 1997.

[31] C. J. Colbourn, Embedding partial Steiner triple systems is NP-complete, *J. Combin. Theory Ser. A* 35 (1983), 100–105.

[32] C. J. Colbourn and M. J. Colbourn, Combinatorial isomorphism problems involving 1-factorizations, *Ars. Combin.* 9 (1980), 191–200.

[33] C. J. Colbourn, M. J. Colbourn, J. J. Harms, and A. Rosa, A complete census of $(10, 3, 2)$ block designs and of Mendelsohn triple systems of order ten. III. $(10, 3, 2)$ block designs without repeated blocks, *Congr. Numer.* 37 (1983), 211–234.

[34] C. J. Colbourn and J. H. Dinitz, Eds., *The CRC Handbook of Combinatorial Designs*, CRC Press, Boca Raton, Fla., 1996.

[35] C. J. Colbourn and J. H. Dinitz, Latin squares, *The CRC Handbook of Combinatorial Designs* (C. J. Colbourn and J. H. Dinitz, Eds.), CRC Press, Boca Raton, Fla., 1996, pp. 97–110.

[36] C. J. Colbourn and J. H. Dinitz, Applications of combinatorial designs to communications, cryptography, and networking, *Surveys in Combinatorics, 1999* (J. D. Lamb and D. A. Preece, Eds.), Cambridge University Press, Cambridge, 1999, pp. 37–100.

[37] C. J. Colbourn, S. S. Magliveras, and R. A. Mathon, Transitive Steiner and Kirkman triple systems of order 27, *Math. Comp.* 58 (1992), 441–449.

[38] C. J. Colbourn, S. S. Magliveras, and D. R. Stinson, Steiner triple systems of order 19 with nontrivial automorphism group, *Math. Comp.* 59 (1992), 283–295.

[39] C. J. Colbourn and P. C. van Oorschot, Applications of combinatorial designs in computer science, *ACM Computing Surv.* 21 (1989), 223–249.

[40] C. J. Colbourn and A. Rosa, *Triple Systems*, Clarendon Press, Oxford, 1999.

[41] M. J. Colbourn, Algorithmic aspects of combinatorial designs: A survey, *Ann. Discrete Math.* 26 (1985), 67–136.

[42] M. J. Colbourn and C. J. Colbourn, Concerning the complexity of deciding isomorphism of block designs, *Discrete Appl. Math.* 3 (1981), 155–162.

[43] F. N. Cole, L. D. Cummings, and H. S. White, The complete enumeration of triad systems in 15 elements, *Proceedings of the National Academy of Sciences of the United States of America* 3 (1917), 197–199.

[44] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, 2001.

[45] D. Crnković, Symmetric $(70, 24, 8)$ designs having $\mathrm{Frob}_{21} \times Z_2$ as an automorphism group, *Glas. Mat. Ser. III* 34(54) (1999), 109–121.

[46] P. Dembowski, Verallgemeinerungen von Transitivitätsklassen endlicher projektiver Ebenen, *Math. Z.* 69 (1958), 59–89.

[47] P. Dembowski, *Finite Geometries*, Springer-Verlag, Berlin, 1997. Reprint of the 1968 edition.

[48] R. H. F. Denniston, Non-existence of a certain projective plane, *J. Austral. Math. Soc.* 10 (1969), 214–218.

[49] P. C. Denny, Search and enumeration techniques for incidence structures, Research Report CDMTCS-085, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, 1998.

[50] P. C. Denny and P. B. Gibbons, Case studies and new results in combinatorial enumeration, *J. Combin. Des.* 8 (2000), 239–260.

[51] P. C. Denny and R. Mathon, A census of t-$(t+8, t+2, 4)$ designs, $2 \leq t \leq 4$, *J. Statist. Plann. Inference* 106 (2002), 5–19.

[52] L. E. Dickson and F. H. Safford, Solution to problem 8 (group theory), *The American Mathematical Monthly* 13 (1906), 150–151.

[53] R. Diestel, *Graph Theory*, 2nd ed., Springer-Verlag, New York, 2000.

[54] J. H. Dinitz, D. K. Garnick, and B. D. McKay, There are 526,915,620 nonisomorphic one-factorizations of K_{12}, *J. Combin. Des.* 2 (1994), 273–285.

[55] J. D. Dixon and B. Mortimer, *Permutation Groups*, Springer-Verlag, New York, 1996.

[56] Z. Eslami and G. B. Khosrovshahi, Classification of some large sets of designs, *J. Geom.* 67 (2000), 105–110.

[57] Z. Eslami and G. B. Khosrovshahi, A complete classification of 3-$(11, 4, 4)$ designs with nontrivial automorphism group, *J. Combin. Des.* 8 (2000), 419–425.

[58] Z. Eslami and G. B. Khosrovshahi, Some new 6-$(14, 7, 4)$ designs, *J. Combin. Theory Ser. A* 93 (2001), 141–152.

[59] Z. Eslami, G. B. Khosrovshahi, and B. Tayfeh-Rezaie, On halvings of the 2-$(10, 3, 8)$ design, *J. Statist. Plann. Inference* 86 (2000), 411–419.

[60] Z. Eslami, G. B. Khosrovshahi, and B. Tayfeh-Rezaie, On classification of 2-$(8, 3)$ and 2-$(9, 3)$ trades, *J. Combin. Math. Combin. Comput.* 38 (2001), 231–242.

[61] I. A. Faradžev, Constructive enumeration of combinatorial objects, *Problèmes Combinatoires et Théorie des Graphes*, (Université d'Orsay, July 9–13, 1977), CNRS, Paris, 1978, pp. 131–135.

[62] L. M. Finkelstein and W. M. Kantor, Eds., *Groups and Computation*, Amer. Math. Soc., Providence, R.I., 1993.

[63] L. M. Finkelstein and W. M. Kantor, Eds., *Groups and Computation, II*, Amer. Math. Soc., Providence, R.I., 1997.

[64] B. Ganter, R. Mathon, and A. Rosa, A complete census of $(10, 3, 2)$-block designs and of Mendelsohn triple systems of order ten. II. Mendelsohn triple systems with repeated blocks, *Congr. Numer.* 22 (1978), 181–204.

[65] The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. Available electronically at ⟨URL: `http://www-gap.dcs.st-and.ac.uk/~gap`⟩.

[66] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Fransisco, 1979.

[67] E. N. Gelling and R. E. Odeh, On 1-factorizations of the complete graph and the relationship to round-robin schedules, *Congr. Numer.* 9 (1974), 213–221.

[68] P. B. Gibbons, *Computing Techniques for the Construction and Analysis of Block Designs*, PhD Thesis, University of Toronto, 1976.

[69] P. B. Gibbons, Computational methods in design theory, *The CRC Handbook of Combinatorial Designs* (C. J. Colbourn and J. H. Dinitz, Eds.), CRC Press, Boca Raton, Fla., 1996, pp. 718–740.

[70] P. B. Gibbons, R. A. Mathon, and D. G. Corneil, Computing techniques for the construction and analysis of block designs, *Util. Math.* 11 (1977), 161–192.

[71] C. Godsil and G. Royle, *Algebraic Graph Theory*, Springer-Verlag, New York, 2001.

[72] L. A. Goldberg, *Efficient Algorithms for Listing Combinatorial Structures*, Cambridge University Press, Cambridge, 1993.

[73] L. A. Goldberg, Computation in permutation groups: Counting and randomly sampling orbits, *Surveys in Combinatorics, 2001* (J. W. P. Hirschfeld, Ed.), Cambridge University Press, Cambridge, 2001, pp. 109–143.

[74] M. Goldberg, The graph isomorphism problem, *Handbook of Graph Theory* (J. L. Gross and J. Yellen, Eds.), CRC Press, Boca Raton, Fla., 2004, pp. 68–78.

[75] S. W. Golomb, A mathematical theory of discrete classification, *Information Theory* (C. Cherry, Ed.), Butterworths, Washington, D.C., 1961, pp. 404–425.

[76] S. W. Golomb and L. D. Baumert, Backtrack programming, *J. Assoc. Comput. Mach.* 12 (1965), 516–524.

[77] R. L. Graham, M. Grötschel, and L. Lovász, Eds., *Handbook of Combinatorics*, 2 vols., North-Holland, Amsterdam, 1995.

[78] R. L. Graham, S.-Y. R. Li, and W.-C. W. Li, On the structure of t-designs, *SIAM J. Algebraic Discrete Methods* 1 (1980), 8–14.

[79] J. E. Graver and W. B. Jurkat, The module structure of integral designs, *J. Combin. Theory Ser. A* 15 (1973), 75–90.

[80] R. Grund, A. Kerber, and R. Laue, MOLGEN, ein Computeralgebra-System für die Konstruktion molekularer Graphen, *Match* 27 (1992), 87–131.

[81] T. Grüner, R. Laue, and M. Meringer, Algorithms for group actions applied to graph generation, *Groups and Computation, II* (L. Finkelstein and W. M. Kantor, Eds.) Amer. Math. Soc., Providence, R.I., 1997, pp. 113–122.

[82] E. Haberberger, A. Betten, and R. Laue, Isomorphism classification of t-designs with group theoretical localisation techniques applied to some Steiner quadruple systems on 20 points, *Congr. Numer.* 142 (2000), 75–96.

[83] M. Hall, Jr., R. Roth, G. H. J. van Rees, and S. A. Vanstone, On designs $(22, 33, 12, 8, 4)$, *J. Combin. Theory Ser. A* 47 (1988), 157–175.

[84] M. Hall, Jr. and J. D. Swift, Determination of Steiner triple systems of order 15, *Math. Tables Aids Comput.* 9 (1955), 146–152.

[85] M. Hall, Jr., J. D. Swift, and R. J. Walker, Uniqueness of the projective plane of order eight, *Math. Tables Aids Comput.* 10 (1956), 186–194.

[86] Z. Hedrlín and A. Pultr, On full embeddings of categories of algebras, *Illinois J. Math.* 10 (1966), 392–406.

[87] D. Held and M.-O. Pavčević, Symmetric $(79, 27, 9)$-designs admitting a faithful action of a Frobenius group of order 39, *European J. Combin.* 18 (1997), 409–416.

[88] C. M. Hoffmann, *Group-Theoretic Algorithms and Graph Isomorphism*, Springer-Verlag, Berlin, 1982.

[89] S. K. Houghten, L. H. Thiel, J. Janssen, and C. W. H. Lam, There is no $(46, 6, 1)$ block design, *J. Combin. Des.* 9 (2001), 60–71.

[90] D. R. Hughes and F. C. Piper, *Design Theory*, Cambridge University Press, Cambridge, 1985.

[91] H. L. Hwang, On the structure of (v, k, t) trades, *J. Statist. Plann. Inference* 13 (1986), 179–191.

[92] N. Immerman, *Descriptive Complexity*, Springer-Verlag, New York, 1999.

[93] N. Ito, J. S. Leon, and J. Q. Longyear, Classification of 3-$(24, 12, 5)$ designs and 24-dimensional Hadamard matrices, *J. Combin. Theory Ser. A* 31 (1981), 66–93.

[94] A. V. Ivanov, Constructive enumeration of incidence systems, *Ann. Discrete Math.* 26 (1985), 227–246.

[95] Z. Janko and T. van Trung, Construction of a new symmetric block design for $(78, 22, 6)$ with the help of tactical decompositions, *J. Combin. Theory Ser. A* 40 (1985), 451–455.

[96] T. R. Jensen and B. Toft, *Graph Coloring Problems*, Wiley, New York, 1995.

[97] M. Jerrum, *Counting, Sampling and Integrating: Algorithms and Complexity*, Birkhäuser, Basel, 2003.

[98] D. S. Johnson and M. A. Trick, Eds., *Cliques, Coloring, and Satisfiability*, Amer. Math. Soc., Providence, R.I., 1996.

[99] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, On generating all maximal independent sets, *Inform. Process. Lett.* 27 (1988), 119–123.

[100] W. M. Kantor and Á. Seress, Eds., *Groups and Computation, III*, Walter de Gruyter, Berlin, 2001.

[101] K. S. Kapralov, The nonexistence of ternary $(10, 15, 7)$ codes, *Proc. 7th International Workshop on Algebraic and Combinatorial Coding Theory (ACCT'2000)*, (Bansko, Bulgaria, June 18–24, 2000), 2000, pp. 189–192.

[102] P. Kaski, L. B. Morales, P. R. J. Östergård, D. A. Rosenblueth, and C. Velarde, Classification of resolvable 2-$(14, 7, 12)$ and 3-$(14, 7, 5)$ designs, *J. Combin. Math. Combin. Comput.* 47 (2003), 65–74.

[103] P. Kaski and P. R. J. Östergård, Enumeration of balanced ternary designs, *Discrete Appl. Math.* 138 (2004), 133–141.

[104] P. Kaski and P. R. J. Östergård, Miscellaneous classification results for 2-designs, *Discrete Math.* 280 (2004), 65–75.

[105] P. Kaski and P. R. J. Östergård, There exist nonisomorphic STS(19) with equivalent point codes, *J. Combin. Des.* 12 (2004), 443–448.

[106] P. Kaski, P. R. J. Östergård, S. Topalova, and R. Zlatarski, Steiner triple systems of order 19 and 21 with subsystems of order 7, *Discrete Math.*, to appear.

[107] A. Kerber, *Applied Finite Group Actions*, 2nd ed., Springer-Verlag, Berlin, 1999.

[108] A. Kerber and R. Laue, Group actions, double cosets, and homomorphisms: Unifying concepts for the constructive theory of discrete structures, *Acta Appl. Math.* 52 (1998), 63–90.

[109] G. B. Khosrovshahi and S. Ajoodani-Namini, A new basis for trades, *SIAM J. Discrete Math.* 3 (1990), 364–372.

[110] G. B. Khosrovshahi and C. Maysoori, On the bases for trades, *Linear Algebra Appl.* 226/228 (1995), 731–748.

[111] G. B. Khosrovshahi, M. Mohammad-Noori, and B. Tayfeh-Rezaie, Classification of 6-$(14, 7, 4)$ designs with nontrivial automorphism groups, *J. Combin. Des.* 10 (2002), 180–194.

[112] H. Kimura, New Hadamard matrix of order 24, *Graphs Combin.* 5 (1989), 235–242.

[113] H. Kimura, Classification of Hadamard matrices of order 28, *Discrete Math.* 133 (1994), 171–180.

[114] D. E. Knuth, Estimating the efficiency of backtrack programs, *Math. Comp.* 29 (1975), 121–136.

[115] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, Mass., 1998.

[116] D. E. Knuth, Dancing links, *Millennial Perspectives in Computer Science* (J. Davies, B. Roscoe, and J. Woodcock, Eds.), Palgrave, Basingstoke, England, 2000, pp. 187–214.

[117] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser, Boston, 1993.

[118] W. Kocay, Groups & Graphs, a Macintosh application for graph theory, *J. Combin. Math. Combin. Comput.* 3 (1988), 195–206.

[119] W. Kocay, On writing isomorphism programs, *Computational and Constructive Design Theory* (W. D. Wallis, Ed.), Kluwer, Dordrecht, the Netherlands, 1996, pp. 135–175.

[120] G. Kolesova, C. W. H. Lam, and L. Thiel, On the number of 8×8 Latin squares, *J. Combin. Theory Ser. A* 54 (1990), 143–148.

[121] E. S. Kramer, D. W. Leavitt, and S. S. Magliveras, Construction procedures for t-designs and the existence of new simple 6-designs, *Ann. Discrete Math.* 26 (1985), 247–273.

[122] E. S. Kramer, S. S. Magliveras, and R. Mathon, The Steiner systems $S(2, 4, 25)$ with nontrivial automorphism group, *Discrete Math.* 77 (1989), 137–157.

[123] E. S. Kramer and D. M. Mesner, t-designs on hypergraphs, *Discrete Math.* 15 (1976), 263–296.

[124] D. L. Kreher, D. de Caen, S. A. Hobart, E. S. Kramer, and S. P. Radziszowski, The parameters 4-$(12, 6, 6)$ and related t-designs, *Australas. J. Combin.* 7 (1993), 3–20.

[125] D. L. Kreher and S. P. Radziszowski, The existence of simple 6-$(14, 7, 4)$ designs, *J. Combin. Theory Ser. A* 43 (1986), 237–243.

[126] D. L. Kreher and S. P. Radziszowski, Finding simple t-designs by using basis reduction, *Congr. Numer.* 55 (1986), 235–244.

[127] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*, CRC Press, Boca Raton, Fla., 1999.

[128] C. W. H. Lam, How reliable is a computer-based proof? *Math. Intelligencer* 12 (1990) no. 1, 8–12.

[129] C. W. H. Lam, The search for a finite projective plane of order 10, *Amer. Math. Monthly* 98 (1991), 305–318.

[130] C. W. H. Lam, Application of group theory to combinatorial searches, *Groups and Computation* (L. Finkelstein and W. M. Kantor, Eds.), Amer. Math. Soc., Providence, R.I., 1993, pp. 133–138.

[131] C. Lam, Computer construction of block designs, *Surveys in Combinatorics, 1997* (R. A. Bailey, Ed.), Cambridge University Press, Cambridge, 1997, pp. 49–64.

[132] C. W. H. Lam, G. Kolesova, and L. Thiel, A computer search for finite projective planes of order 9, *Discrete Math.* 92 (1991), 187–195.

[133] C. W. H. Lam and L. Thiel, Backtrack search with isomorph rejection and consistency check, *J. Symbolic Comput.* 7 (1989), 473–485.

[134] C. W. H. Lam, L. Thiel, and S. Swiercz, The nonexistence of code words of weight 16 in a projective plane of order 10, *J. Combin. Theory Ser. A* 42 (1986), 207–214.

[135] C. W. H. Lam, L. Thiel, and S. Swiercz, The nonexistence of finite projective planes of order 10, *Canad. J. Math.* 41 (1989), 1117–1123.

[136] C. W. H. Lam, L. Thiel, S. Swiercz, and J. McKay, The nonexistence of ovals in a projective plane of order 10, *Discrete Math.* 45 (1983), 319–321.

[137] C. Lam and V. D. Tonchev, Classification of affine resolvable 2-$(27, 9, 4)$ designs, *J. Statist. Plann. Inference* 56 (1996), 187–202. Corrigendum appears in [*J. Statist. Plann. Inference* 86 (2000), 277–278].

[138] E. S. Lander, *Symmetric Designs: An Algebraic Approach*, Cambridge University Press, Cambridge, 1983.

[139] R. Laue, Construction of combinatorial objects – a tutorial, *Bayreuth. Math. Schr.* 43 (1993), 53–96.

[140] R. Laue, Constructing objects up to isomorphism, simple 9-designs with small parameters, *Algebraic Combinatorics and Applications* (A. Betten, A. Kohnert, R. Laue, and A. Wassermann, Eds.), Springer-Verlag, Berlin, 2001, pp. 232–260.

[141] R. Laue, Solving isomorphism problems for t-designs, *Designs 2002: Further Computational and Constructive Design Theory* (W. D. Wallis, Ed.), Kluwer, Boston, 2003, pp. 277–300.

[142] A. K. Lenstra, J. H. W. Lenstra, and L. Lovász, Factoring polynomials with rational coefficients, *Math. Ann.* 261 (1982), 515–534.

[143] J. S. Leon, An algorithm for computing the automorphism group of a Hadamard matrix, *J. Combin. Theory Ser. A* 27 (1979), 289–306.

[144] J. S. Leon, Computing automorphism groups of error-correcting codes, *IEEE Trans. Inform. Theory* 28 (1982), 496–511.

[145] J. S. Leon, Computing automorphism groups of combinatorial objects, *Computational Group Theory* (M. D. Atkinson, Ed.), Academic Press, London, 1984, pp. 321–335.

[146] J. S. Leon, Permutation group algorithms based on partitions, I: Theory and algorithms, *J. Symbolic Comput.* 12 (1991), 533–583.

[147] J. S. Leon, Partitions, refinements, and permutation group computation, *Groups and Computation, II* (L. Finkelstein and W. M. Kantor, Eds.) Amer. Math. Soc., Providence, R.I., 1997, pp. 123–158.

[148] M. J. Letourneau and S. K. Houghten, Optimal ternary $(10, 7)$ error-correcting codes, *Congr. Numer.* 155 (2002), 71–80.

[149] M. J. Letourneau and S. K. Houghten, Optimal ternary $(11, 7)$ and $(14, 10)$ error-correcting codes, *J. Combin. Math. Combin. Comput.* 51 (2004), 159–164.

[150] J. H. van Lint, *Introduction to Coding Theory*, 3rd ed., Springer-Verlag, Berlin, 1999.

[151] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, 2nd ed., Cambridge University Press, Cambridge, 2001.

[152] S. Litsyn and A. Vardy, The uniqueness of the Best code, *IEEE Trans. Inform. Theory* 40 (1994), 1693–1698.

[153] E. M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *J. Comput. System Sci.* 25 (1982), 42–65.

[154] E. M. Luks, Permutation groups and polynomial-time computation, *Groups and Computation* (L. Finkelstein and W. M. Kantor, Eds.), Amer. Math. Soc., Providence, R.I., 1993, pp. 139–175.

[155] M. M-Noori and B. Tayfeh-Rezaie, Backtracking algorithm for finding t-designs, *J. Combin. Des.* 11 (2003), 240–248.

[156] S. MacLane, *Categories for the Working Mathematician*, 2nd ed., Springer-Verlag, New York, 1998.

[157] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.

[158] F. J. MacWilliams, N. J. A. Sloane, and J. G. Thompson, On the existence of a projective plane of order 10, *J. Combin. Theory Ser. A* 14 (1973), 66–78.

[159] S. S. Magliveras and D. W. Leavitt, Simple 6-$(33, 8, 36)$ designs from $P\Gamma L_2(32)$, *Computational Group Theory* (M. D. Atkinson, Ed.), Academic Press, London, 1984, pp. 337–352.

[160] F. Margot, Small covering designs by branch-and-cut, *Math. Program.* 94B (2003), 207–220.

[161] R. Mathon, Symmetric $(31, 10, 3)$ designs with nontrivial automorphism group, *Ars. Combin.* 25 (1988), 171–183.

[162] R. Mathon, Computational methods in design theory, *Surveys in Combinatorics, 1991* (A. D. Keedwell, Ed.), Cambridge University Press, Cambridge, 1991, pp. 101–117. Reprinted in [*Computational and Constructive Design Theory* (W. D. Wallis, Ed.), Kluwer, Dordrecht, the Netherlands, 1996, pp. 29–48].

[163] R. Mathon and D. Lomas, A census of 2-$(9, 3, 3)$ designs, *Australas. J. Combin.* 5 (1992), 145–158.

[164] R. Mathon and A. Rosa, Some results on the existence and enumeration of BIBD's, Mathematics Report 125-Dec-1985, Department of Mathematics and Statistics, McMaster University, Hamilton, 1985.

[165] R. Mathon and A. Rosa, Tables of parameters of BIBDs with $r \leq 41$ including existence, enumeration, and resolvability results, *Ann. Discrete Math.* 26 (1985), 275–307.

[166] R. Mathon and A. Rosa, 2-(v, k, λ) designs of small order, *The CRC Handbook of Combinatorial Designs* (C. J. Colbourn and J. H. Dinitz, Eds.), CRC Press, Boca Raton, Fla., 1996, pp. 3–41.

[167] B. D. McKay, Hadamard equivalence via graph isomorphism, *Discrete Math.* 27 (1979), 213–214.

[168] B. D. McKay, Practical graph isomorphism, *Congr. Numer.* 30 (1981), 45–87.

[169] B. D. McKay, *nauty* user's guide (version 1.5), Technical Report TR-CS-90-02, Computer Science Department, Australian National University, Canberra, 1990.

[170] B. D. McKay, Isomorph-free exhaustive generation, *J. Algorithms* 26 (1998), 306–324.

[171] B. D. McKay, A. Meynert, and W. Myrvold, Small Latin squares, quasigroups, and loops, preprint.

[172] B. McKay, W. Myrvold, and J. Nadon, Fast backtracking principles applied to find new cages, *Proc. Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, (San Francisco, Jan. 25–27, 1998), ACM Press, New York, 1998, pp. 188–191.

[173] B. D. McKay and S. P. Radziszowski, The nonexistence of 4-$(12,6,6)$ designs, *Computational and Constructive Design Theory* (W. D. Wallis, Ed.), Kluwer, Dordrecht, the Netherlands, 1996, pp. 177–188.

[174] B. D. McKay and S. P. Radziszowski, Towards deciding the existence of 2-$(22,8,4)$ designs, *J. Combin. Math. Combin. Comput.* 22 (1996), 211–222.

[175] B. D. McKay and E. Rogoyski, Latin squares of order 10, *Electron. J. Combin.* 2 (1995), #N3, 4pp.

[176] N. S. Mendelsohn and S. H. Y. Hung, On the Steiner systems $S(3,4,14)$ and $S(4,5,15)$, *Util. Math.* 1 (1972), 5–95.

[177] M. Meringer, Erzeugung regulärer Graphen, MSc Thesis, Universität Bayreuth, 1996.

[178] M. Meringer, Fast generation of regular graphs and construction of cages, *J. Graph Theory* 30 (1999), 137–146.

[179] G. L. Miller, On the $n^{\log n}$ isomorphism technique (a preliminary report), *Proc. 10th ACM Symposium on Theory of Computing*, (San Diego, May 1–3, 1978), ACM Press, New York, 1978, pp. 51–58.

[180] G. L. Miller, Graph isomorphism, general remarks, *J. Comput. System Sci.* 18 (1979), 128–142.

[181] T. Miyazaki, The complexity of McKay's canonical labeling algorithm, *Groups and Computation, II* (L. Finkelstein and W. M. Kantor, Eds.) Amer. Math. Soc., Providence, R.I., 1997, pp. 239–256.

[182] L. B. Morales and C. Velarde, A complete classification of $(12,4,3)$-RBIBDs, *J. Combin. Des.* 9 (2001), 385–400.

[183] L. B. Morales and C. Velarde, Enumeration of resolvable 2-$(10,5,16)$ and 3-$(10,5,6)$ designs, 13 (2005), 108–119.

[184] S. Niskanen and P. R. J. Östergård, Cliquer user's guide, Version 1.0, Technical Report T48, Communications Laboratory, Helsinki University of Technology, Espoo, 2003.

[185] H. W. Norton, The 7×7 squares, *Annals of Eugenics* 9 (1939), 269–307.

[186] P. R. J. Östergård, Constructing combinatorial objects via cliques, *Surveys in Combinatorics, 2005* (B. Webb, Ed.), Cambridge University Press, Cambridge, to appear.

[187] P. R. J. Östergård, Classification of binary/ternary one-error-correcting codes, *Discrete Math.* 223 (2000), 253–262.

[188] P. R. J. Östergård, Enumeration of 2-$(12,3,2)$ designs, *Australas. J. Combin.* 22 (2000), 227–231.

[189] P. R. J. Östergård, There are 270,474,142 nonisomorphic 2-$(9,4,6)$ designs, *J. Combin. Math. Combin. Comput.* 37 (2001), 173–176.

[190] P. R. J. Östergård, Classifying subspaces of Hamming spaces, *Des. Codes Cryptogr.* 27 (2002), 297–305.

[191] P. R. J. Östergård, A fast algorithm for the maximum clique problem, *Discrete Appl. Math.* 120 (2002), 195–205.

[192] P. R. J. Östergård, A 2-$(22, 8, 4)$ design cannot have a 2-$(10, 4, 4)$ subdesign, *Des. Codes Cryptogr.* 27 (2002), 257–260.

[193] P. R. J. Östergård, T. Baicheva, and E. Kolev, Optimal binary one-error-correcting codes of length 10 have 72 codewords, *IEEE Trans. Inform. Theory* 45 (1999), 1229–1231.

[194] P. R. J. Östergård and P. Kaski, Enumeration of 2-$(9, 3, \lambda)$ designs and their resolutions, 27 (2002), 131–137.

[195] P. R. J. Östergård and P. Kaski, Enumeration of 2-$(9, 3, \lambda)$ designs and their resolutions, *Des. Codes Cryptogr.* 27 (2002), 131–137.

[196] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Mass., 1994.

[197] P. M. Pardalos, T. Mavridou, and J. Xue, The graph coloring problem: A bibliographical survey, *Handbook of Combinatorial Optimization* (D.-Z. Du and P. M. Pardalos, Eds.), Vol. 2, Kluwer, Dordrecht, the Netherlands, 1998, pp. 331–395.

[198] M.-O. Pavčević, Symmetric designs of Menon series admitting an action of Frobenius groups, *Glas. Mat. Ser. III* 31(51) (1996), 209–223.

[199] E. Petrank and R. M. Roth, Is code equivalence easy to decide? *IEEE Trans. Inform. Theory* 43 (1997), 1602–1604.

[200] C. Pietsch, *Über die Enumeration von Inzidenzstrukturen*, PhD Thesis, Universität Rostock, 1993.

[201] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 3rd ed., Wiley, New York, 1998.

[202] V. S. Pless and W. C. Huffman, Eds., *Handbook of Coding Theory*, 2 vols., Elsevier, Amsterdam, 1998.

[203] P. W. Purdom, Tree size by partial backtracking, *SIAM J. Comput.* 7 (1978), 481–491.

[204] P. W. Purdom, Jr. and C. A. Brown, *The Analysis of Algorithms*, Holt, Rinehart & Winston, New York, 1985.

[205] R. C. Read, Every one a winner; or, How to avoid isomorphism search when cataloguing combinatorial configurations, *Ann. Discrete Math.* 2 (1978), 107–120.

[206] R. C. Read and D. G. Corneil, The graph isomorphism disease, *J. Graph Theory* 1 (1977), 339–363.

[207] G. H. J. van Rees, $(22, 33, 12, 8, 4)$-BIBD, an update, *Computational and Constructive Design Theory* (W. D. Wallis, Ed.), Kluwer, Dordrecht, the Netherlands, 1996, pp. 337–357.

[208] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice Hall, Englewood Cliffs, N.J., 1977.

[209] J. J. Rotman, *An Introduction to the Theory of Groups*, 4th ed., Springer-Verlag, New York, 1995.

[210] G. F. Royle, An orderly algorithm and some applications in finite geometry, *Discrete Math.* 185 (1998), 105–115.

[211] A. Sade, An omission in Norton's list of 7×7 squares, *Ann. Math. Statistics* 22 (1951), 306–307.

[212] C. Savage, A survey of combinatorial Gray codes, *SIAM Rev.* 39 (1997), 605–629.

[213] B. Schmalz, Verwendung von Untergruppenleitern zur Bestimmung von Doppelnebenklassen, *Bayreuth. Math. Schr.* 31 (1990), 109–143.

[214] B. Schmalz, t-Designs zu vorgegebener Automorphismengruppe, *Bayreuth. Math. Schr.* 41 (1992), 1–164.

[215] B. Schmalz, The t-designs with prescribed automorphism group, new simple 6-designs, *J. Combin. Des.* 1 (1993), 125–170.

[216] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, Chichester, England, 1986.

[217] E. Seah and D. R. Stinson, An enumeration of nonisomorphic one-factorizations and Howell designs for the graph K_{10} minus a one-factor, *Ars. Combin.* 21 (1986), 145–161.

[218] E. Seah and D. R. Stinson, On the enumeration of one-factorizations of complete graphs containing prescribed automorphism groups, *Math. Comp.* 50 (1988), 607–618.

[219] N. V. Semakov and V. A. Zinov'ev, Equidistant q-ary codes with maximal distance and resolvable balanced incomplete block designs, *Problemy Peredachi Informatsii* 4 (1968) no. 2, 3–10. Translated from Russian in [*Problems Inform. Transmission* 4 (1968) no. 2, 1–7].

[220] Á. Seress, *Permutation Group Algorithms*, Cambridge University Press, Cambridge, 2003.

[221] E. Spence, A complete classification of symmetric $(31, 10, 3)$ designs, *Des. Codes Cryptogr.* 2 (1992), 127–136.

[222] E. Spence, Symmetric $(41, 16, 6)$-designs with a nontrivial automorphism of odd order, *J. Combin. Des.* 1 (1993), 193–211.

[223] E. Spence, Classification of Hadamard matrices of order 24 and 28, *Discrete Math.* 140 (1995), 185–243.

[224] E. Spence, Construction and classification of combinatorial designs, *Surveys in Combinatorics, 1995* (P. Rowlinson, Ed.), Cambridge University Press, Cambridge, 1995, pp. 191–213.

[225] E. Spence, The complete classification of Steiner systems $S(2, 4, 25)$, *J. Combin. Des.* 4 (1996), 295–300.

[226] D. Spielman, Faster isomorphism testing of strongly regular graphs, *Proc. 28th ACM Symposium on Theory of Computing*, (Philadelphia, May 22–24, 1996), ACM Press, New York, 1996, pp. 576–584.

[227] R. P. Stanley, *Enumerative Combinatorics*, Cambridge University Press, Cambridge, 1997/1999.

[228] D. R. Stinson, Isomorphism testing of Steiner triple systems: Canonical forms, *Ars. Combin.* 19 (1985), 213–218.

[229] D. R. Stinson and E. Seah, 284 457 Steiner triple systems of order 19 contain a subsystem of order 9, *Math. Comp.* 46 (1986), 717–729.

[230] J. D. Swift, Isomorph rejection in exhaustive search techniques, *Combinatorial Analysis* (R. Bellman and M. Hall, Jr., Eds.), Amer. Math. Soc., Providence, R.I., 1960, pp. 195–200.

[231] V. D. Tonchev, Hadamard matrices of order 28 with automorphisms of order 13, *J. Combin. Theory Ser. A* 35 (1983), 43–57.

[232] V. D. Tonchev, Hadamard matrices of order 28 with automorphisms of order 7, *J. Combin. Theory Ser. A* 40 (1985), 62–81.

[233] S. Topalova, Symmetric 2-$(69, 17, 4)$ designs with automorphisms of order 13, *J. Statist. Plann. Inference* 95 (2001), 335–339.

[234] S. Topalova, Classification of Hadamard matrices of order 44 with automorphisms of order 7, *Discrete Math.* 260 (2003), 275–283.

[235] R. J. Walker, An enumerative technique for a class of combinatorial problems, *Combinatorial Analysis* (R. Bellman and M. Hall, Jr., Eds.), Amer. Math. Soc., Providence, R.I., 1960, pp. 91–94.

[236] W. D. Wallis, *One-Factorizations*, Kluwer, Dordrecht, the Netherlands, 1997.

[237] W. D. Wallis, A. P. Street, and J. Seberry, *Combinatorics: Room Squares, Sum-Free Sets, Hadamard Matrices*, Springer-Verlag, Berlin, 1972.

[238] A. Wassermann, Finding simple t-designs with enumeration techniques, *J. Combin. Des.* 6 (1998), 79–90.

[239] A. Wassermann, Attacking the market split problem with lattice basis reduction, *J. Comb. Optim.* 6 (2002), 5–16.

[240] H. Wielandt, *Finite Permutation Groups*, Academic Press, New York, 1964.

[241] H. S. Wilf, *Combinatorial Algorithms: An Update*, SIAM, Philadelphia, 1989.

[242] R. M. Wilson, The necessary conditions for t-designs are sufficient for something, *Util. Math.* 4 (1973), 207–215.

[243] L. A. Wolsey, *Integer Programming*, Wiley, New York, 1998.

ISBN 951-22-7711-5
ISSN 1457-7615