Helsinki University of Technology
Department of Engineering Physics and Mathematics
Institute of Mathematics

Master's Thesis

Classification of Steiner Quadruple Systems

Olli Pottonen

Supervisor: Professor Olavi Nevanlinna
Instructor: Professor Patric Östergård

Espoo
16th November 2005

TEKNILLINEN KORKEAKOULU
TEKNILLISEN FYSIIKAN JA MATEMATIIKAN OSASTO

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Olli Pottonen

Osasto: Teknillisen fysiikan ja matematiikan osasto

Pääaine: Matematiikka

Sivuaine: Tietojenkäsittelyteoria

Työn nimi: Steinerin nelikkosysteemien luokittelu

Title in English: Classification of Steiner Quadruple Systems

Professuurin koodi ja nimi: Mat-1 Matematiikka

Työn valvoja: Professori Olavi Nevanlinna

Työn ohjaaja: Professori Patric Östergård

Työn tavoitteena on luokitella 16 pisteen Steinerin nelikkosysteemit. Kyseessä on jo jonkin aikaa avoimena olleen laskennallisen ongelman ratkaiseminen.

Työn alussa tarkastellaan Steinerin systeemeitä teoreettiselta kannalta, kuitenkin luokitteluun liittyviin tuloksiin keskittyen. Nelikkosysteemien olemassaoloa ja lukumäärää tutkitaan, kuten myös niiden yhteyttä Steinerin kolmikkosysteemeihin ja tiettyihin koodeihin. Myös Pasch–konfiguraatioita ja niiden hyödyntämistä isomorfiatarkasteluissa tarkastellaan.

McKayn kehittämä luokittelumenetelmä, kanonisilla lisäyksillä tuottaminen, esitellään varsin yleisellä tasolla. Menetelmää soveltamalla kehitetään luokittelualgoritmi Steinerin nelikkosysteemeille. Lisäksi esitetään Kasken ja Östergårdin kehittämä samankaltainen algoritmi Steinerin kolmikkosysteemeille. Myös vaihtoehtoinen, Zinovievin ja Zinovievin kehittämä luokittelumenetelmä esitellään lyhyesti.

Nelikkosysteemejä tuotettaessa ja isomorfiakarsintaa suoritettaessa kohdataan kaksi vaikeaa osaongelmaa: täsmällisten peitteiden etsiminen tietyille joukoille ja systeemeiden kanonisen nimeämisen laskeminen. Näitä ongelmia ja niiden vaativuutta tarkastellaan.

Vaikka käytettävän algoritmin oikeellisuus on todistettu matemaattisella tarkkuudella, voi ohjelmointivirhe johtaa virheellisiin tuloksiin. Tällaisten mahdollisten virheiden havaitsemiseksi testattiin laskennan tulosten johdonmukaisuutta.

Luokittelun tuloksena saatiin yksi edustaja jokaisesta 16 pisteen Steinerin nelikkosysteemien isomorfialuokasta. Isomorfialuokkia on yhteensä 1,054,163 kappaletta. Luokkien edustajia tutkimalla saatiin selville joitain uusia tuloksia, kuten resolvoitumattoman 16 pisteen Steinerin nelikkosysteemin olemassaolo.

Sivumäärä: 59 **Avainsanat:** Steinerin nelikkosysteemit, luokittelu, isomorfiakarsinta, johdetut sommitelmat, Steinerin systeemit

Täytetään osastolla

Hyväksytty: **Kirjasto:**

Author:	Olli Pottonen
Department:	Department of Engineering Physics and Mathematics
Major subject:	Mathematics
Minor subject:	Theoretical Computer Science

Title:	Classification of Steiner Quadruple Systems
Title in Finnish:	Steinerin nelikkosysteemien luokittelu
Chair:	Mat-1 Mathematics
Supervisor:	Professor Olavi Nevanlinna
Instructor:	Professor Patric Östergård

The goal of this thesis is to classify Steiner quadruple systems with 16 points and thereby solve a computational problem that has been open for some time.

In the beginning of this thesis the Steiner quadruple systems are studied from theoretical point of view, however with emphasis on results relevant to the classification. The existence and number of quadruple systems are considered, as well as their connection to Steiner triple systems and certain codes. Also Pasch configuration and their utilization in isomorphism consideration are studied.

A classification method due to McKay, generation by canonical augmentation, is presented on a rather general level. By applying the method a classification algorithm is developed for Steiner quadruple systems. Also a similar algorithm due to Kaski and Östergård for classification of Steiner triple systems is presented. In addition an alternative classification algorithm by Zinoviev and Zinoviev is briefly discussed.

When the quadruple systems are generated and the isomorph rejection carried out, two difficult subproblems are encountered: computing exact covers for certain sets and computing canonical labellings of systems. These problems and their computational complexity are studied.

Although correctness of the algorithm is proved mathematically, an error in the software could lead to erroneous results. To discover potential error of this type a consistency check was performed on the classification data.

As a result of the classification, a representative from each isomorphism class of Steiner quadruple systems with 16 points was obtained. There are total 1,054,163 isomorphism classes. By studying the class representatives some new results were discovered, for example that a non-resolvable Steiner quadruple systems with 16 points exists.

Number of pages: 59 **Keywords:** Steiner quadruple systems, classification, isomorph rejection, derived designs, Steiner systems

Department fills

Approved: **Library code:**

Preface

My work in the field of combinatorics has been really interesting. Especially solving open research problems has been rewarding. I would like to thank my instructor professor Patric Östergård for his expert guidance on my work on this field. Also Petteri Kaski deserves thanks for his involvement in guiding my work.

I would also like to thank professor Olavi Nevanlinna for his supervision and interest in my work.

Actually carrying out the classification was computationally very intensive task, and would not have been possible had not the Computing Centre of Helsinki University of Technology provided necessary computing resources. Thus I am grateful to Juhani Markula and Jukka Seppänen for making this work possible.

Espoo 16th November 2005

Olli Pottonen

Contents

Chapter 1

Introduction

Combinatorics is a branch of discrete mathematics dealing with arrangements of finite sets satisfying certain conditions. This definition is rather broad but the examples in this thesis hopefully give the reader a good picture of typical combinatorial problems.

The earliest results of combinatorics were motivated by recreational problems such as the well-known Thomas P. Kirkman's schoolgirl problem published in the *Lady's and Gentleman's Diary* of 1850:

> Fifteen young ladies in a school walk out three abreast for seven days in succession: it is required to arrange them daily, so that no two walk twice abreast.

Arrangements satisfying these conditions exist and are nowadays known as Kirkman triple systems of order 15, see Section 2.2. If the girls go to walks three at a time instead of all concurrently, each of them seven times and no two walk twice abreast, the arrangement is known as Steiner triple systems. Both systems can be generalized for other numbers of girls.

Along the years practical applications for these kinds of systems have been found in the design of statistical experiments and in other fields as well. Combinatorics also has connections to other areas of discrete mathematics and applications in computer science and cryptography. Perhaps most important is the connection to coding theory, a branch of discrete mathematics motivated by the problem of finding data encodings that can be correctly decoded in spite of occasional errors caused by a noisy transmission channel.

One of the central problems in combinatorics is classification of different kinds of designs, i. e., finding up to equivalence all designs of a certain kind. Results of classification can be useful for practical problems such as finding a design with some desired property, as well as for theoretical ones, such as finding further evidence or counterexamples for conjectures or establishing (non)existence

results.

Classification problems are closely related to counting problems, where the task is to determine the number of equivalence classes of designs with prescribed properties. A third important type of problems are the existence problems, in which the task is to determine whether a design with given properties exists. For the designs considered in this work the existence problem has been completely solved but the counting and classification problems have not. Complete classifications and counts, obtained with computer searches, are known only for few parameter values.

Often classification and sometimes also existence problems can only be solved with computers. Unfortunately computers and software are not perfectly reliable, but reliable enough for most tasks such as approximative numerical calculations. However the reliability and verifiability requirements sets for mathematical proofs are higher, and the use of computers for proving theorems is problematic. In this thesis a classification result is obtained with use of computers. Also in this task the reliability of computers should be addressed since we wish to obtain an exact solution that can not be easily verified.

In this work the classification problem is solved for Steiner quadruple systems of order 16. This task, which was considered probably impossible in 1992 [13], was indeed possible with today's fast computers and modern algorithms. Still the task was laborous and required approximately 12 years of CPU time.

The main contribution of this thesis is developing an classification algorithm for Steiner quadruple systems, and applying the algorithm for order 16. Most of the other results are taken from literature and some have been generalized. Also some of the theoretical results such as #P-completeness of #Exact cover (Theorem 20), a bound for automorphism group orders (Theorem 9) and the contents of Section 4.6 are apparently new.

The thesis is organized as follows. In Chapter 2, the relevant combinatorial structures are introduced, and their connections and other properties are discussed, with emphasis on results that are useful considering the classification.

Chapter 3 deals with the general classification framework of generation by canonical augmentation, and its application to Steiner systems. In addition a method for classification of Steiner quadruple systems with limited rank is briefly discussed.

In Chapter 4 deals with two computational problems that occur frequently in combinatorics and need to be solved as subproblems of the classification, namely the exact cover problem and computing canonical labellings.

In Chapter 5 the most important computational results and arguments for their correctness are given.

Chapter 6 concludes the thesis by discussing the results and possible further

research.

Appendix A contains a brief introduction to group theory, which is very useful in studying isomorphisms. In appendix B basic concepts of computational complexity are introduced.

Chapter 2

Combinatorial Designs

2.1 Incidence Structures

Combinatorial structures such as Steiner quadruple systems have several possible representations. Below two of these are defined. We will alternate between these two depending on their suitability for different tasks.

Definition 1. An *incidence structure* is a triple (P, B, I) where P and B are sets and $I \subseteq P \times B$ is a relation. The elements of P are called *points* and the elements of B *blocks*. If $(p, B) \in I$, then the point p and block B are said to be incident.

If $\mathcal{X}$ is an incidence structure, then we may write $P(\mathcal{X})$, $B(\mathcal{X})$ and $I(\mathcal{X})$ for the point set, block set and incidence relation, respectively. We say that the set $P' \subseteq P$ is incident to a block B if every $p \in P'$ is incident to B.

Definition 2. A *set system* is a pair $(P, \mathcal{B})$ where P is a set and $\mathcal{B}$ is a collection of subsets of P. The elements of P are called *points* and the elements of $\mathcal{B}$ blocks. A point p and a block B are said to be *incident* if $p \in B$.

If two blocks are incident to exactly the same points, they are called *repeated blocks*. A design that does not contain repeated blocks is called *simple*. Non-simple set systems are possible when the block set is considered as a multiset.

Obviously incidence structures and set systems are rather similar structures. Each incidence structure has a unique corresponding set system. Conversely each set system has a corresponding incidence structure, but not a unique one since the blocks can be labelled in several ways (unless the block set is empty). On some occasions the labelling of the blocks may contain useful additional information whereas sometimes it may be confusing that the blocks can be labelled in several different ways.

During the rest of this work we will consider incidence structures and set systems

as different representations for same objects. For example the Definitions 3, 4 and 5 concern incidence structures but are also applicable to set systems.

In set system representation two blocks that are incident to a common point are intersecting sets. Accordingly also blocks of an incidence structure are called intersecting if they are incident to at least one common point.

Definition 3. The *dual* of an incidence structure $\mathcal{X}$ is the incidence structure $\mathcal{X}^*$ where $P(\mathcal{X}^*) = B(\mathcal{X})$, $B(\mathcal{X}^*) = P(\mathcal{X})$ and $I(\mathcal{X}^*) = \{(B, p) \mid (p, B) \in I(\mathcal{X})\}$.

Definition 4. An incidence structure $\mathcal{Y}$ is a *substructure* of an incidence structure $\mathcal{X}$ if $P(\mathcal{Y}) \subseteq P(\mathcal{X})$, $B(\mathcal{Y}) \subseteq I(\mathcal{X})$ and $I(\mathcal{Y}) \subseteq I(\mathcal{X})$. The substructure is proper if $\mathcal{Y} \neq \mathcal{X}$.

Definition 5. A *derived design* associated with an incidence structure $\mathcal{X}$ and a point $p \in P(\mathcal{X})$ is the substructure $\mathcal{X}_p$ of $\mathcal{X}$, where

$$P(\mathcal{X}_p) = P(\mathcal{X}) \setminus \{p\} \tag{2.1.1}$$

$$B(\mathcal{X}_p) = \{B \in B(\mathcal{X}) \mid (p, B) \in I(\mathcal{X})\}, \tag{2.1.2}$$

and $I(\mathcal{X}_p)$ is $I(\mathcal{X})$ restricted to $P(\mathcal{X}_p) \times B(\mathcal{X}_p)$.

Informally a derived design is obtained by deleting all blocks not incident with p and removing the point p from all remaining blocks.

It is commonly assumed that the point and block sets are finite, although some research concerning infinite structures has been carried out. In this thesis only finite structures are considered.

Definition 6. Two incidence structures $\mathcal{X}$ and $\mathcal{Y}$ are *isomorphic*, denoted $\mathcal{X} \cong \mathcal{Y}$, if there exist a pair of bijections, (f_P, f_B), $f_P : P(\mathcal{X}) \to P(\mathcal{Y})$, $f_B : B(\mathcal{X}) \to B(\mathcal{Y})$ such that $(p, B) \in I(\mathcal{X})$ if and only if $(f_P(p), f_B(B)) \in I(\mathcal{Y})$. Such a pair is called an *isomorphism*.

Some group actions are useful for studying isomorphisms. Let $\mathcal{X}$ be an incidence structure with $P(\mathcal{X}) = P$ and $B(\mathcal{X}) = B$. Action of $(g, h) \in S_P \times S_B$ (see Appendix A for definitions) is defined as

$$(g, h) * \mathcal{X} = (g, h) * (P, B, I(\mathcal{X})) := (g * P, h * B, (g, h) * I(\mathcal{X})), \tag{2.1.3}$$

where

$$(g, h) * I(\mathcal{X}) := \{(g * P, h * B) \mid (P, B) \in I(\mathcal{X})\}. \tag{2.1.4}$$

Action of $g \in S_P$ on the points is $g *_P \mathcal{X} := (g, e) * \mathcal{X}$ where $e \in S_B$ is the identity permutation. Similarly action of $h \in S_B$ on the blocks is $h *_B \mathcal{X} := (e, h) * \mathcal{X}$ where $e \in S_P$ is the identity permutation.

Now assuming that $P(\mathcal{X}) = P(\mathcal{Y}) = P$ and $B(\mathcal{X}) = B(\mathcal{Y}) = B$, the two incidence structures $\mathcal{X}$ and $\mathcal{Y}$ are isomorphic if and only if $g * \mathcal{X} = \mathcal{Y}$ for some $g \in S_{\mathcal{P}} \times S_{\mathcal{B}}$.

The action on points is defined analogously for set systems: if $\mathcal{X} = (P, \mathcal{B})$ is a set system and $g \in S_P$, then the action is defined as $g * \mathcal{X} := (P, g * \mathcal{B})$. Any action that permutes the blocks cannot be meaningfully defined for set systems.

Definition 7. If incidence structures $\mathcal{X}$ and $\mathcal{Y}$ are isomorphic so that f_P is the identity mapping, we say that $\mathcal{X}$ and $\mathcal{Y}$ are *point equivalent*. Analogously they are *block equivalent* if f_B is the identity mapping.

Two incidence structures are point equivalent if the corresponding set systems are equal. Since forming the set system corresponding to a given incidence structure is straightforward, deciding point equivalence is a computationally easy problem, whereas deciding isomorphism is not (see Section 4.2).

Two incidence structures $\mathcal{X}$ and $\mathcal{Y}$ with $P(\mathcal{X}) = P(\mathcal{Y}) = P$ are isomorphic if and only if there exists $g \in S_P$ such that $g *_P \mathcal{X}$ and $\mathcal{Y}$ are point equivalent.

Definition 8. The *automorphism group* of an incidence structure $\mathcal{X}$, is the stabilizer of $\mathcal{X}$ in the group $S_{P(\mathcal{X})} \times S_{B(\mathcal{X})}$, i. e., it consists of all isomorphisms from $\mathcal{X}$ onto itself. The *point automorphism group* of $\mathcal{X}$ consists of all $g \in S_{P(\mathcal{X})}$ such that (g, f) belongs to the automorphism group of $\mathcal{X}$ for some f. The automorphism group of a set system $(P, \mathcal{B})$ is its stabilizer in the group S_P.

The automorphism group of $\mathcal{X}$ is denoted by $\mathrm{Aut}(\mathcal{X})$.

The automorphism group of a set system is isomorphic to the point automorphism group of a corresponding incidence structure. If an incidence structure is simple, its automorphism group and point automorphism group are isomorphic. This thesis deals with simple structures, and we need not worry about the distinction between automorphism group of a set system, the automorphism group of a corresponding incidence structure and the point automorphism group of the incidence structure.

During the computational part we assume that the point and block sets of all structures are subsets of $\mathbb{N}$. We denote the set off all such incidence structures by $\mathscr{S}_{\mathbb{N}}$. From a computational point of view this representation is quite simple and easy to work with. Also all incidence structures do not constitute a set in formal set theory, whereas $\mathscr{S}_{\mathbb{N}}$ does. In more theoretical parts of this work the possible point and block sets are not limited in this way.

2.2 Steiner Systems

Definition 9. A t-(v, k, λ) design, $0 < t \leq k \leq v$, $0 < \lambda$ is an incidence structure with v points such that each block is incident to k points, and each set of t distinct points is incident to exactly λ blocks.

Definition 10. A *Steiner system* $S(t, k, v)$ is a t-$(v, k, 1)$-design.

The parameter v is called the *order* of the design, and t-(v, k, λ) designs are called t-designs for short. Steiner systems are an important special case of t-designs. Another important special case are *balanced incomplete block designs*, BIBDs for short, which are 2-(v, k, λ) designs. Unlike some t-designs with higher λ, all Steiner systems are simple. Furthermore all duals of Steiner systems with $2 \leq t$ and $k < v$ are also simple, and in these cases the automorphism group and point automorphism group of a incidence structure are equal.

The main focus of this work is on *Steiner quadruple systems*, i. e., $S(3, 4, v)$ systems. Also *Steiner triple systems*, i. e., $S(2, 3, v)$ systems are of importance. We will abbreviate Steiner triple and quadruple systems of order v by STS(v) and SQS(v), respectively. The parameter v may be omitted when there is no need to specify the order of the systems.

Note that each derived system of a t-(v, k, λ) design is a $(t-1)$-$(v-1, k-1, \lambda)$ design. Especially each derived system of an SQS(v) is an STS($v-1$). This connection between STSs and SQSs is the reason for studying STS in this work.

Theorem 1. *For any t-(v, k, λ) design and $0 \leq l \leq t$, the number of blocks containing any fixed set of l points is*

$$b_l = \lambda \binom{v-l}{t-l} \Big/ \binom{k-l}{t-l} = \lambda \frac{(v-l)!(k-t)!}{(k-l)!(v-t)!} \tag{2.2.1}$$

Thus each t-(v, k, λ) design is also a l-(v, k, b_l) design for $1 \leq l \leq t$.

Proof. Consider set T of l points. There are $\binom{v-l}{t-l}$ sets T' such that $T \subseteq T'$ and $|T'| = t$. Each T' is incident to exactly λ blocks, and each block incident with T is incident to $\binom{k-l}{t-l}$ sets of form T'. □

The values b_0 and b_1 are of special interest. The total number of blocks in the design is b_0, and the number of blocks incident to any given point, or *replicate number*, is $r = b_1$.

The requirement that b_l must be integer for $0 \leq l \leq t$ gives necessary conditions for the existence of $S(t, k, v)$. The values of v satisfying these conditions are called *admissible*.

Theorem 2. *If a Steiner triple system of order v exists, then $v \equiv 1$ or $3 \pmod 6$.*

Proof. As stated above, the requirement that b_l be integers is a necessary existence condition. For an STS(v) these conditions are $2 \mid v-1$ and $6 \mid v(v-1)$. Only $v \equiv 0, 1$ or $3 \pmod 6$ satisfy the latter requirement, and of these only 1 or $3 \pmod 6$ satisfy both. □

Theorem 3. *If a Steiner quadruple system of order v exists, then $v \equiv 2$ or $4 \pmod 6$.*

Proof. Since a derived system of an SQS(v) is an STS($v-1$), the claim follows from Theorem 2. □

Theorem 4. *A Steiner quadruple system of order v exists if and only if $v \equiv 2$ or $4 \pmod 6$, $v \geq 4$.*

Proof. The "only if" is only a repetition of Theorem 3. Recursive construction of an SQS(v) for $v \equiv 2$ or $4 \pmod 6$, $v \geq 4$, is given by Hanani [12]. □

Theorem 5. *A Steiner triple system of order v exists if and only if $v \equiv 1$ or $3 \pmod 6$, $v \geq 3$.*

Proof. The "only if" part is a mere repetition of Theorem 2. When $v \equiv 1$ or $3 \pmod 6$, $v \geq 3$, by Theorem 4 there exists an SQS($v+1$), derived designs of which are STS(v). □

Existence of STS(v) for admissible v can be proved without resorting to Theorem 4. In fact it can be proved with much simpler proof than what is required for Theorem 4, and it was first proved by Kirkman [18] over a century before the existence problem of SQS(v) was solved.

The survey [13] contains an overview of Hanani's original proof as well as of another, more sophisticated proof, also due to Hanani. Both proofs rely on several recursive constructions. It is still an open problem to prove Theorem 4 by direct, i. e., non-recursive, construction.

Let $N(v)$ be the number of nonisomorphic Steiner quadruple systems with v points. It is well known [13] that $N(4) = 1$ (trivial), $N(8) = N(10) = 1$ (originally in [2]), and $N(14) = 4$ (originally in [31]). Prior to this work $N(16)$ was unknown but some lower bounds were known. These results are $N(16) \geq 8$ by Doyen and Vandensavel [8], $N(16) \geq 15$ by Gibbons and Mathon [10], $N(16) \geq 282$ by Gibbons, Mathon and Corneil [11], $N(16) \geq 31{,}021$ by Lindner and Rosa [21], the slightly improved result $N(16) \geq 31{,}310$ also by Lindner and Rosa [22], and $N(16) \geq 712{,}250$ by Zinoviev and Zinoviev [39, 41, 43]. The last of these results is discussed in Section 3.1. It was also established in [21] that $N(20) \geq 10^{17}$. Thus a complete classification of SQS(20) requires huge advances in either computer technology or classification algorithms.

For admissible values, $N(v)$ grows exponentially, but the exact growth rate is not known. Lenz [20] has proved the following lower bound for the asymptotic growth rate using recursive constructions.

Theorem 6. *For admissible v,*

$$\liminf_{v\to\infty} \frac{\log N(v)}{v^3} > 0 \tag{2.2.2}$$

Obtaining the following upper bound on $N(v)$ is not difficult.

Theorem 7.

$$\log N(v) \leq b_0 \log v < \frac{v^3}{24} \log v \tag{2.2.3}$$

where b_0 is the number of blocks.

Proof. Fix the point set and any total order on it. Now every SQS(v) and many other set systems can be obtained with the following algorithm: Find the lexicographically smallest uncovered triple T. Choose some v, and add $T \cup \{v\}$ to the block set. Repeat b_0 times.

The algorithm makes b_0 choices with v possibilities each time. Thus there are at most v^{b_0} SQS(v) and $\log N(v) \leq b_0 \log v$. Theorem 1 gives $b_0 = \frac{v(v-1)(v-2)}{24} < \frac{v^3}{24}$. □

By theorem 6 $N(v)$ grows exponentially. It is believed that the lower bound can be improved and $\log N(v)$ has asymptotical growth rate $v^3 \log v(1 + o(1))$ [13]. (The formulation of Theorems 6 given in [13] is slightly inaccurate.)

Let $S(v)$ be the number of nonisomorphic Steiner triple systems with v points. The known values are $S(3) = S(7) = S(9) = 1$, $S(13) = 2$, $S(15) = 80$ and $S(19) = 11{,}084{,}874{,}829$. For $v \leq 15$ the values have been obtained by hand calculation—and later the values have been verified with computers—and the value of $S(19)$ was computed by Kaski and Östergård [14]. The following theorem is from [14].

Theorem 8. *For all admissible v,*

$$(e^{-5}v)^{v^2/6} \leq S(v) \leq (e^{-1/2}v)^{v^2/6}. \tag{2.2.4}$$

Every derived system of an SQS(16) is an STS(15), as noted above. The converse result, that every STS(15) is a derived system of a SQS(16), also holds. This result was completed by Diener, Schmitt and De Vries [7]. The result was based on computer searches, and it is an interesting open question whether similar result holds for all admissible v. For every v such that STS(v) exist, also SQS($v+1$) exist, and it is plausible that every STS(v) is a derived system.

Definition 11. For an SQS $\mathcal{S}$, $\beta(\mathcal{S})$ denotes the number of pairwise nonisomorphic derived systems.

Clearly $1 \leq \beta(\mathcal{S}) \leq v$. An SQS($v$) $\mathcal{S}$ is called *heterogeneous* if $\beta(\mathcal{S}) = v$ and *homogeneous* if $\beta(\mathcal{S}) = 1$. It is known that at least 69 of the 80 STS(15) occur in homogeneous SQS and at least one does not [35]. Not much research has been carried out concerning homogeneous and heterogeneous SQSs.

While attempting a classification of Steiner quadruple systems, it may be practical to have a priori bound for the automorphism group order of the systems. The following theorem gives such a bound.

Theorem 9. *Assume that $\mathcal{S}$ is a Steiner system with v points and $\mathcal{S}'$ is one of its derived systems. Then $|\mathrm{Aut}(\mathcal{S})| \leq v|\mathrm{Aut}(\mathcal{S}')|$ where equality may hold only if the action of $\mathrm{Aut}(\mathcal{S})$ on the point set is transitive.*

Proof. Assume that $\mathcal{S}$ has the point set $\mathcal{P} = \{p_1, p_2, \ldots, p_v\}$ and that $\mathcal{S}'$ is the derived system associated with the point p_1.

Now $\mathrm{Aut}(\mathcal{S}) \subseteq S_{\mathcal{P}}$ and $\mathrm{Aut}(\mathcal{S}') \subseteq S_{\mathcal{P}'}$, where $\mathcal{P}' = \mathcal{P} \setminus \{p_1\}$. $\mathrm{Aut}(\mathcal{S}')$ can be considered as a subgroup of $S_{\mathcal{P}}$ by defining $g(p_1) = p_1$ for every $g \in S_{\mathcal{P}'}$.

Define $G_i = \{g \in \mathrm{Aut}(\mathcal{S}) \mid g(p_i) = p_1\}$. Note that G_i may be empty for every $i \neq 1$, and every G_i is nonempty if and only if the action of $\mathrm{Aut}(\mathcal{S})$ is transitive. Now $\mathrm{Aut}(\mathcal{S}) = G_1 \cup G_2 \cup \cdots \cup G_v$. Note that any $g \in \mathrm{Aut}(\mathcal{S})$ that fixes p_1 is also an automorphism of $\mathcal{S}'$. Thus $h \in G_i$ implies $G_i \subseteq \mathrm{Aut}(\mathcal{S}') \cdot h$, and $|G_i| \leq |\mathrm{Aut}(\mathcal{S}')|$. □

The largest automorphism group order for an STS(15) is 20,160 and only one STS(15) has such an automorphism group. Thus the automorphism group order of an SQS(16) may be at most $16 \cdot 20{,}160 = 322{,}560$. Furthermore, only a homogeneous SQS with automorphisms acting transitively on the point set may have achieve the upper bound. As can be seen in Table 5.3, there is one such SQS.

A BSQS(v) is a *colored* SQS(v), that is, SQS(v) whose each point has a color assigned to it. The coloring is *proper* if each block contains two points with the same color and two points with different colors. The minimum (maximum) number of colors that occur in a strict coloring of a BSQS(v) is its lower (upper) chromatic number, denoted by χ ($\bar{\chi}$). Lo Faro, Milazzo, and Tripodi have proved that for BSQS(16) $\bar{\chi} = 3$ and $\chi = 2$ or $\chi = 3$ [9].

Definition 12. Consider an incidence structure $\mathcal{X} = (P, B, I)$. A *parallel class* of $\mathcal{X}$ is a set $R \subseteq B$ such that for each $p \in P$ there is a unique $B \in R$ to which p is incident. A *resolution* of $\mathcal{X}$ is partition of B into parallel classes.

Definition 13. Two resolutions $\mathcal{R}$ and $\mathcal{R}'$ are isomorphic if there exists an isomorphism (f_P, f_B) of the associated incidence structures such that f_B maps parallel classes of $\mathcal{R}$ to parallel classes of $\mathcal{R}'$.

A design may have several nonisomorphic resolutions, or no resolutions at all. A design that has a resolution is called *resolvable*.

Each parallel class of an $S(t, k, v)$ must consist of v/k blocks. This gives a necessary condition for the existence of a resolution. The number of parallel classes is equal to the replicate number of the design. A Kirkman triple system mentioned in Chapter 1 is a resolution of a Steiner triple system.

The following lemma made practical implementation of some resolution related algorithms easier.

Lemma 10. *The blocks of an* SQS(16) *form at most* 3,185 *parallel classes.*

Proof. For SQS(16), Theorem 1 gives $b_0 = 140$, $b_1 = 35$, and $b_2 = 7$.

Consider a parallel class $\{B_1, B_2, B_3, B_4\}$. The block B_1 can be chosen in 140 ways. By inclusion-exclusion principle, there are $4(b_1 - 1) - \binom{4}{2}(b_2 - 1) = 100$ other blocks that intersect B_1, and 39 blocks that do not. Thus B_2 can be chosen in 39 ways.

Consider the blocks that intersect neither B_1 nor B_2. There are at most $\binom{8}{3}/\binom{4}{3} = 14$ of those. Accordingly B_3 can be chosen in at most 14 different ways. Since any Steiner system is simple, the choices of B_1, B_2 and B_3 determine B_4 uniquely.

Since the same parallel class can be formed in 4! different ways by choosing the same blocks in different order, the total number of parallel classes is at most $140 \times 39 \times 14 \times 1/4! = 3185$. □

2.3 Projective Spaces

In Section 2.4 Pasch configurations are defined and studied. To facilitate that study a brief overview of projective spaces is included here.

Definition 14. Let q be a prime power implying that a Galois field $\mathbb{F}_q$ exists. A projective space $PG(n, q)$ is an incidence structure having the 1-dimensional subspaces of $\mathbb{F}_q^{n+1}$ as points and the 2-dimensional subspaces as blocks. A point is incident to the blocks containing it as a subspace.

Theorem 11. *Every* $PG(n, q)$ *is an* $S(2, q+1, \frac{q^{n+1}-1}{q-1})$. *Especially a* $PG(n, 2)$ *is an* $\mathrm{STS}(2^{n+1} - 1)$.

Proof. Clearly for two non-equal one-dimensional subspaces there is a unique two-dimensional subspace that contains both of them. Hence we only need to verify that the parameters $q + 1$ and $\frac{q^{n+1}-1}{q-1}$ are correct.

The set $\mathbb{F}^* := \mathbb{F}_q^{n+1} \setminus \{0\}$ consists of $q^{n+1} - 1$ elements. Linear dependence, when considered as a binary relation, is an equivalence relation in $\mathbb{F}^*$ having equivalence classes of size $q - 1$. These equivalence classes correspond to the one-dimensional subspaces of $\mathbb{F}_q^{n+1}$. Thus the number of points of the design is $\frac{q^{n+1}-1}{q-1}$.

Now assume that $L \subseteq \mathbb{F}_q^{n+1}$ is a two-dimensional subspace spanned by linearly independent vectors $\mathbf{x}, \mathbf{y}$. Thus $L = \{\alpha\mathbf{x} + \beta\mathbf{y} \mid \alpha, \beta \in \mathbb{F}_q\}$, and we see that L consists of q^2 elements and $L^* = L \setminus \{0\}$ of $q^2 - 1$ elements. As above, by considering equivalence classes of L^*, we conclude that L contains $\frac{q^2-1}{q-1} = q + 1$ distinct one-dimensional subspaces. □

In the case $q = 2$, the result of adding any vector to itself is zero. If the vector $\mathbf{0}$ is ignored, each point of $PG(n, 2)$ is a vector in $\mathbb{F}_2^{n+1} \setminus \{0\}$ and the blocks of $PG(n, 2)$ are the sets $\{x, y, z\} \subseteq \mathbb{F}_2^{n+1} \setminus \{0\}$ such that $x + y + z = 0$.

Similarly one obtains a SQS(2^{n+1}) by taking the point set $\mathbb{Z}_2^{n+1}$ and as blocks the 4-sets $\{w, x, y, z\}$ such that $w + x + y + z = 0$. Especially this construction gives the unique SQS(8).

2.4 Pasch Configurations

Pasch configurations are easily detectable substructures of STSs that are very useful for efficiently deciding isomorphism of STSs.

Definition 15. A *Pasch configuration* (or a *quadrilateral* or a *fragment*) is a substructure of an STS that has block set of form $\{\{a, b, c\}, \{a, d, e\}, \{b, d, f\}, \{c, e, f\}\}$, and the point set $\{a, b, c, d, e, f\}$.

The Pasch configuration is the only substructure of an STS with 6 points and 4 blocks such that each block is incident with 3 points. It is also the only substructure with 6 point such that each point is incident with 2 blocks and each block with 3 points.

The number of Pasch configurations occurring in STSs varies significantly, as is shown by the following theorems.

Theorem 12. *Anti-Pasch* STS(v)*, i. e.,* STS(v) *that do not contain Pasch configurations exist for infinitely many* v*.*

There are some admissible v for which anti-Pasch systems do not exist, e. g., $v = 7$ and 13. For more details and a proof of Theorem 12, the reader is referred to [23].

The following result was pointed out by Stinson [34].

Theorem 13. *Define* $p(v) := v(v - 1)(v - 3)/24$*. Any* STS($v$) *contains at most* $p(v)$ *Pasch configurations. Furthermore a* STS(v) *contains* $p(v)$ *Pasch configurations if and only if it is a projective space* $PG(n, 2)$*. Thus for infinitely many* v *there exist* STS(v) *with* $p(v)$ *Pasch configurations.*

Proof. Choose a point a. Obviously it may be chosen in v ways. There are $(v-1)/2$ blocks that contain it and thus $(v-1)(v-3)/4$ pairs of blocks intersecting in a. Each such pair may occur in at most two Pasch configurations. Since each Pasch configuration contains six points, same configuration can be obtained with six different choices of a, and there are at most $\frac{1}{6}v(v - 1)(v - 3)/4 = v(v-1)(v-3)/24 = p(v)$ Pasch configurations, and the upper bound is obtained if and only if each pair of intersecting blocks occurs in two Pasch configurations.

Now consider a $PG(n,2)$ and two intersecting blocks, $\{\mathbf{a},\mathbf{b},\mathbf{c}\}$ and $\{\mathbf{a},\mathbf{x},\mathbf{y}\}$. By the observations in Section 2.3 we have $\mathbf{b}=\mathbf{a}+\mathbf{c}$ and $\mathbf{x}=\mathbf{a}+\mathbf{y}$. Thus $\mathbf{b}+\mathbf{x}+(\mathbf{c}+\mathbf{y})=2(\mathbf{a}+\mathbf{c}+\mathbf{y})=0$. Clearly $\mathbf{c}+\mathbf{y}+(\mathbf{c}+\mathbf{y})=0$. Thus the STS contains the blocks $\{\mathbf{b},\mathbf{x},\mathbf{c}+\mathbf{y}\}$ and $\{\mathbf{c},\mathbf{y},\mathbf{c}+\mathbf{y}\}$ that form Pasch configuration with the two original blocks. Similarly the STS must contain blocks $\{\mathbf{b},\mathbf{y},\mathbf{c}+\mathbf{x}\}$ and $\{\mathbf{c},\mathbf{x},\mathbf{c}+\mathbf{x}\}$, and the two original blocks occur in two Pasch configurations. Hence $PG(n,2)$ contains $p(v)$ Pasch configurations.

To obtain the converse implication, assume that $(P,\mathcal{B})$ is an STS(v) with $p(v)$ Pasch configurations. Thus if $\{a,b,c\}$ and $\{a,d,e\}$ are blocks of the STS, then so are $\{b,d,f\}$, $\{c,e,f\}$, $\{b,e,g\}$ and $\{c,d,g\}$, for some f,g.

Now define $G=\mathcal{P}\cup\{0\}$ and define addition in G by $p+0=0+p=p$, $p+p=0$ and for $p,q,r\in\mathcal{P}, p\neq q$, $p+q=r$ where r is chosen so that $\mathcal{B}$ contains the block $\{p,q,r\}$. Since the structure is an STS, addition of any two points is well defined. The Pasch property implies that addition is associative. It is easy to see that addition as defined above is commutative and that each element except 0 has order 2. Thus G is an additive group isomorphic to $\mathbb{Z}_2^{n+1}$ with $v=2^{n+1}-1$. By defining multiplication as $0\cdot x=0$, $1\cdot x=x$, the group G is extended to a vector space isomorphic to $\mathbb{F}_2^{n+1}$, establishing that the STS is a $PG(n,2)$. □

It seems that no detailed description of an algorithm for finding all Pasch configurations can be found in the literature. However it has long been known that all Pasch configurations of a STS(v) can be found in time $O(v^3)$ [34]. The following simple algorithm achieves this time bound.

For any two intersecting blocks $\{a,b,c\},\{a,d,e\}$, find the blocks containing the pairs $\{b,d\}$ and $\{c,e\}$. If those two blocks intersect, the four blocks form a Pasch configuration. Similarly find the blocks containing the pairs $\{b,e\}$ and $\{c,d\}$. To ensure that each Pasch configuration is counted exactly once, it is required that a is smallest of the points occurring in the configuration, with respect to some total order of the point set.

With suitable preprocessing the block containing a given pair of points can be found in constant time. Each point occurs in $O(v)$ blocks and thus there are $O(v^2)$ pairs of blocks that intersect in a. Since there are $O(v)$ choices for a, the running time of the algorithm is $O(v^3)$. Since $O(v^3)$ is the best possible upper bound for the number of Pasch configurations, the algorithm has optimal asymptotical running time for finding all Pasch configurations. It is not known whether only determining the number of Pasch configurations can be done quicker.

Note that corresponding points in isomorphic systems occur in equally many Pasch configurations, and isomorphic systems have equally many Pasch configurations.

Consider an STS(v) $\mathcal{S}$. For $p\in P(\mathcal{S})$ define $q(p)$ as the number of Pasch

configurations p occurs in. Now the multiset $q(\mathcal{S}) = \{q(p) \mid p \in P(\mathcal{S})\}$ is an easily computable invariant for STS. For $v = 15$, this invariant is *complete*, i. e., it assigns different values for non-isomorphic systems. This makes the invariant very useful.

2.5 Codes

Definition 16. Let $\mathbb{Z}_q$ be the ring of integers modulo q. A q-ary code of length n is an arbitrary non-empty subset C of $\mathbb{Z}_q^n$. The elements of C are called codewords.

We define the support of a codeword $\mathbf{x}$ similarly to the support of a function as $\operatorname{supp}(\mathbf{x}) := \{i \in \{1, 2, \ldots, n\} \mid x_i \neq 0\}$.

Definition 17. The *Hamming weight* of a codeword $\mathbf{x}$ is the number of nonzero elements and is denoted by $\operatorname{wt}(\mathbf{x})$. The metric

$$d(\mathbf{x}, \mathbf{y}) := |\{i \in \{1, 2, \ldots, n\} \mid x_i \neq y_i\}| \tag{2.5.1}$$

is called the *Hamming distance* or the *Hamming metric*.

It is easy to see that the Hamming metric satisfies the axioms of a metric. Also Hamming weight and distance are connected: $d(\mathbf{x}, \mathbf{y}) = \operatorname{wt}(\mathbf{x} - \mathbf{y})$ and $\operatorname{wt}(\mathbf{x}) = d(\mathbf{x}, \mathbf{0})$. Furthermore $\operatorname{wt}(\mathbf{x}) = |\operatorname{supp}(\mathbf{x})|$.

The minimum distance of code $C \subseteq \mathbb{Z}_q^n$ with at least two codewords is $d(C) = \min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\}$. A code has constant weight w if $\operatorname{wt}(\mathbf{x}) = w$ for every $\mathbf{x} \in C$. The cardinality of a code is simply the number of codewords, $|C|$.

A code over $\mathbb{Z}_q$ is called a *q-ary code*. A code with $q = 2$ is called *binary*.

Definition 18. An $(n, d, N)_q$ code is a q-ary code with length n, cardinality N and minimum distance d. An (n, w, d, N) code is an $(n, d, N)_2$ code with constant weight w.

Codes with specified minimum distance are often called *error-correcting codes*, since a codeword of a code with minimum distance d may be reconstructed if it has at most $\lfloor \frac{d-1}{2} \rfloor$ erroneous coordinates. Also up to $d - 1$ errors will be detected for certain, and if d is large enough, any number of errors will be detected with a high probability. Obviously error correction and detection are desirable in data transmission and recording, although mere error detection is of limited use in the latter case.

In the aforementioned applications codes should have large cardinality to maximize data throughput. Also some regularity properties of the code may be useful considering efficiency of data processing.

Considering the use of codes for data encoding it is irrelevant whether the code is subset of $\mathbb{Z}_q^n$ or subset of K^n for some K with cardinality q. If K is a Galois field $\mathbb{F}_{p^l}$, p prime, then $\mathbb{F}_{p^l}^n$ is a vector space over K. This algebraic structure may be useful. Use of the field structure is possible for some codes over $\mathbb{Z}_q^n$, since the Galois field $\mathbb{F}_p$, p prime, is equal to $\mathbb{Z}_p$.

The *rank* of a code $C \subseteq \mathbb{F}_q$, denoted by $\mathrm{rank}(C)$, is the dimension of the subspace of $\mathbb{F}_q^n$ spanned by the code. A code is *linear* if it is a subspace of $\mathbb{F}_q^n$. The inner product in $\mathbb{F}_q^n$ is defined as $\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + \cdots + x_n y_n$.

Unfortunately following the common conventions leads to using $\cdot$ to denote both group multiplication and the inner product, but it will be clear from context which operation is meant.

Definition 19. Let C be a code over $\mathbb{F}_q$. The *dual code* of C, denoted by $C^\perp$, is the code $C^\perp = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x} \cdot \mathbf{c} = 0 \text{ for every } \mathbf{c} \in C\}$.

It is easy to verify that every dual code is linear. The following theorem is a well known result of linear algebra.

Theorem 14. *For any code C over $\mathbb{F}_q^n$*

$$\mathrm{rank}(C) + \mathrm{rank}(C^\perp) = n. \tag{2.5.2}$$

Proof. Let A be a matrix having words of C as rows. A defines a linear mapping. Now the claim follows from [3, Section 14, Theorems 7.1 and 7.7]. □

A binary code $C \subseteq \mathbb{Z}_2^n$ can be represented as a incidence structure with point set $\{1, 2, \ldots, n\}$, block set C and incidence relation $I = \{(m, \mathbf{x}) \mid x_m \neq 0\}$. Thus a codeword is incident to its support. Also every incidence structure can be represented as a binary code, but not uniquely: the point set may be mapped to $\{1, 2, \ldots, n\}$ in several ways. However all choices lead to isomorphic codes.

Theorem 15. *An $S(t, k, v)$, $v > k$ corresponds to a $(v, k, 2(k - t + 1), N)$ code where $N = \binom{v}{t} / \binom{k}{t}$, and vice versa.*

Proof. Consider codewords $\mathbf{x}, \mathbf{y}$ such that $\mathrm{wt}(\mathbf{x}) = \mathrm{wt}(\mathbf{y}) = w$ and $P = \mathrm{supp}(\mathbf{x}) \cap \mathrm{supp}(\mathbf{y})$. The proof is based on the following equation:

$$d(\mathbf{x}, \mathbf{y}) = |\,\mathrm{supp}(\mathbf{x}) \setminus \mathrm{supp}(\mathbf{y})| + |\,\mathrm{supp}(\mathbf{y}) \setminus \mathrm{supp}(\mathbf{x})| = 2w - 2|P|. \tag{2.5.3}$$

The condition $v > k$ excludes the degenerate special case where the Steiner systems contains only one block and the code only one word.

Clearly an $S(t, k, v)$ corresponds to a code with constant weight k. Theorem 1 gives the cardinality N. Two blocks of a $S(t, k, v)$ have at most $t - 1$ points incident to both of them, thus for the corresponding code $|P| < t - 1$ and

$d(\mathbf{x},\mathbf{y}) \geq 2(k-t+1)$ when $\mathbf{x} \neq \mathbf{y}$ by (2.5.3). Accordingly the code has minimum distance at least $2(k-t+1)$. Since $v > k$, there are two blocks intersecting in $t-1$ points and the minimum distance is exactly $2(k-t+1)$.

Consider an incidence structure corresponding to a $(v, k, 2(k-t+1), N)$ code. The incidence structure covers each set of t points at most once, for if there would be some set P of t points incident to two blocks, the distance of the codewords corresponding to the blocks would be at most $2k - 2|P| = 2k - 2t$, a contradiction. Since $N = \binom{v}{t} / \binom{k}{t}$, the blocks cover all sets of t points. □

Especially the SQS(16) correspond to $(16, 4, 4, 140)$ codes.

Definition 20. The *rank* of an SQS is the rank of the corresponding $(v, 4, 4, b_0)$ code.

The results in [39] imply that the rank of an SQS(16) is at least 11. Since each block of an SQS is incident with an even number of points, the vector $\mathbf{v} = (1, 1, \ldots, 1)$ belongs to the dual code of any SQS(16) and the rank of an SQS(16) may be at most 15.

Chapter 3

Isomorph-Free Construction

Typically classification algorithms consists of two parts: exhaustive generation and isomorph rejection. In advanced algorithms these two parts are not independent, but rather the isomorph rejection relies on the generation being done in certain way.

Having some sort of substructures as a starting point in the exhaustive generation may enhance efficiency of the algorithm be eliminating symmetry. These substructures are then completed with some kind of exhaustive search.

In several cases some other than the most straightforward representation of the combinatorial structures is employed in a classification algorithm for efficiency reasons.

There are some general classification frameworks that can be applied for classification of all kinds of combinatorial structures. In this section we present one such framework and discuss applying it for classification of STS and SQS. Before that we briefly discuss another kind of classification algorithm developed by Zinoviev and Zinoviev. This algorithm is obtained by detailed analysis of the SQS(16) instead of applying a general framework.

3.1 The Zinoviev Classification Method

This section contains a brief overview of the classification methods with which Zinoviev and Zinoviev [39, 41] classified SQS(16) with rank at most 14. In their work several codes are used including the code representation of SQS given by Theorem 15. Before discussing classification in greater detail we need some preliminary results, which are generalizations of results mentioned in [39].

Lemma 16. *Let C be a code corresponding to an* STS(v). *Then every $\mathbf{x} \neq \mathbf{0}$ in the dual code of C has weight $(v+1)/2$.*

Note that we consider the dual code of a nonlinear code. This is an unusual, but a completely valid approach. We could equivalently study the dual code of the linear code spanned by C. These remarks apply to Theorem 17 as well.

Proof. First note that $v \equiv 1, 3 \pmod 6$ implies that $\frac{v+1}{2}$ is an integer.

Denote the coordinates of $\mathbf{x}$ by x_i, i. e., $\mathbf{x} = (x_1, x_2, \ldots, x_v)$. Since $\mathbf{x} \neq 0$, we may assume without loss of generality that $x_1 = 1$. Consider the words $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^l$ of C with the element 1 in their support. Since C corresponds to a STS, for each $i \in \{2, 3, \ldots, v\}$ there is unique $\mathbf{y}^k$ such that $i \in \operatorname{supp}(\mathbf{y}^k)$. Now assume that $\operatorname{supp}(\mathbf{y}^k) = \{1, i, j\}$. Since $\mathbf{y}^k$ is in the dual of C, $\mathbf{y}^k \cdot \mathbf{x} = x_1 + x_i + x_j = 0$ and either $x_i = 0, x_j = 1$ or $x_i = 1, x_j = 0$. Thus $\operatorname{supp}(\mathbf{x})$ contains 1 and exactly half of $\{2, 3, \ldots, v\}$, and $|\operatorname{supp}(\mathbf{x})| = 1 + \frac{v-1}{2}$. □

Theorem 17. *Assume that C is a code of a $S(t+1, t, v)$ system, $t \geq 2$, $t+1 < v$ and $\mathbf{x} \in C^\perp$. For odd t, $\operatorname{wt}(\mathbf{x}) \in \{0, \frac{v-t+3}{2}, v\}$ and for even t, $\operatorname{wt}(\mathbf{x}) \in \{0, \frac{v-t+3}{2}\}$.*

Proof. For $t = 2$ the claim follows from Lemma 16. We proceed with induction.

Assume that t is odd and that the claim holds for $t-1$. If every coordinate of $\mathbf{x}$ is equal to 1, claim holds. If $x_i = 0$, consider the derived design associated with i. By removing coordinate x_i from $\mathbf{x}$ one obtains a vector $\mathbf{x}'$ which is in dual code of the derived design. By the induction hypothesis $\operatorname{wt}(\mathbf{x}') \in \{0, \frac{(v-1)-(t-1)+3}{2}\} = \{0, \frac{v-t+3}{2}\}$. Since $\operatorname{wt}(\mathbf{x}) = \operatorname{wt}(\mathbf{x}')$, the claim holds.

Now assume that t is even and that the claim holds for $t-1$. Since $k = t+1$ is odd, $x_i = 0$ for some i. Defining $\mathbf{x}'$ as above, one concludes that $\operatorname{wt}(\mathbf{x}) = \operatorname{wt}(\mathbf{x}') \in \{0, \frac{(v-1)-(t-1)-3}{2}, v-1\} = \{0, \frac{v-t-3}{2}, v-1\}$. For the sake of contradiction assume that $\mathbf{x}$ has weight $v-1$ and that $x_i = 0$. Since $k = t+1 < v$, by (2.2.1) we have $b_1 < b_0$, and C contains at least one word $\mathbf{c}$ such that $c_i = 0$. Since $\operatorname{wt}(c_i) = k$ is odd, $\mathbf{x} \cdot \mathbf{c} = 1$, a contradiction. Thus $\operatorname{wt}(\mathbf{x}) \in \{0, \frac{v-t-3}{2}\}$. □

Corollary 18. *If C is a code corresponding to an* SQS(v)*, then for any $\mathbf{x} \in C^\perp$ $\operatorname{wt}(\mathbf{x}) \in \{0, v/2, v\}$.*

Proof. If $v = t + 1 = 4$, then the claim holds. For $v > 4$ the claim follows directly from Theorem 17. □

By applying previous results we obtain a lower bound for the minimum distance of dual codes of Steiner systems with $k = t+1$. Assume $\mathbf{x}, \mathbf{y} \in C^\perp$, $\mathbf{x} \neq \mathbf{y}$ and recall that $d(\mathbf{x}, \mathbf{y}) = \operatorname{wt}(\mathbf{x} - \mathbf{y})$. Since any dual code is linear, $0 \neq \mathbf{x} - \mathbf{y} \in C^\perp$. Theorem 17 gives us lower bound for the weight of any non-zero vector $\mathbf{z} \in C^\perp$, which is lower bound for the minimum distance of $C^\perp$ by the preceding calculation.

Now assume that C is the code of an SQS(16) with rank at most 14, in which case the rank of $C^\perp$ is at least 2. As noted in Section 2.5, the vector $\mathbf{v} = (1, 1, \ldots, 1)$ belongs to the dual code of C. Since the rank of $C^\perp$ is at least two, $C^\perp$ contains a codeword $\mathbf{x}$ that has weight 8. Thus $C^\perp$ contains the vectors $\mathbf{x}$ and $\mathbf{x} + \mathbf{v}$. Without loss of generality we may assume that the supports of those vectors are $L = \{1, 2, \ldots, 8\}$ and $R = \{9, 10, \ldots, 17\}$, respectively. Thus for each codeword $\mathbf{x} \in C$, $|\operatorname{supp}(\mathbf{x}) \cap L|$ and $|\operatorname{supp}(\mathbf{x}) \cap R|$ are both even. Thus the set C can be partitioned into sets S_0, S_2 and S_4, where $S_i = \{\mathbf{x} \in C \mid \operatorname{supp}(\mathbf{x}) \cap L = i\}$.

For two codewords $\mathbf{x}$ and $\mathbf{y}$ over $\mathbb{Z}_2^8$, denote the codeword of $\mathbb{Z}_2^{16}$ obtained by concatenating $\mathbf{x}$ and $\mathbf{y}$ by $\mathbf{x} \parallel \mathbf{y}$. Let C_1 be the code $\{\mathbf{x} \mid \mathbf{x} \parallel \mathbf{0} \in S_4\}$ and C_2 the code $\{\mathbf{x} \mid \mathbf{0} \parallel \mathbf{x} \in S_0\}$. The code C_1 corresponds to an SQS(8), as does C_2. The SQS(8) is unique (up to isomorphism) and given in Section 2.3. Thus we may without loss of generality assume that $C_1 = C_2$. To obtain complete classification, possible choices for S_2 need to be classified up to $\operatorname{Aut}(C_1)$ isomorphism.

For the rest of this section by partition we mean an ordered partition $L = (L_1, \ldots, L_7)$ of all codewords over $\mathbb{Z}_2^8$ with weight 2 such that for $\mathbf{x}$, $\mathbf{y} \in L_i$ $\operatorname{supp}(\mathbf{x}) \cap \operatorname{supp}(\mathbf{y}) = \emptyset$ unless $\mathbf{x} = \mathbf{y}$. Equivalently we may require that each L_i is a $(8, 2, 4, 4)$ code. Yet another equivalent requirement is as follows: equate codeword $\mathbf{x}$ with $\operatorname{supp}(\mathbf{x}) = \{k, l\}$ with edge $\{k, l\}$ in the complete graph with vertex set $\{1, 2, \ldots, 8\}$. Now a partition corresponds to a 1-factor of the complete graph.

Denote by $\mathbf{e}_i$ the codeword over $\mathbb{Z}_2^8$ with $\operatorname{supp}(\mathbf{e}_i) = \{i, 8\}$, $1 \leq i \leq 7$. The *heading* associated with the set S_2 is its subset $F = \{\mathbf{x} \parallel \mathbf{y} \in S_2 \mid \mathbf{x} = \mathbf{e}_i \text{ or } \mathbf{y} = \mathbf{e}_i \text{ for some i}\}$.

Up to isomorphism all headings can be formed as follows. Define the action of $\pi \in S_7$ on $(L_1, \ldots, L_7)$ by $\pi * (L_1, \ldots, L_7) = (L_{\pi^{-1}(1)}, \ldots, L_{\pi^{-1}(7)})$, and let Q_L denote the subgroup of permutations of S_7 for which the partition $\pi * L$ is $\operatorname{Aut}(C_1)$-isomorphic to L. Classify the partitions L up to $\operatorname{Aut}(C_1)$ isomorphism and the partitions L' up to $\operatorname{Aut}(C_1)$ isomorphisms that fix the point 8, and call the isomorphism class representatives thus obtained canonical partitions. Now by constructing sets of form $\{\mathbf{e}_{\pi^{-1}(i)} \parallel \mathbf{x} \mid \mathbf{x} \in L'_i\} \cap \{\mathbf{y} \parallel \mathbf{e_i} \mid \mathbf{y} \in L_{\pi^{-1}(i)}\}$, where L and L' are chosen among the canonical partitions and π from $(Q_{L'}, Q_L)$ double coset representatives of S_7, one gets a complete classification of headings.

When headings have been classified, all their extensions to a valid set S_2 were searched, and a final isomorph rejection was carried out. An invariant based on the isomorphism classes of the derived designs was employed in the classification; the same invariant is employed in the classification algorithm developed in this thesis.

If the rank of C is at most 13, duality consideration lead to four sets L_1, L_2, L_3, L_4 such that cardinality of each L_i is four and $|\operatorname{supp}(\mathbf{x}) \cap L_i|$ is even for each code-

word $\mathbf{x} \in C$. In this case the blocks can be partitioned into 3 sets S_1, S_2 and S_4 where each $\mathbf{x} \in S_i$ intersect i of the sets L_1, L_2, L_3, L_4. The structure of S_1 is trivial and unique. The code S_4 corresponds to a $(4, 2, 64)_4$ code and S_2 to a $(4, 2, 18)_4$ code with constant weight 2.

Classification of SQS(16) with rank at most 13 is achieved as follows. Classify the possible S_2 and S_4 codes. For each pair (S_2, S_4) find the $(\mathrm{Aut}(S_2), \mathrm{Aut}(S_4))$ double coset representatives in the group $\mathrm{Aut}(S_1)$, and for each representative π construct the SQS that corresponds to codes S_1, S_2 and $\pi * S_4$. Finally perform isomorph rejection.

3.2 Canonical Labelling of Incidence Structures

One way of deciding isomorphism, or any other equivalence relation, is by computing canonical representations, a concept which is formalized by the following definition.

Definition 21. For a set T and an equivalence relation $\cong$, a canonical representation is given by a function $f : T \to T$ such that, for any $s \in T$, $s \cong f(s)$ and for any $s, r \in T$, $s \cong r$ implies $f(s) = f(r)$.

Now $f(s)$ is the canonical representation of s, and $s \cong r$ if and only if $f(s) = f(r)$.

On many occasions a procedure for computing canonical representations is more useful than a procedure for deciding equivalence. As a simple example consider a set $Q \subseteq T$, $|Q| = n$. Partitioning Q into equivalence classes may require $n(n-1)/2$ calls to a procedure deciding isomorphism of two elements. However, using canonical representations only n calls of the canonical representative procedure and total $O(n \log n)$ operations are required: compute the canonical representation for each element of Q, and sort those.

In isomorphism consideration even more useful than canonical representation is the isomorphism that maps an element to its canonical representation.

Definition 22. Assume that the elements of T are equivalent if they are on the same G-orbit, that is, $a \cong b$ if $a = g * b$ for some $g \in G$. Now a function $g : S \to G$ is a canonical labelling if the function $f : S \to S$, $s \mapsto g(s) * s$ gives a canonical representation.

This definition can naturally be applied to incidence structures provided that we have a proper acting group G. The equivalence relation used here is isomorphism.

Definition 23. A function $\tau : \mathscr{S}_{\mathbb{N}} \to S_{\mathbb{N}} \times S_{\mathbb{N}}$ such that $S \cong S'$ implies that $\tau(S) * S = \tau(S') * S'$ will be called a *canonical labelling*.

A function $\tau_P : \mathscr{S}_{\mathbb{N}}$ such that $S \cong S'$ implies that $\tau_P(S) *_P S$ and $\tau_P(S') *_P S'$ are point equivalent is called a *point canonical labelling*. Block canonical labelling, denoted by τ_B, is defined analogously.

If τ_P is a point canonical labelling, one can define a canonical labelling based on it. For $S \in \mathscr{S}_{\mathbb{N}}$, consider lexicographical order of the blocks of $\tau_P(S) *_P S$ when considered as a set system. Let $\tau_B(S)$ be the function that maps the smallest block (in lexicographical order) to 1, the second smallest to 2 etc. Now $(\tau_P(S), \tau_B(S))$ is a canonical labelling of S. By similar construction one obtains a canonical labelling from a block canonical labelling.

Also each canonical labelling is of the form $\tau(S) = (\tau_P(S), \tau_B(S))$, where τ_P and τ_B are canonical labellings of points and blocks.

In Chapter 4 the canonical labelling used in the SQS classification is considered in greater detail and from a computational point of view. In practice having a canonical labelling for SQS suffices, and accordingly the algorithm uses some tailored optimizations. However a canonical labelling of arbitrary incidence structures is a nice theoretical tool, and could also be computed if necessary.

3.3 Generation by Canonical Augmentation

A sophisticated and very general classification framework was described by McKay in [30]. In this section a simpler variant of that framework is described. This variant is applicable only to classification of incidence structures. Still it is more general than what is required for the SQS classification.

As mentioned in Section 2.1, in the classification we assume that all incidence structures belong to $\mathscr{S}_{\mathbb{N}}$. Some well-order on the set $\mathscr{S}_{\mathbb{N}}$ will be needed, that is, we expect that there is a total order on $\mathscr{S}_{\mathbb{N}}$ so that each non-empty set has a smallest element. Since $\mathscr{S}_{\mathbb{N}}$ is countably infinite, the requirement is not trivial but easy to fulfill.

In the isomorphism considerations we employ the action by the group $G = S_{\mathbb{N}} \times S_{\mathbb{N}}$, for structures of $\mathscr{S}_{\mathbb{N}}$ are isomorphic if and only if they are on the same G-orbit.

The "empty" incidence structure, i. e., one that has the empty set as both point and block sets, will be denoted as ϵ. Note that ϵ is a substructure of any incidence structure.

Assume that there is a finite set $\mathscr{S}$, $\epsilon \notin \mathscr{S}$, of incidence structures of interest such that $\mathscr{S}$ is closed under the action of G. Now the classification framework requires an *extension relation* $\mathscr{R} \subseteq \mathscr{S} \times \mathscr{S}$, a *canonical substructure function* $m : S \to S \cup \{\epsilon\}$ and an *invariant function* $f : \mathscr{S} \to \mathbb{N}$ such that the following requirements hold:

(R1) $(S', S) \in \mathscr{R}$ implies that S is a proper substructure of S'.

(R2) For each $g \in G, (S', S) \in \mathscr{R}$ implies $(g * S', g * S) \in \mathscr{R}$.

(R3) For each $S \in \mathscr{S}, (m(S), S) \in \mathscr{R}$ unless $m(S) = \epsilon$.

(R4) If $S' \cong S$, there exists $g \in G$ such that $S' = g * S$ and $m(S') = g * m(S)$.

(R5) $f(g * S) = f(S)$ for every g.

(R6) If $(S', S) \in \mathscr{R}$, then $f(S') \geq f(m(S))$.

If $(S', S) \in \mathscr{R}$, then S is called an *extension* of S'. The process of finding all S for which $(S', S) \in \mathscr{R}$ is called *extending* S'. One possible way of extending substructures will be considered in Section 4.1.

The original framework by McKay did not include a invariant function. This version of the framework essentially reduces to a framework without an invariant if f is any integral constant (which clearly satisfies (R5) and (R6)).

The canonical labelling can be utilized in the definition and computation of canonical substructures as suggested by Kaski [17]. Namely, if $u : \mathscr{S} \to \mathscr{S} \cup \{\epsilon\}$ is an arbitrary function that maps each structure to some of its proper substructures, then the mapping $S \mapsto m(S) = \tau(S)^{-1} * u(\tau(S) * S)$ satisfies (R4). To see this, assume that S and S' are isomorphic, hence $\tau(S) * S = \tau(S') * S'$, which implies $\tau(S) * m(S) = \tau(S') * m(S')$, and further $m(S) = \tau(S)^{-1}\tau(S') * m(S')$. Also $S = \tau(S)^{-1}\tau(S') * S'$.

Invariants are very useful in classifications. An invariant can be used to speed up the computation of a canonical labelling as described in Section 4.4, in which case the function f can be based on the same invariant, and equation (R6) will hold because of the way the invariant was used in the computation of the canonical labelling. See Section 4.4 for more details.

We call a structure $S \in \mathscr{S}$ *reducible* if there exists an $S' \in \mathscr{S}$ such that $(S', S) \in \mathscr{R}$. Otherwise S is *irreducible*. Following the convention in [14] the irreducible objects are called *seeds*. The classification of seeds is a separate and on many occasions much easier task. Let $\mathscr{S}'_0$ be a maximal set of pairwise nonisomorphic seeds which can be obtained by classifying the seeds.

Still one more function is needed: the order of a structure. In this work order does not mean quite the same thing as in [30]. The order of S is denoted by $r(S)$ and defined as follows. The order of any irreducible structure is 0. The order of a reducible structure S is $r(S) = r(m(S))+1$. By (R1), r is well defined and finite for every S. By (R2),(R3), and (R4) isomorphic systems have equal order. We denote the set of structures with order n by $\mathscr{S}_n$. Especially $\mathscr{S}_0$ is the set of irreducible structures.

Now the elements of $\mathscr{S}$ can be classified by calling the procedure DESIGN-C1 once for every $S' \in \mathscr{S}'_0$.

DESIGN-C1(S')

```
 1  print S'
 2  for every S ∈ 𝒮 such that (S', S) ∈ ℛ
 3        do
 4           if f(S'') < f(S') for some S'' such that (S'', S) ∈ ℛ
 5              then reject S
 6           if S' is not in the Aut(S)-orbit of m(S)
 7              then reject S
 8           if S is not the smallest element in its Aut(S')-orbit,
 9              then reject S
10           if s has not been rejected
11              then DESIGN-C1(S)
```

We call the tests performed in lines 4, 6 and 8 of the procedure DESIGN-C1 the invariant test, parent test and automorphism test, respectively. Only structures that pass every test are printed.

Since $\mathscr{S}$ is finite, also $\mathscr{R}$ is finite. Thus the algorithm terminates after a finite number of operations.

In the classifications considered in this work testing whether S is the smallest element in its $\mathrm{Aut}(S')$ orbit is achieved by precomputing $\mathrm{Aut}(S')$ and testing for each $g \in \mathrm{Aut}(S')$ whether $g * S < S$. It might be possible to devise a more optimized test but this test works well enough in practice. Note that if S' has trivial automorphism group, S is always element in its $\mathrm{Aut}(S')$-orbit.

If S' has many automorphisms,the test in line 8 may be rather inefficient. For those cases the alternative algorithm DESIGN-C2 can be used.

Let $\phi : \mathscr{S}_0 \to \{\text{TRUE}, \text{FALSE}\}$ be any function. In practice $\phi(S')$ should be TRUE when S' has small enough automorphism group for calling DESIGN-C1(S') to be more efficient than calling DESIGN-C2(S'). Now the algorithm DESIGN-CLASSIFICATION classifies all designs of S.

DESIGN-CLASSIFICATION

```
1  for each S' ∈ 𝒮'_0
2  if φ(S')
3     then DESIGN-C1(S')
4     else DESIGN-C2(S')
```

Theorem 19. *If $\mathscr{S}'_0$ is as described before and $\phi : \mathscr{S}_0 \to \{\text{TRUE}, \text{FALSE}\}$ an arbitrary function, calling* DESIGN-CLASSIFICATION *outputs exactly one system in each G-orbit of $\mathscr{S}$.*

Proof. We proceed by induction on the order of the structures. For the struc-

```
DESIGN-C2(S')
 1  print S'
 2  C ← ∅
 3  for every S ∈ 𝒮 such that (S', S) ∈ ℛ
 4        do
 5           if f(S'') < f(S') for some S'' such that (S'', S) ∈ ℛ
 6              then reject S
 7           if S' is not in the Aut(S)-orbit of m(S')
 8              then reject S
 9           if τ(S) * S ∈ C
10              then reject S
11           if S has not been rejected
12              then
13                   C ← C ∪ {τ(S) * S}
14                   DESIGN-C2(S)
```

tures of order 0, i. e., seeds, the claim holds. Now assume that the claim holds for structures of orders at most n.

First note that testing whether $f(S'') < f(S')$ for some S'' is redundant, since if S is rejected in that test, by requirement (R5) and (R6) S' is not in the $\mathrm{Aut}(S)$-orbit of $m(S)$ which will lead to rejection of S later on.

First we show that the algorithm does not output isomorphic systems. To arrive at a contradiction, assume that two isomorphic systems S and S' of order $n+1$ are output. Now, by (R4) $m(S)$ and $m(S')$ are isomorphic systems of order n. By the induction hypothesis there is a unique $L \in \mathscr{S}$ that is isomorphic to both of them and accepted by the algorithm. Thus both S and S' are generated by a call DESIGN-C1(L) or DESIGN-C2(L).

Consider the case that both were generated by DESIGN-C1. Let g be as in (R4). Since both S and S' are accepted, there are $h \in \mathrm{Aut}(S)$, $h' \in \mathrm{Aut}(S')$ such that $L = h * m(S)$, $L = h' * m(S')$. Now hgh'^{-1} maps S' to S and L to itself. Thus both S and S' are in the same $\mathrm{Aut}(L)$-orbit and only the smaller of them is accepted, a contradiction.

Now let us consider the case that both S and S' were generated by DESIGN-C2. In this case we have $\tau(S) * S = \tau(S') * S'$. Thus, after S is output, C contains $\tau(S) * S$ and S' will be rejected. A contradiction again.

We still need to show that for an arbitrary $S \in \mathscr{S}$ of order $n+1$ some system isomorphic to S will be output. Let L be the unique accepted element of $\mathscr{S}_n$ isomorphic to $m(S)$, $L = g * m(S)$. Let $h \in \mathrm{Aut}(L)$ be such that $S' := hg * S$ is minimal. By (R2) and (R3), $(L, S') \in \mathscr{R}$.

Since S and S' are isomorphic, there is a k such that $m(S') = k * m(S)$ and

$S' = k * S$. Thus $hgk^{-1} \in \mathrm{Aut}(S')$ maps $m(S')$ to L, and L is in the $\mathrm{Aut}(S')$ orbit of $m(S')$.

By definition S' is smallest element in its $\mathrm{Aut}(L)$ orbit; thus DESIGN-C1(L) accepts S'. However DESIGN-C2(L) may reject S' if C contains $\tau(S') * S'$, but in that case some system isomorphic to S' has already been accepted. □

3.4 Efficiency Considerations

The more often equality does not hold in (R6), the more useful the invariant is. At the extreme, equality holds only if S' and $m(S)$ are in the same $\mathrm{Aut}(S)$-orbit, in which case the invariant uniquely determines the canonical substructure. However, such an invariant can rarely be efficiently computed, and the choice of a good invariant is a trade-off between usefulness and efficiency.

In the STS and SQS classifications the classification of the seeds was much easier task than the classification proper. In general the classification of the seeds is a separate task whose difficulty is not discussed here.

There are two natural requirements that the algorithm should satisfy to be efficient: it should not consider too many elements of $\mathscr{S}$ and the basic operations should be efficient.

Efficiency of computing canonical labellings is considered in Section 4.2. The time required for deciding whether S is the smallest element in its $\mathrm{Aut}(S')$-orbit depends on the order of the group $\mathrm{Aut}(S')$. Computing any reasonable invariant is clearly faster than computing canonical labelling. With the use of invariants more demanding tests are often avoided.

The more structures the algorithm considers, the more time it requires. Some of those structures are rejected and some are not. For all sound classification algorithms an equal number of structures pass all tests and are output. However the number of rejected structures may vary, and should be small.

For the number of structures that the algorithm studies to be small, the cardinality of $\mathscr{S}'_0$ should not be very large, its elements should not have automorphism groups of large orders, and on average $S \in \mathscr{S}$ should have few nonisomorphic structures S' for which $(S', S) \in \mathscr{R}$.

Since the main loop of the algorithm studies each element of $\mathscr{S}'_0$, the fewer elements there are in $\mathscr{S}'_0$, the better.

Assume that $S \in \mathscr{S}'_0$ and $g \in \mathrm{Aut}(S)$. Now if $S' \in \mathscr{S}$ is generated by extending S, so is $g * S$. Of course it is possible that $S = g * S$, but complete structures tend to have fewer automorphisms than seeds. Clearly $S \cong g * S$. Thus the automorphisms of S lead to generation of redundant structures and it is desirable that S has few automorphism. The same conclusion can be drawn by studying

line 8 of the algorithm DESIGN-C1. Also automorphisms of S do not only lead to redundant structures, they also slow down the execution of that line.

Since the classifications may be computationally very intensive, the algorithm should be parallelizable. Unlike some simpler and more intuitive algorithms, generation by canonical augmentation is easily parallelizable. Each seed can be considered independently whereas other algorithms may have a global variable similar to C in DESIGN-C2. Also there is no need to store the accepted structures, whose number may be too large for storing them all to be feasible.

When possible, the larger the order of the automorphism group of S is, the larger $f(S)$ should be. This way the invariant is specially efficient when S has many automorphisms and hence most likely many extensions. Equivalently it implies that structures are accepted when constructed by extending a substructure with minimal number of automorphisms.

3.5 Classification of Steiner Triple Systems

The framework given in Section 3.3 is so general that there are several possible classification algorithms for STS(v) based on it, some more efficient than others. Here the algorithm used by Kaski and Östergård [14] for $v = 19$ is described.

The set $\mathscr{S}$ consists of merely two parts, the STS(19) and the seeds, where a seed is a substructure of STS(19) consisting of a block and all other blocks intersecting it. The extension relation is defined so that $(S', S) \in \mathscr{R}$ if S' is a proper substructure of S. This choice leads to a classification algorithm consisting of two parts—extending the designs into STS(19) and rejecting isomorphic designs—instead of several interleaving stages of extension and isomorph rejection.

Alternatively the seeds could be the derived designs of STS(19). Each derived design consists of 18 points and 9 blocks such that each point occurs in exactly one block. It is not hard to see that this kind of structure has automorphism group order $9! \cdot 2^9 = 185{,}794{,}560$. Thus derived designs are not suitable seeds in this case.

The seeds actually used in the classification have less trivial structure, and it is not easy to say how many automorphisms they have. It turns out that due to their less trivial structure, they have less symmetry and are more suitable for being seeds. However it is quite easy to see that each structure consists of $3(9-1)+1 = 25$ blocks since the replicate number for STS(19) is 9. There are $14,648$ such nonisomorphic structures [14].

The function ϕ is quite simply TRUE for a seed S if the automorphism group of S has order at most 200 and false otherwise.

The canonical substructure function m is defined based on the canonical la-

belling by $m(S) = \tau_B^{-1}(S) *_B u(\tau_B(S) *_B S)$. Here u maps S to its substructure consisting of block b, all blocks intersecting b and all points, where b is the smallest block (according to the standard order of $\mathbb{N}$). Since $u(S)$ does not depend on the labelling of the points, it suffices to use canonical labelling of blocks instead of a canonical labelling.

Assume that $(S', S) \in \mathscr{R}$ for a seed S' and an STS S. Furthermore, let b' be the block of S' intersecting all other blocks and b the block that $\tau_B(S)$ maps to the smallest block of $\tau_B(S) * S$. Now S' is in the $\mathrm{Aut}(S)$-orbit of $m(S)$ if and only if the blocks b' and b are in the same $\mathrm{Aut}(S)$-orbit.

An invariant function based on counting the number of Pasch configurations was employed.

3.6 Classification of Steiner Quadruple Systems

Also the classification of SQS(v) can be accomplished with several different algorithms based on the framework of Section 3.3. Here the algorithm that we used for the classification of SQS(16) is described.

As in Section 3.5, the set $\mathscr{S}$ consists of two parts, which are in this case the SQS(16) and their derived designs, i. e., STS(15).

The canonical substructure function is $m(S) = \tau_P^{-1}(S) *_P u(\tau_P(S) *_P S)$, where $u(S)$ is the derived systems of S associated with smallest point. Since u does not depend on the labelling of blocks, point canonical labelling can be used.

In the SQS(16) classification the seeds are the 80 nonisomorphic STS(15), which have been classified almost a century ago [36]. Each STS(15) consists of 15 points and 35 blocks, and most have an automorphism group of small order [26]. Choosing seeds as in STS classification would lead to seeds consisting of 16 points and 101 blocks, and classifying the seeds would be quite a challenge.

Let us number the 80 isomorphism classes of STS(15) such that the larger the order of the automorphism group of an STS(15), the greater number its isomorphism class has. The invariant function f assigns to an STS(15) the number of its automorphism group. In this way the invariant is most effective for seeds with large automorphism group. Especially there is one STS(15) with automorphism group order almost 100 times greater than automorphism group order of any other STS(15) [26], and almost all extensions of that STS were rejected by the invariant test; only homogeneous extensions pass that test. Since on average the other STSs of order 15 have small automorphism group order, it was not worth the trouble to implement Design-C2, and the function ϕ was chosen to be identically True.

In practice the invariant was computed by counting for each point the number of Pasch configurations it occurs in.

As noted in Section 3.4, it is desirable that structures of $\mathscr{S}$ have few seeds from which they can be constructed. When the seeds are as in Section 3.5, the number of seeds from which a complete structure can be constructed is equal to the number of blocks, whereas if seeds are the derived designs, the number of seeds from which a structure can be constructed is equal to the number of points. Since all t-designs with $t \geq 2$ and $v > k$ have at least as many blocks as they have points, it seems that seeds chosen as in Section 3.6 are in general more suitable.

The classification algorithm was implemented as two separate programs: one that finds all extensions of the seeds and another one that performs the isomorph rejection. To further parallelize the algorithms, the task of extending a seed was divided into 315 subtasks, each of which consisted of traversing one branch of the search tree of the exact cover algorithm introduced in Section 4.1.

Dividing the classification software into two parts had some advantages as well as disadvantages. The first program generated huge amounts of data, copying and recording of which was troublesome. There would have been significantly less data to store if only the structures that passed isomorph rejection had been recorded.

However storing all the data may be very useful. For example if the isomorph rejection software has a bug and produces erroneous results, the software need to be fixed and rerun. If all extensions have been saved, only the isomorph rejection need to be rerun, and a lot of time is saved.

Sections 4.4 and 4.5 consider computing the canonical labelling in the SQS(16) classification.

Chapter 4

Auxiliary Algorithms

In this section the tasks of extending partial designs and computing canonical labellings are discussed in more detail.

4.1 Exact Cover Search

Some efficient algorithm is needed for the task of finding for given $S \in \mathscr{S}_n$ every $S' \in \mathscr{S}_{n+1}$ such that $(S, S') \in \mathscr{R}$. In fact this part of the classification of SQS(16) required significantly more time than the isomorph rejection. It seems that this part is more demanding also from computational complexity point of view, and there are more sophisticated algorithms for isomorph rejection.

When dealing with Steiner systems, the task of finding such completions can be formulated as an exact cover problem.

Problem 1 (Exact cover)**.** The problem instance consists of a set T and a set S that is a collection of subsets of T. A solution is a subset of S in which each $t \in T$ occurs exactly once.

Now consider the task of completing an $S \in \mathscr{S}_0$ to an $S(t, k, v)$ design. In other words, the task is to find a set of blocks that covers each t-subset left uncovered by the blocks of S. Thus every extension corresponds to an exact cover and vice versa.

To study the complexity of the exact cover problem we list several variants of it.

Problem 2 (Exact cover(D))**.** Given an instance of the exact cover problem, determine whether a solution exists or not.

Problem 3 (#Exact cover)**.** Given an instance of the exact cover problem, determine the number of solutions it has.

Problem 4 (EXACT COVERS)**.** Given an instance of the exact cover problem, output all of its solutions.

It is easy to see that Problems 2–4 are in ascending order by difficulty, that is, as a by-product of solving any of the problems we get the answer to the previous ones. Also solving Problem 1 gives solution of Problem 2. The most difficult one, Problem 4, is the one that needs to be solved in the classification of SQS(16).

It is well known that EXACT COVER(D) is NP-complete [32, p. 201]. Thus it is unlikely that an efficient algorithm exists for solving EXACT COVER. However #EXACT COVER is a #P-complete problem, thus most likely even harder than EXACT COVER(D).

Theorem 20. EXACT COVER(D) *is* NP*-complete and* #EXACT COVER *is* #P*-complete.*

Proof. Both claims are proved by a parsimonious reduction of 3SAT to EXACT COVER (see Appendix B for relevant definitions).

Consider an instance of 3SAT with variables $x_1, \ldots, x_n$ and clauses α_1, ..., α_m. Define an occurrences of a literal y as pairs (y, i) such that y occurs in the clause α_i.

Let T and S have the same meaning as in the definition of Problem 1. The set T consists of the integers $1, \ldots, n$, the clauses $\alpha_1, \ldots, \alpha_m$, and all occurrences of the literals.

The set S consists of subsets of T constructed as follows. For each integer $1 \leq i \leq n$ form a set F_i consisting of i and all occurrences of x_i, and a set T_i consisting of i and all occurrences of $\neg x_i$. For every clause $\alpha_k = (y_1 \wedge y_2 \wedge y_3)$ form all subsets of $\{\alpha_k, (y_1, k), (y_2, k), (y_3, k)\}$ that contain the clause and at least one occurrence of some literal. (The reduction works only for 3SAT since the number of subsets generated is exponential in the size of the clause).

If the 3SAT instance is satisfied when $x_i =$ TRUE for $i \in A$ and $x_i =$ FALSE for $i \notin A$, the corresponding EXACT COVER problem has a solution consisting of sets T_i such that $i \in A$, F_i such that $i \notin A$, and maximal sets covering all clauses and the uncovered occurrences of literals. Conversely every exact cover is of this form and corresponds to a unique truth assignment satisfying the original clauses.

It is not hard to see that space requirement of the transformation is logarithmic in the number of clauses. □

The more widely known proof for NP-completeness of EXACT COVER relies on a reduction producing EXACT COVER instances where each subset consists of exactly 3 elements. That proof shows that also the special case EXACT COVER

BY 3-SETS is NP-complete. However the reduction used in that proof is not parsimonious, and it is not known by the author whether EXACT COVER BY 3-SETS is #P-complete.

The results above state that #EXACT COVER is #P-complete, not that the special cases induced by Steiner systems are. However Colbourn has proved that also the special case induced by partial Steiner triple systems is NP-complete [5]. It is not known by the author whether similar results hold for other Steiner system or concerning the #P-completeness of extending Steiner (triple) systems.

The best known algorithm for solving the exact cover problem is a simple backtracking search given below. Efficiency of the algorithm can be increased by a clever choice of data structures suggested by Knuth [19].

```
EXACT-S(T, S, C)
1  Choose any t ∈ T
2  for each s ∈ S such that t ∈ s
3        do
4           C' ← C ∪ {s}
5           T' ← T \ s
6           S' ← {r ∈ S | r ∩ s = ∅}
7           if T' = ∅
8              then print C'
9              else  EXACT-S(T', S', C')
```

```
EXACT-SEARCH(T, S)
   EXACT-S(T, S, ∅)
```

The choice of t in line 1 should be made with some heuristic to further increase efficiency of the algorithm. Choosing t that occurs in least number of sets of S is simple but effective. More complicated methods usually use more time than they save.

4.2 The Isomorphism Problem

Virtually any kind of discrete structures can be represented as partitioned graphs (see Section 4.3). Especially deciding isomorphism of other structures can be reduced to deciding the isomorphism of graphs. Also plain graph would suffice, but are rarely as practical.

Since the problem of deciding graph isomorphism has been widely studied, from a theoretical as well as a practical point of view, it makes sense to use reduction to graph isomorphism in practice, as is done in this work. However, insight into the original structures may be useful for developing invariants that speed up

isomorphism computations, see Section 4.3. The *nauty* software package (see Section 4.4) was used for computing canonical labellings. It also computes generators and the order of the automorphism group.

Problem 5 (GRAPH ISOMORPHISM)**.** Given two graphs G and G', determine whether they are isomorphic or not.

Deciding isomorphism of partitioned graphs can be reduced to deciding isomorphism of graphs, and deciding isomorphism of graphs is a special case of deciding isomorphism of partitioned graphs. Thus there is no significant difference between complexity deciding isomorphism of graphs and complexity of deciding isomorphism of partitioned graphs.

Computing automorphisms and isomorphisms are closely related tasks for graphs as well as partitioned graphs;

Clearly GRAPH ISOMORPHISM is in NP. It is not known to be in P, although much effort has been used for finding efficient algorithms. In a theoretical sense (asymptotic worst case time) probably the best known algorithm for GRAPH ISOMORPHISM is computing canonical labellings by combining techniques due to Zemlyachenko and Luks [1]; this way isomorphism can be decided in modestly exponential time $\exp(\sqrt{v}+o(1))$, where v is the number of vertices. For several special cases there exist more efficient algorithms. For example, isomorphism of graphs with bounded vertex degree can be decided in polynomial time [1].

It is not known whether GRAPH ISOMORPHISM is NP-complete, but there are results suggesting it is not. The counting problem #GRAPH ISOMORPHISM is polynomially reducible to the corresponding decision problem [24], but for known NP-complete problems the corresponding counting problem seems to be harder. Also if GRAPH ISOMORPHISM is NP-complete, the polynomial hierarchy collapses to the second level [4], which is considered unlikely.

Accordingly it seems that GRAPH ISOMORPHISM is neither in P nor NP-complete. Indeed, unless P = NP, NP contains problems that are neither in P nor NP-complete [32, Theorem 14.1].

In this work the task of computing canonical labellings of Steiner systems is reduced to computing canonical labellings of partitioned graphs. There are some requirements for the graph representation. It should be isomorphism preserving, that is, two graphs should be isomorphic if and only if the corresponding Steiner systems are isomorphic. Canonical labelling of a Steiner system should be possible to obtain from the canonical labelling of the corresponding graph. It is also desirable that the representation be automorphism preserving, that is, a Steiner system and the corresponding graph should have isomorphic automorphism groups. Also there are more practical requirements: it should be possible to transform Steiner systems to graphs and processing the graphs efficiently.

4.3 Partitioned Graphs

Assume that there is a set $\tilde{V}$ such that the vertex set of each graph is a subset of $\tilde{V}$. The set $\tilde{V}$ should be well-ordered, that is, there should be a total order $\prec$ on $\tilde{V}$ such that each nonempty subset of $\tilde{V}$ has a smallest element. In the classification software we let $\tilde{V} = \mathbb{N}$.

Only finite graphs are needed in the classification, and in the following definitions it is assumed that the graphs are finite. To some extent the definitions and results could be generalized for infinite graphs. Especially Definition 26 is directly applicable to infinite graphs.

Definition 24. A *partitioned* or *ordered* graph is a triple (V, E, Γ) such that (V, E) is a graph and $\Gamma = (V_1, V_2, \ldots, V_n)$ is an ordered partition of the vertex set V. The subsets V_i are called *cells* of the partition.

An unlabelled graph (V, E) can be considered as a partitioned graph with the partition $\Gamma = (V)$. At the other extreme we have graphs with *discrete* partition, i. e., partition that consists of cells of cardinality one.

Sometimes partitioned graphs are called colored graphs in which case the cells are color classes. However, the most common meaning of coloring of a graph involves additional restrictions totally unrelated to the isomorphism properties considered here.

Definition 25. Assume that Γ is a partition of a finite set $V \subseteq \tilde{V}$. The *canonical partition* $c(\Gamma)$ is the partition $c(\Gamma) = \gamma * \Gamma = (W_1, \ldots, W_n)$, where $\gamma \in S_{\mathbb{N}}$ is chosen so that $\gamma * V$ consists of the $|V|$ smallest elements of $\tilde{V}$, and in addition $v_1 \in W_i$, $v_2 \in W_j$ and $i < j$ imply that $v_1 < v_2$.

It is not hard to see that the canonical partition exists and is unique for any V, but the permutation γ need not be unique. Note that the partitions Γ and $g * \Gamma$, $g \in S_{\tilde{V}}$, have the same canonical partition.

Definition 26. Let $G = (V, E, \Gamma)$ and $G' = (V', E', \Gamma')$ be two partitioned graphs, $\Gamma = (V_1, \ldots, V_n)$, $\Gamma' = (V'_1, \ldots, V'_n)$. An *isomorphism* from G to G' is a bijection $f : V \to V'$ that maps E onto E', and furthermore satisfies $f(V_i) = V'_i$ for $1 \leq i \leq n$. If an isomorphism from G to G' exists, we say that G and G' are *isomorphic*.

Since we assume that $V, V' \subseteq \tilde{V}$, the graphs G and G' are isomorphic if and only if $g * G = G'$ for some $g \in S_{\tilde{V}}$.

Broadly speaking an *invariant* is some property of a graph that does not depend on the labelling of the vertices. It seems there is no consensus on exact definitions, e. g., [14] and [28] define vertex invariant differently. The latter definition is used in this work. The usefulness of invariants is briefly discussed in Section 4.4.

Definition 27. Let $\mathscr{G}$ be a set of partitioned graphs closed under the action of $S_{\tilde{V}}$. A *vertex invariant* is a mapping $\phi : \mathscr{G} \times \tilde{V} \to \mathbb{Z}$ such that if $\gamma \in S_{\tilde{V}}$, then $\phi(G, v) = \phi(\gamma * G, \gamma * v)$.

A vertex invariant assigns a value to a vertex of a graph. Similarly a *graph invariant* assigns a value to a graph, but this time the set of possible values is different.

Definition 28. Let $\mathscr{G}$ be as in Definition 27. A *graph invariant* Λ assigns to each graph $G = (V, E, \Gamma) \in \mathscr{G}$ an ordered partition $\Lambda(G)$ of the set V such that $\Lambda(\gamma * G) = \gamma * \Lambda(G)$ for every $\gamma \in S_{\tilde{V}}$. It is also required that $\Lambda(G)$ is *finer* than Γ, i. e., every cell of $\Lambda(G)$ is a subset of some cell of Γ.

The mapping $\Lambda(G) = \Gamma$ is a trivial invariant as is any permutation of the cells.

Definition 29. A graph invariant Λ is *order preserving* if, for any graph $G = (V, E, \Gamma) \in \mathscr{G}$, $\Gamma = (V_1, \ldots, V_n)$, the partition $\Lambda(G) = (W_1, \ldots, W_m)$ satisfies the following condition: $W_k \subseteq V_i$, $W_l \subseteq V_j$ and $i < j$ imply $k < l$.

Informally an order preserving invariant splits the cells into smaller ones, but does not change their order.

Every vertex invariant ϕ defines a derived graph invariant Λ_ϕ such that for $G = (V, E, \Gamma) \in \mathscr{G}$ and $\Gamma = (V_1, \ldots, V_n)$, $\Lambda_\phi(G)$ has cells of form $W_{i,j} = \{v \in V_i \mid \phi(G, v) = j\}$. The partition $\Lambda_\phi(G)$ consists of all non-empty sets of this form ordered lexicographically by the pairs (i, j). Since ϕ is a vertex invariant, Λ_ϕ is a graph invariant. It is also easy to see that Λ_ϕ is order-preserving.

One can also derive a vertex invariant from a graph invariant. However, in practice graph invariants are often best described by giving the corresponding vertex invariant, and are computed by computing the vertex invariant, see for example the list of invariants in [28]. When dealing with graph representations of combinatorial designs, one may develop invariants for the graphs based on invariants of the designs. For example in the SQS classification a vertex invariant based on the isomorphism classes of derived designs was employed.

4.4 nauty

nauty is a graph canonical labelling software by Brendan McKay [27, 28].

Definition 30. A canonical labelling of partitioned graphs is a function that maps a graph $G = (V, E, \Gamma)$ to $\sigma_G \in S_{\tilde{V}}$ such that

$$\sigma_G * \Gamma = c(\Gamma), \tag{4.4.1}$$

and for each $\mu \in S_{\tilde{V}}$

$$\sigma_G * G = \sigma_{\mu * G} * (\mu * G). \tag{4.4.2}$$

Equation (4.4.1) is somewhat unnecessary since the equation (4.4.2) guarantees that the canonical labelling solves the isomorphism problem and satisfies Definition 22. However, equation (4.4.1) is quite useful in practice.

In addition to computing the canonical labelling of a graph, *nauty* also computes generators of its automorphism group and the order of the automorphism group.

The following theorem shows how to obtain a canonical labelling with the use of an invariant, and Theorem 22 shows that the use of an invariant does not interfere with *nauty*'s automorphism computation. Theorem 22 and the part of Theorem 21 stating that the new mapping satisfies (4.4.2) are proved in [14].

Theorem 21. *Assume that Λ is an order preserving vertex invariant, Δ is the mapping $(V, E, \Gamma) = G \mapsto \Delta(G) = (V, E, \Lambda(G))$ and that $G \mapsto \sigma_G$ is a canonical labelling of graphs. Then the mapping $G \mapsto \sigma_{\Delta(G)}$ is also a canonical labelling. If Λ is not order preserving, (4.4.2) will still hold for $\sigma_{\Delta(G)}$.*

Proof. Assume that $G = (V, E, \Gamma)$, $\Gamma = (V_1, \ldots, V_n)$, $\Lambda(G) = (W_1, \ldots, W_m)$. Since G is fixed, we may denote $\sigma_{\Delta(G)}$ briefly by σ.

It is easy to see that $\sigma * V$ consists of the $|V|$ smallest elements of $\tilde{V}$. To prove that $\sigma * \Gamma = c(\Gamma)$, we still need to show that for arbitrary $v_1 \in \sigma * V_i$, $v_2 \in \sigma * V_j$, $i < j$, we have $v_1 \prec v_2$.

Define $v'_m = \sigma^{-1} * v_m$ for $m = 1, 2$. Now $v'_1 \in V_i$ and $v'_2 \in V_j$. Since $\Lambda(G)$ is a finer partition than Γ, it contains cells W_l and W_k such that $v'_1 \in W_l \subseteq V_i$, $v'_2 \in W_k \subseteq V_j$. Since Λ is order preserving, $l < k$. Now $\sigma * \Lambda(G) = c(\Lambda(G))$ by the definition of σ, which implies $\sigma * v'_1 \prec \sigma * v'_2$ by the definition of canonical partition.

To finish the proof, note that $\sigma_{\Delta(\mu * G)}(\mu * G) = \sigma_{\mu * \Delta(G)}(\mu * G) = \sigma_{\Delta(G)} * G$ where the first equation follows the fact that Λ is a graph invariant and second from the fact that the mapping $G \mapsto \sigma_G$ is a canonical labelling. □

Theorem 22. *Assume that Λ is a graph invariant and Δ is the mapping $(V, E, \Gamma) = G \mapsto \Delta(G) = (V, E, \Lambda(G))$. For an arbitrary graph G,* $\mathrm{Aut}(G) = \mathrm{Aut}(\Delta(G))$.

Proof. If $\gamma \in \mathrm{Aut}(\Delta(G))$, γ is an automorphism of G since $\Lambda(G)$ is finer than Γ.

If $\gamma \in \mathrm{Aut}(G)$, $\gamma * \Lambda(G) = \Lambda(\gamma * G) = \Lambda(G)$ by the definition of an invariant. Thus $\gamma \in \mathrm{Aut}(\Delta(G))$. □

The algorithm used by *nauty* works by searching for a suitable discrete partition of the vertex set by considering different ways of refining the original partition. The finer the original partition is, the less refinement is needed and the easier

the task is. However, the difficulty of the computation depends also on several other factors.

A canonical labelling of graphs can be used in the implementation of the generation by canonical augmentation as follows. Let $\mathscr{S}$ be the set of relevant combinatorial structures and $\mathscr{G}$ the set of their graph representations. Assume that a group H acts on both $\mathscr{S}$ and $\mathscr{G}$ such that the orbits of the action are the isomorphism classes.

Let $g : \mathscr{S} \to \mathscr{G}$ be the function that maps each structure to its graph representation. The function g should fulfill the following requirements. The structures S_1 and S_2 are isomorphic if and only if $g(S_1)$ and $g(S_2)$ are. Furthermore if $g(S) = G = (V, E, \Gamma)$, where $\Gamma = (V_1, \ldots, V_n)$, each vertex $v \in V_1$ corresponds to a substructure S' of S such that $(S', S) \in \mathscr{R}$, and this correspondence is bijective. Let h be a function that carries this correspondence, that is, $h(v, G) = S'$ when v, G and S' are as above. Both g and h should be invariant in the sense that $h(\gamma * v, \gamma * G) = \gamma * h(v, G)$ and $g(\gamma * S) = \gamma * g(S)$ and $\gamma \in H$.

Let f be the invariant function needed for the generation by canonical augmentation. Now we can define a vertex invariant ϕ by $\phi(v, G) = f(h(v, G))$ when v is in the first cell of the partition of G, and $\phi(v, G) = 0$ otherwise. It is possible to define ϕ less trivially for vertices not included in the first cell, but there is no general way to it. Let Λ_ϕ be the graph invariant based on the vertex invariant ϕ, and Δ be the function $G = (V, E, \Gamma) \mapsto \Delta(G) = (V, E, \Lambda_\phi(G))$. Now the canonical substructure function can be defined as $m(S) = h(v', g(S))$ where the vertex v' is chosen so that $\sigma_{\Delta(g(S))} * v'$ is minimal.

Assume that $m(S) = G = (V, E, \Gamma)$, $\Gamma = (V_1, \ldots, V_n)$, and furthermore that $\Lambda_\phi(m(S)) = (W_1, \ldots, W_n)$. By the definition of Λ_ϕ, W_1 consists of vertices $v \in V_1$ such that $\phi(v, G)$ is minimal. Since $\sigma_{\Delta(G)} * \Lambda_\phi(G) = c(\Lambda_\phi(G))$, we have $v' \in W_1 \subseteq V_1$, and consequently $h(v', G)$ is defined. In addition by the definition of ϕ we have $f(m(S)) \leq S'$ for all S' such that $(S', S) \in \mathscr{R}$.

If $\sigma_G \in H$ for every G, and g is injective, then canonical labelling of the incidence structures can be defined as $\tau(S) = \sigma_{g(S)}$. To show that τ indeed is a canonical labelling, assume that $S' = \gamma * S$. Now $g(\tau(S') * S') = g(\sigma_{g(\gamma * S)} * \gamma * S) = \sigma_{g(\gamma * S)} * \gamma * g(S) = \sigma_{\gamma * g(S)} * \gamma * g(S) = \sigma_{g(S)} * g(S) = g(\sigma_{g(S)} * S) = g(\tau(S) * S)$, which implies $\tau(S') * S' = \tau(S) * S$ since g is injective. In this case the definition of m given above is of the form $m(S) = \tau(S)^{-1} * u(\tau(S) * S)$ discussed in Section 3.3. This is the case in the SQS classification.

4.5 Point-Block Graph

Perhaps the most straightforward graph representation for an incidence structure (P, B, I) is the graph with vertex set $V = P \cup B$, edge set $E = I$ and partition $\Gamma = (P, B)$. We will call this graph the *point-block graph* of the inci-

dence structure. It is rather easy to see that this representation is isomorphism preserving as well as automorphism preserving, and a canonical labelling of the incidence structure can be obtained in a straightforward fashion from a canonical labelling of its point-block graph.

Point-block graphs are probably the simplest and oldest graph representation of incidence structures, but unfortunately they present difficulties for many graph isomorphism programs including *nauty* [27]. Accordingly also other graph representations are considered and used in other classifications.

This graph representation and the construction at the end of Section 4.4 was used in the classification of the SQS(16) for computing canonical labellings. The function h maps a vertex to the derived designs associated with the point corresponding to the vertex, and the function f is, as described in Section 3.6, a function that assigns to a point the number of the isomorphism class of the derived design associated with the point. In practice the invariant can be computed from the Steiner system, not from its graph representation. This construction, especially the canonical substructure function obtained in this way is equal to the one described in Section 3.6.

4.6 Pair Graph

Pair graphs were not employed in any classification, but they are described here to demonstrate some techniques used with graph representation for proving the isomorphism and automorphism preservation of such representations.

Let $(P, \mathcal{B})$ be an SQS(v). Its *pair graph* has as vertices the 2-subsets of $\mathcal{P}$. Two vertices are adjacent if there is a block containing both vertices as subsets.

Lemma 23. *If K is a clique in the pair graph such that no point occurs in all of its vertices, then K consists of at most* $\max(v/2, 6)$ *vertices.*

Proof. To prove the lemma a somewhat tedious study of several different cases is needed. First assume that every two pairs in K intersect. In this case K has to be $\{\{p, q\}, \{p, r\}, \{q, r\}\}$ for some p, q, r, and $|K| = 3$.

Now assume that there are two pairs that do not intersect. Let those pairs be $\{p, q\}$ and $\{r, s\}$. Since those pairs occur in same block, the SQS must contain the block $B = \{p, q, r, s\}$.

It is impossible for K to contain a pair with one element included in B and another one not, for if there were such a pair, say $\{p, t\}$, there had to be block $\{p, t, r, s\}$, a contradiction.

Assume that K contains a third pair included in B, say $\{p, r\}$, and a pair that is not included in B, say $\{t, v\}$. Hence the SQS contains blocks $\{p, q, t, v\}$ and $\{p, r, t, v\}$, a contradiction. Thus either K contains only two subsets of B or K

consists entirely of subsets of B. In latter case K consist of at most $\binom{4}{2} = 6$ vertices.

In the former case there is a pair $\{v, w\}$ not included in B. Thus no point of $\mathcal{P} \setminus B$ may occur in more than one pair of K, for if there were pairs $\{v, w\}$ and $\{v, y\}$, there had to be blocks $\{v, w, r, s\}$ and $\{v, y, r, s\}$, a contradiction. Since no two pairs of K have a point in common, there are at most $v/2$ pairs. □

Theorem 24. *An* SQS(v) *can be constructed (up to isomorphism) from its pair graph.*

Proof. If the pair graph consists of only six vertices, then $v = 4$ and the SQS consists of only one block. Otherwise $v \geq 8$ and the following algorithm reconstructs the SQS:

Assume that $p \in \mathcal{P}$. The set $K_p := \{\{p, q\} \mid q \in \mathcal{P}, q \neq p\}$ is a clique of the pair graph since for all q_1, q_2 there is a block which contains $\{p, q_1, q_2\}$. Since there are $v - 1$ possible values for q, the clique consists of $v - 1$ vertices.

According to Lemma 23 all cliques of size $v - 1$ are of form K_p for some p. Assign a label to each such clique. Each vertex occurs in exactly two such cliques. Assign the label $\{p, r\}$ to a vertex occurring in cliques p and r. If two vertices $\{p, r\}$ and $\{t, s\}$ are neighbors, add $\{p, r, t, s\}$ to the block set unless it is already included in it. □

Note that we do not use the algorithm presented in Theorem 24 as a computational method. We consider it as a theoretical result needed to establish that nonisomorphic SQSs cannot have isomorphic pair graphs. Yet we still need a stronger reconstructability result.

Theorem 25. *For $v \geq 8$ the automorphism groups of an* SQS(v) *and its pair graph are isomorphic.*

Proof. Let τ be an automorphism of an $SQS(v)$. If its action of the pair graph is defined on natural way, $\tau * \{p, v\} := \{\tau * p, \tau * v\}$, it is also an automorphism of the pair graph.

Let μ be an automorphism of the pair graph. Since μ maps cliques to cliques, for each $p \in P$ there is $q \in P$ such that $\mu * K_p = K_q$. If we define $\mu * p := q$, then μ is an automorphism of the SQS. □

Experiments established that computing canonical labellings of point-block graphs of SQS(16) with *nauty* was slow when no invariants were used, whereas computing canonical labellings of pair graphs was rather quick. However, since the use of invariants significantly reduced the running time of computing canonical labellings of point-blocks graphs, there was no need to use the less practical pair graph representation.

Obtaining a canonical labelling of an SQS from a canonical labelling of the pair graph is possible but somewhat tedious. However, canonical labellings of pair graphs could be used for isomorph rejection even without using it for computing canonical labellings of SQSs. The definition of the canonical substructure function m could be based on the canonical labelling of the pair graph instead of the canonical labelling of the incidence structure. Also the algorithm DESIGN-C2 could store the canonical labellings of the pair graphs instead of the canonical labellings of the incidence structures.

4.7 Block Graph

The *block graph* G_i of a Steiner system $S(t,k,v)$ $(P,\mathcal{B})$ is the graph with vertex set $\mathcal{B}$ and edge set $E = \{(B_1,B_2) \mid |B_1 \cap B_2| = i\}$. Kaski and Östergård [14] used block graphs with $i = 1$ for representing STS(19). Results similar to Lemma 23 and Theorems 24 and 25 can be found in [14] and [33]. Since vertices of G_1 correspond to blocks of the STS, the canonical labelling of a block graph gives the block canonical labelling of the corresponding STS, which was employed in the classification of the STS(19).

Chapter 5

Results

5.1 Miscellaneous results

Running the SQS(16) classification required approximately twelve years of CPU time, of which less than a day was used for the isomorph rejection and the rest for the exact cover search. The search was distributed using the batch system *autoson* [29]. A total of 107 computers, almost all with 2.2 or 2.4 GHz Intel CPUs, were at some point running the search.

There are 1,054,163 nonisomorphic SQS(16). The total number of labelled SQS(16) is 14,311,959,985,625,702,400; this number was computed with formulas given in Section 5.3.

De Vries has established [35] that 69 of the 80 STS(15) are derived systems of a homogeneous SQS. The classification results imply that none of the remaining 11 STS(15) is.

For each BSQS(16) it was determined whether its lower chromatic number is 2 or 3. This was achieved by simple exhaustive search among the 2^{16} possible 2-colorings. Total $349,058$ nonisomorphic SQS(16) have lower chromatic number $\chi = 2$.

The SQS(16) with rank at most 14 have been classified by Zinoviev and Zinoviev [39, 41, 43]. Their results agree with results obtained in this work and given in Table 5.4.

In Table 5.2 the standard numbering of STS given in [25] is used instead of the numbering used in classification.

A slight majority of the nonisomorphic systems have nontrivial automorphisms, but relatively few have more than 4 automorphisms.

Table 5.1: Nonisomorphic SQS(16) having given β (see Definition 11)

β	#	β	#
1	1,641	9	128,416
2	12,338	10	101,257
3	34,934	11	72,842
4	72,907	12	42,672
5	106,084	13	18,807
6	143,248	14	5,667
7	161,399	15	1,115
8	150,717	16	119

Table 5.2: Number of SQS having given STS(15) as derived system

No	#	No	#	No	#	No	#
1	13,711	21	47,125	41	5,780	61	14,179
2	240,118	22	49,243	42	105	62	2,606
3	213,133	23	157,868	43	275	63	5,503
4	759,223	24	134,657	44	671	64	3,478
5	410,563	25	166,233	45	1,068	65	183
6	257,899	26	196,444	46	363	66	187
7	43,092	27	75,791	47	4,738	67	108
8	699,707	28	73,897	48	542	68	161
9	725,288	29	63,255	49	344	69	241
10	742,266	30	22,692	50	343	70	4,800
11	294,132	31	57,948	51	597	71	168
12	324,812	32	37,117	52	1,020	72	131
13	389,642	33	26,625	53	6,059	73	40
14	301,162	34	29,240	54	5,130	74	310
15	431,065	35	2,959	55	1,163	75	452
16	77,610	36	1,817	56	482	76	3,307
17	143,673	37	35	57	360	77	34
18	427,530	38	742	58	5,786	78	52
19	57,425	39	6,252	59	5,513	79	5
20	186,917	40	6,562	60	523	80	5

Table 5.3: Nonisomorphic SQS(16) having given automorphism group order

$\lvert Aut(\mathcal{X})\rvert$	#	$\lvert Aut(\mathcal{X})\rvert$	#	$\lvert Aut(\mathcal{X})\rvert$	#
1	459,466	32	2,732	336	5
2	344,972	36	1	384	24
3	1,721	42	7	512	8
4	174,544	48	159	576	1
5	2	60	1	768	9
6	861	64	585	1,152	2
8	53,197	80	1	1,344	1
9	4	96	84	1,536	5
12	759	128	178	2,688	1
16	14,522	168	4	3,072	2
21	12	192	41	21,504	1
24	216	256	34	322,560	1

Table 5.4: Ranks of the SQS(16)

Rank	#
11	1
12	15
13	4,131
14	708,103
15	341,913

5.2 Resolvability

One of the interesting properties that can be studied when complete classification is available is the resolvability of the designs. Typically the interesting problem is whether resolvable designs of a certain kind exists. With SQS(16) the situation is different: it is known that there exists resolvable SQS(16), but the existence of nonresolvable SQS(16) has been an open question [39].

After classifying the SQS(16) we wish to determine the resolvability of each of those. There are two well known algorithms for this task, both of which can be formulated as exact cover problems.

One way of searching for resolutions is to choose a point $p' \in P$ and find all blocks incident with p'. Label these blocks as B_i. Each parallel class contains a unique B_i. Assign the label i to the parallel class containing B_i. Now the task of completing the parallel classes can be formulated as an exact cover problem: cover the pairs (p, i), $p \in P$, where p is not incident with B_i, and the blocks B not incident with p', with sets of the form $S_{i,B} = B \cup \{(p, i) \mid p \text{ is incident with } B\}$.

An alternative algorithm consists of two exact cover searches. In the first one all ways to cover the point set with blocks are determined. In other words all parallel classes are generated. When all parallel classes are known, one tries to cover the block set with parallel classes, that is, form resolutions.

Both algorithms were implemented. Since the latter one was significantly faster, resolvability of every SQS(16) was determined by it. It was established that 617,865 of the 1,054,163 SQS(16) are resolvable and the rest are not.

5.3 Reliability of the Results

Although the computation was not a proof of a theorem at least in the strictest sense, some remarks about computer-based proofs apply to computations of this sort as well.

Traditionally mathematical proofs have been written and verified by mathematicians, but during the 20th century also computers were used for both tasks. Computers are naturally very useful when the necessary computations or case by case analysis would require huge amounts of time by humans.

In such cases not only writing a proof but also verifying one is practically impossible without a computer. This leads to the question about the reliability of computer generated proofs, which has been debated and remains controversial. A more detailed discussion of the question is omitted. Instead some argument for the correctness of the SQS(16) classification are presented.

The classification software consists of *nauty*, exact cover subroutines provided

by Petteri Kaski [15] and some code specific to the SQS classification. Both *nauty* and the exact cover subroutine have been successfully used in several classifications and it is unlikely that either of them should contain undiscovered bugs.

The classification software correctly classified SQS(14) obtaining $N(14) = 4$ in accordance with previous results.

To provide further evidence on the correctness of the result a double-counting consistency test was employed. This check is analogous to test employed in [14]. Let us now consider only SQS(16) with point set $\{1, 2, \ldots, 16\}$ represented as set systems. Assume that, as a result of computation, we have obtained a set $\mathscr{S}'$ containing one representative from each isomorphism class. According to orbit-stabilizer (see Appendix A) theorem the total number of SQS(16) is

$$\sum_{S \in \mathscr{S}'} \frac{16!}{\mathrm{Aut}(S)}. \tag{5.3.1}$$

Assume that S_i, $i = 1, 2, \ldots, 80$ are the 80 nonisomorphic STS(15) and M_i is the number of ways S_i can be extended to an SQS(16). Again the orbit-stabilizer theorem gives us a formula for the total number of SQS(16):

$$\frac{1}{16} \sum_{i=1}^{80} M_i \frac{16!}{\mathrm{Aut}(S_i)}. \tag{5.3.2}$$

The division by 16 is necessary since each SQS(16) has 16 derived STS(15) and can be generated by extending any of them.

To actually compute the sum (5.3.2), one needs only the number of exact covers found during the classification whereas performing isomorph rejection is necessary to obtain the values occurring in the sum (5.3.1). If there is an error in the classification software, it is very unlikely that the classification would be erroneous consistently enough for sums (5.3.1) and (5.3.2) to agree.

Chapter 6

Conclusions

In this thesis the Steiner quadruple systems of order 16 were successfully classified. This task was completed by applying the McKay framework [30] in a similar way than it has been applied for classification of Steiner triple systems [14]. The exact cover algorithm by Knuth [19] and the graph canonical labelling software nauty by McKay [28] were used in implementation of the classification.

The fact that we have obtained yet another classification result with the tools mentioned above provides additional proof for the power of the tools. However the task was computationally very hard, and finding more efficient methods is an interesting problem.

Since the exact cover search required most of the CPU time, optimizing the search is naturally most obvious way to improve efficiency of the classification method. In the classification described in this thesis the search tree had a huge number of branches that had no leaves, and some way to prune these branches could provide great improvement in efficiency.

The classification method by Zinoviev and Zinoviev provides an interesting alternative. So far only systems with rank at most 14 have been classified with it, but classification of systems with rank 15 is expected to be ready in near future [43]. It is not at all apparent how the classification method for systems with given rank can be applied in this case.

Although the running time required by the the algorithm by Zinoviev and Zinoviev was not mentioned their papers, one would expect it to be significantly less than with our method. However, the method applied in this thesis has some advantages. The method is more general, although it is possible that the method by Zinoviev and Zinoviev can be generalized. Also our method includes some sort of verification of the results, which has proved itself indispensable in practice.

When considering the computational difficult of classification, also some theoretical results are of interest. The exact complexity of several relevant problem

such as completing partial Steiner systems and computing canonical labellings is not known. Furthermore it is not known what NP-completeness actually means, or can NP-complete problems only be solved in exponential time.

Now that Steiner quadruple systems of order 16 have been classified, 20 is the smallest admissible order for which a classification has not yet been obtained. Due to the exponential growth in the number of nonisomorphic systems, a complete classification for order 20 is not a feasible goal with today's technology. However, there are some partial classifications for similar structures which might be feasible for SQS(20), e. g., classification of systems with limited rank [39, 40] or classification of systems with prescribed group of automorphisms [16].

The exhaustive list obtained in the classification may be quite useful in future research. In this thesis it was established that non-resolvable SQS(16) exists. Similar results could be obtained regarding other properties. It might also be possible to use the SQS(16) as seeds in classification of $S(4, 5, 17)$ or some codes.

Appendix A

Groups and Group Actions

In this work a group theoretical framework is used for isomorphism consideration; a brief introduction to group theory is included here. Special emphasis is put on group actions. For a more comprehensive introduction to group theory the reader is referred to any textbook of abstract algebra, e. g., [3]. Since rings and fields play only a minor role in this work, we skip their definition and again refer the reader to any textbook on abstract algebra.

Definition 31. A *group* is a set G with a binary operation $\cdot : G \times G \to G$ fulfilling the following axioms:

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \text{ holds for all } x, y, z \in G. \tag{A.0.1}$$

$$\text{There exists an } e \in G \text{ such that } g = e \cdot g \text{ for all } g \in G. \tag{A.0.2}$$

$$\text{Every } g \in G \text{ has an inverse } g^{-1} \in G \text{ such that } g^{-1} \cdot g = e. \tag{A.0.3}$$

In equations (A.0.2) and (A.0.3) e refers to the same element, called *identity*. It is easy to see that the the inverse of any element and the identity are unique. It follows from the axioms that $g = g \cdot e$ and $g \cdot g^{-1} = e$ for any $g \in G$.

The number of elements in G is called the *order* of the group. Note that (A.0.2) implies that the set G is not empty and hence the order of a group is at least 1. The order may be infinite.

Any binary operation satisfying (A.0.1) is called an associative operation.

Definition 32. If G is a group a with binary operation $\cdot$, and $H \subseteq G$ is a group with the same binary operation, then H is called *subgroup* of G. H is a *proper* subgroup if $\{e\} \subsetneq H \subsetneq G$.

Definition 33. A group satisfying

$$x \cdot y = y \cdot x \text{ for all } x, y \in G \tag{A.0.4}$$

is called *commutative* or an *abelian* group.

For two groups G_1 and G_2, their Cartesian product $G_1 \times G_2$ is a group with operation $(g_1, g_2) \cdot (g_1', g_2') = (g_1 \cdot g_1', g_2 \cdot g_2')$.

Sometimes $\cdot$ is omitted from the notations, that is, $x \cdot y$ is denoted by xy.

Definition 34. Let X be a nonempty set. The set of all permutations of X, i. e., bijections from X onto X, is a group with composition of mappings as the group operation. The group is called *symmetric group* on X and denoted S_X. Subgroups of S_X are called permutation groups on X.

Sometimes it is required that the set X be finite, but no such requirements are made here. It is common to denote $S_{\{1,2,\ldots,n\}}$ by S_n for brevity.

The concept of group actions is very useful in the study of isomorphisms of combinatorial structures.

Definition 35. The *action* of a group G on a set X is a binary operation $* : G \times X \to X$ having the following properties:

$$\text{If } e \in G \text{ is the identity, then } e * s = s \text{ for every } s \in X, \tag{A.0.5}$$

$$\text{For all } x, y \in G, s \in X, \text{ we have } x * (y * s) = (x \cdot y) * s. \tag{A.0.6}$$

We say that G acts of X and X is a G-set.

If G is a permutation group on X, then the *natural* action of G on X is defined as $g * s = g(s)$. The natural action is employed in this study.

Definition 36. The *orbit* of $s \in X$ under the action of G is the set $G * s := \{g * s \mid g \in G\}$, and the *stabilizer* of s is $N_G(s) := \{g \in G \mid g * s = s\}$.

Note that the stabilizer of any element is a subgroup of G.

Theorem 26. *Let X be a G-set. Define a relation $\cong$ on X such that $x \cong y$ if $x = g * y$ for some $g \in G$. The relation is an equivalence relation. The equivalence classes of X are the orbits of the elements of X, which form a partition of X.*

If G acts on X, then the action can be extended to the power set of X: if $T \subseteq X$, $g * T := \{g * t \mid t \in T\}$. Also if G acts on the sets X and Y, the actions can be extended to $X \times Y$ as $g * (x, y) := (g * x, g * y)$. This can be generalized to the cartesian product of any number of G-sets. These extensions are used without explicit mentioning.

Properties unaffected by group action are called invariants. Pasch configurations (see Section 2.4) provide useful invariants for Steiner triple systems. Invariants for graphs are studied in Section 4.3.

The action of G on itself provides some nice examples. Action by *translation* is defined as $g * x = g \cdot x$ and action by *conjugation* as $g * x = g \cdot x \cdot g^{-1}$.

Action by translation is *transitive*, i. e., it has only one orbit, G. Action by conjugation is transitive if and only if G consists of one element only, since $G * e = \{e\}$. Subgroups that are invariant under conjugation are called *normal subgroups*. Normal subgroups are a important concept in group theory, as are *simple* groups, i. e., groups that do not have proper normal subgroups. Also $g * x = x \cdot g^{-1}$ is an action of G on itself.

Definition 37. Let H be a subgroup of G. Consider the action of H on G given by $h * g = g \cdot h^{-1}$. The orbits of the action are called the (left) cosets of H. The number of cosets is called the index of H and denoted as $[G : H]$.

For two groups K and H which are subgroups of G, the (K, H) double cosets are the orbits of G under the action of $K \times H$ on G defined as $(k, h) * g = k \cdot g \cdot h^{-1}$. Note that in general the (K, H) and (H, K) double coset are not equal.

Alternatively one could define cosets as sets of form $gH = \{g \cdot h \mid h \in H\}$ where $g \in G$, but Definition 37 allows us to employ Theorem 26.

For two groups K and H which are subgroups of G, the (K, H) double cosets are the orbits of G under the action of $K \times H$ on G defined as $(k, h) * g = k \cdot g \cdot h^{-1}$. Note that in general the (K, H) and (H, K) double coset are not equal.

Note that H is the orbit of any $h \in H$. Furthermore, the action is injective in the sense that $h * g = h' * g$ implies $h = h'$. Thus all orbits have equal cardinality, $|H|$. Combining this with Theorem 26 proves the following theorem.

Theorem 27 (Lagrange)**.** *If H is a subgroup of a finite group G, then $|G| = [G : H]|H|$. Especially $|H|$ divides $|G|$.*

Lagrange's theorem has a lot of important applications in group theory. For example, it is easy to see that no group of prime order has a proper subgroup.

A set T containing exactly one element from each coset of H is called a *transversal*. Each $g \in G$ has a unique representation as a product $t \cdot h$, $t \in T, h \in H$. Analogously a set T' containing one element of every orbit of G-set X is called an *orbit transversal*, and each $x \in X$ has a (not necessarily unique) representation $x = g * t$, $g \in G$, $t \in T'$. The following theorem gives a somewhat similar, but unique, representation.

Theorem 28 (Orbit-stabilizer theorem)**.** *Let G be a group acting on a set X. If $x \in X$ and $T \subseteq G$ is a transversal of $N_G(x)$, then the mapping $T \ni t \mapsto t * x$ is a bijection from T onto $G * x$. Thus, if G and X are finite, then*

$$|X| = \sum_{x \in O} [G : N_G(x)] = |G| \sum_{x \in O} \frac{1}{|N_G(x)|} \tag{A.0.7}$$

where $O \subseteq X$ contains exactly one element from each orbit.

Theorem 28 can be used to compute the total number of elements when one representative from each orbit is known, e. g., as a result of classification. The

following theorem on the other hand gives a formula for computing the number of orbits without explicitly determining the orbits or an orbit transversal.

Theorem 29 (Cauchy–Frobenius)**.** *Let G be a finite group acting on a finite set X. Define $fix_X(g) = |\{x \in X \mid g * x = x\}|$. The number of orbits is*

$$\frac{1}{|G|} \sum_{g \in G} fix_X(g) \tag{A.0.8}$$

Theorem 29 is often called Burnside's lemma.

Appendix B

Computational Complexity

One of the most important properties of an algorithm is the amount of time and memory required for executing it. These requirements naturally depend on the input. Unfortunately analyzing the running time or memory consumption will prove difficult if one seeks the exact solution, and the result will depend on several implementation-specific details. Accordingly it is customary to consider the rate of growth of the resource requirements.

Definition 38. Let $\mathcal{F}$ be the set of functions $f : \mathbb{N} \to \mathbb{N}$ and $g \in \mathcal{F}$. The set of functions having growth rate at most g is defined as

$$O(g) = \{f \in \mathcal{F} \mid \text{there exists } N, c, \text{such that } f(n) \leq cg(n) \text{ for } n \geq N\}. \tag{B.0.1}$$

The sets of functions having growth rate at least g and exactly g are defined, respectively, as

$$\Omega(g) := \{f \in \mathcal{F} \mid g \in O(f)\} \tag{B.0.2}$$

and

$$\Theta(g) := O(g) \cap \Omega(g). \tag{B.0.3}$$

It is common to use somewhat inaccurate notations such as $f = O(g)$ instead of $f \in O(g)$. The relation $\equiv$ defined by $f \equiv g$ if $f \in \Theta(g)$ is an equivalence relation.

In complexity theory it is common to consider *decision problems*, that is, problems of deciding whether the input has a prescribed property or not. Although most of the algorithmic problems of practical importance are not decision problems, they have natural decision problem variants. For example consider the task of finding a path from one given vertex to another. A natural decision problem variant would be to determine whether such a path exists. Typically this kind of simplification has no significant effect on the complexity of the problem.

A decision problem can be equated with the set of all inputs having the relevant property. Since in computer science we assume that all data is (or at least can be) represented in binary, we equate decision problems with sets of binary strings, called *languages*. All non-trivial (from a computational complexity point of view) languages consist of an infinite number of strings.

If an algorithm solves a decision problem, we say that it *accepts* the corresponding language.

For a binary string x let $|x|$ denote its length. The set of all finite binary strings is denoted by $\{0,1\}^*$. The size of a problem instance is its length when expressed as a binary string.

Definition 39. $\mathsf{TIME}(f(n))$ is the set of problems each of which can be solved with an algorithm that requires at most $g(n) = O(f(n))$ operations for solving a problem instance of size n.

Definition 40.

$$\mathsf{P} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(n^k) \tag{B.0.4}$$

The set P is a complexity class, that is, a set of computational problems connected by their difficulty. In the case of P, the problems are not very difficult computationally at least in theoretical sense.

The most simple and widely used, although somewhat controversial, definition of "efficient" algorithms states that an algorithm is efficient if its execution always terminates in a polynomial time with respect to the size of the input. According to this definition P is exactly the set of efficiently solvable problems. Hence one of the central problems of theoretical computer science is to determine what kinds of problems P includes, especially whether it includes the set NP defined below.

Definition 41. A relation $Q \subset \{0,1\}^* \times \{0,1\}^*$ is *polynomially balanced* if constants k, c exists such that $(x,y) \in Q$ implies $|y| \leq c|x|^k$. Q is *polynomially computable* if the problem of deciding whether $(x,y) \in Q$ belongs to P.

NP is the set of languages L such that $x \in L$ if and only if there exists y such that $(x,y) \in Q$, where the relation Q is polynomially balanced and polynomially computable.

NP is short for *nondeterministic polynomial*, since each language $L \in \mathsf{NP}$ is accepted by an algorithm that *nondeterministically* "guesses" the value of y and verifies (deterministically) that $(x,y) \in Q$, altogether in polynomial time with respect to $|x|$.

More precisely, a nondeterministic algorithm accepts a language L if the following holds: $x \in L$ if and only if some nondeterministic choices made while executing the algorithm on input x leads to an affirmative result.

The complexity class NP is interesting since most, although not all, computational problems of practical importance belong to NP. It is easy to see that $\mathsf{P} \subseteq \mathsf{NP}$. One of the most important open questions of theoretical computer science is whether $\mathsf{P} = \mathsf{NP}$ holds. In other words the question is whether all problems in NP can be solved efficiently, i. e., in polynomial time, with deterministic algorithms. Since in spite of extensive effort, no polynomial time algorithm has been found for certain NP problems, it is strongly believed that $\mathsf{P} \neq \mathsf{NP}$.

As an example of a NP problem we have the satisfiability problem involving Boolean expressions. A *literal* is either a Boolean variable or the negation of a Boolean variable. A *clause* is a Boolean expression that is a disjunction of literals. A Boolean expression is in *conjunctive normal form* if it consists of a conjunction of clauses.

Problem 6 (SAT or Satisfiability)**.** Given a Boolean expression in conjunctive normal form, determine whether the expression evaluates to True for some values of the variables.

To verify that SAT belongs to NP let y be a string describing the values of the variables and let $(x, y) \in Q$ if the expression x evaluates to True when the values of variables are given by y. It is straightforward to compute Q in polynomial time. Also $|y| \leq c|x|$ for some c since a string of n character may contain at most n Boolean variables.

The following special case of SAT is very useful in the study of NP.

Problem 7 (3SAT)**.** Given a Boolean expression in conjunctive normal form such that each clause consists of exactly three literals, determine whether the expression is satisfiable or not.

Definition 42. A *reduction* from a language L to a language L' is a mapping $R : \{0,1\}^* \to \{0,1\}^*$ such that $x \in L$ if and only if $R(x) \in L'$. Furthermore it is required that computing $R(x)$ requires at most $O(\log |x|)$ bits of memory.

If R is a reduction from L to L' and R' a reduction from L' to L'', then R and R' can be combined to obtain a reduction from L to L''. Proving this is not as easy as it might seem since the definition of reduction limits the memory, not time, of the computation, see [32, Propositien 8.2].

Since computing $R(x)$ requires at most $c \log |x|$ bits of memory, there are at most $2^{c \log |x|} = 2^c |x|^c$ states in which the algorithm may be during the computation. Hence the computation terminates in polynomial time.

If a reduction from L to L' exists and L' is accepted by some efficient algorithm, L is accepted by an efficient algorithm obtained by combining R and the algorithm for L'. Also if L is *not* accepted by any efficient algorithm, neither is L'. The former implication is useful in algorithm design and the latter in studying computational complexity.

Definition 43. A language L is NP-*complete* if $L \in \mathsf{NP}$, and in addition for any $L' \in \mathsf{NP}$ there exists a reduction from L' to L.

Completeness for other complexity classes is defined analogously. For some classes it is an open question whether any complete problems exist. However for NP there exists a great number of known complete problems.

The NP-complete problems are in computational complexity sense the most difficult of all NP problems. None of them can be solved in polynomial time unless P=NP and vice versa. Recall that it is believed that P≠NP. So far the best known algorithms for NP-complete problems require in worst cases exponential time to complete. Thus proving that a problem is NP-complete shows that it is difficult to solve in practical and probably also in theoretical sense.

Theorem 30 (Cook's theorem)**.** SAT *is* NP-*complete.*

Proof. See [32, Theorem 8.2]. □

Once the NP-completeness of a language L has been established, it may be used in proving NP-completeness of other languages as follows: if L can be reduced to L', any NP-language can be reduced to L' by combining reduction. To finish the NP-completeness proof for L' one needs to prove $L' \in \mathsf{NP}$ which is often trivial, but on some occasions difficult. This proof technique starting with Cook's theorem is used in virtually all NP-completeness proofs, Cook's theorem itself naturally excluded.

Another sign of the importance of Cook's theorem and the satisfiability problem is the fact that analogous results hold for many other complexity classes.

Still one more complexity class, #P, is required. This time the problems cannot be formulated as decision problems.

Definition 44. The complexity class #P consists of problems of following type: given x, compute the number of y such that $(x, y) \in Q$, where the relation Q is polynomially balanced and polynomially computable.

Clearly each problem in NP corresponds to a problem in #P which is at least as difficult as the original one. It is believed that problems in #P are in general more difficult than problems inNP. As soon will be seen, some NP-complete decision problems correspond to #P-complete counting problems. Quite surprisingly there are problems that are in P and whose counting variants are still #P-complete [32, Theorem 18.3].

As an example of a problem in #P we have #SAT, the problem of determining how many values of the variables satisfy a Boolean expression.

Since problems in #P are not decision problems, the definition of reduction needs slight modifications for the concept of #P -completeness to make sense.

Definition 45. A *reduction* from a problem P to a problem P' is a pair of mappings (R, S) such that for problem instance x of P, $R(x)$ is a problem instance of P'. In addition if N is the solution of $R(x)$, $S(N)$ is the solution of x. Furthermore both P and S can be computed in logarithmic space.

In the #P reductions encountered in this thesis S is the identity function $N \mapsto N$. In this case the reduction is called *parsimonious*.

Theorem 31. #SAT *is* #P*-complete.*

Proof. See [32, Theorem 18.1]. □

Theorem 32. 3SAT *is* NP*-complete and* #3SAT #P*-complete.*

Proof. Both claims are proved by a parsimonious reduction of SAT to 3SAT.

Each clause consisting of less than 3 literals can be made into equivalent 3-clause by repeating some literal. This transformation can be applied by 2-clauses produced by following replacements.

Replace each clause $(x_1 \vee \ldots \vee x_n)$, $n > 3$, with the clauses $(x_1 \vee y_1)$, $(\neg y_1 \vee x_2 \vee y_2)$, $\ldots$, $(\neg y_{n-1} \vee x_n)$ where $y_1, \ldots, y_{n-1}$ are new variables.

It is not hard to see that if some of the x_i is true, there are values of y_i such that the new clauses are satisfied, and conversely. However, if several of the x_i are true, then there are more satisfying truth assignments for the new version than for the original one.

To make the reduction parsimonious, add the clauses $(\neg x_i \vee \neg y_j) = (x_i \implies \neg y_j)$ for all $1 \leq i \leq j \leq n-1$. Now if k is the smallest integer such that $x_k = \textsc{True}$, then the new clauses are satisfied if and only if $y_i = \textsc{True}$ for $i < k$ and $y_i = \textsc{False}$ for $i \geq k$. Thus the reduction conserves satisfiability and is parsimonious.

It is not hard to see that the space requirement of performing the reduction is logarithmic in the number of variables. □

Bibliography

[1] L. Babai and E. M. Luks, *Canonical labeling of graphs*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 171–183.

[2] J. A. Barrau, *On the combinatory problem of Steiner*, Proceedings of the Section of Sciences, Koninklijke Akademie van Wetenschappen te Amsterdam **11** (1908), pp. 352–360.

[3] P. B. Bhattacharya, S. K. Jain, S. R. Nagpaul, *Basic Abstract Algebra, 2nd ed.*, Cambridge University Press, Cambridge, 1994.

[4] R. B. Boppana, J. Håstad and S. Zachos, *Does co-NP have short interactive proofs?* Inform. Process. Lett. **25** (1987), pp. 127–132.

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

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

[7] I. Diener, E. Schmitt and H. L. de Vries, *All* 80 *Steiner triple systems on* 15 *elements are derived*, Discrete Math. **55** (1985), pp. 13–19.

[8] J. Doyen, M. Vandensavel, *Non isomorphic Steiner quadruple systems*, Bull. Soc. Math. Belg. **23** (1971), pp. 393–410.

[9] G. Lo Faro, L. Milazzo, A. Tripodi, *On the upper and lower chromatic numbers of BSQSs*(16), Electron. J. Combin. **8** (2001), Research paper 6, 8 pp.

[10] P. B. Gibbons, R. Mathon, *On a new class of Steiner quadruple systems on* 16 *symbols*, Congr. Numer. **13** (1975), pp. 227–232.

[11] P. B. Gibbons, R. Mathon, D. G. Corneil, *Steiner quadruple systems on* 16 *symbols*, Congr. Numer. **14** (1975), pp. 345–365.

[12] H. Hanani, *On quadruple systems*, Canad. J. Math. **12** (1960), pp. 145–157.

[13] A. Hartman and K. T. Phelps, *Steiner quadruple systems*, in Contemporary design theory, J. H. Dinitz and D. R. Stinson, editors, Wiley, New York, 1992, pp. 205–240.

[14] P. Kaski and P. R. J. Östergård, *The Steiner triple systems of order* 19, Math. Comp. **73** (2004), pp. 2075–2092.

[15] P. Kaski, personal communication.

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

[17] P. Kaski, *Algorithms for Classification of Combinatorial Objects*, Doctoral Dissertation, Research report HUT-TCS-A94, Laboratory for Theoretical Computer Science, Helsinki University of Technology, 2005.

[18] T. P. Kirkman, *On a problem in combinatorics*, Cambridge Dublin Math. J. **2** (1847), pp. 191-204.

[19] D. E. Knuth, *Dancing links*, in Millennial Perspectives in Computer Science, J. Davies, B. Roscoe, and J. Woodcock, editors, Palgrave, Houndmills, 2000, pp. 187–214.

[20] H. Lenz, *On the number of Steiner quadruple systems*, Mitt. Math. Sem. Giessen **169** (1985), pp. 55–71.

[21] C. C. Lindner, A. Rosa, *There are at least* 31,021 *nonisomorphic Steiner quadruple systems of order* 16, Utilitas Math. **10** (1976), pp. 61–64.

[22] C. C. Lindner, A. Rosa, *Steiner Quadruple Systems—A survey*, Discrete Math. **22** (1978), pp. 147–181.

[23] A. C. H. Ling, C. J. Colbourn, M. J. Grannell, and T. S. Griggs, *Construction techniques for anti-Pasch Steiner triple systems*, J. London Math. Soc. (2) **61** (2000), pp. 641–657.

[24] R. Mathon, *A note on the graph isomorphism counting problem*, Inform. Process. Lett., **8** (1979), pp. 131–132.

[25] R. A. Mathon, K. T. Phelps, and A. Rosa, *Small Steiner triple systems and their properties*, Ars. Combin. **15** (1983), pp. 3–110; and **16** (1983), p. 286.

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

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

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

[29] B. D. McKay, autoson—*a distributed batch system for UNIX workstation networks (version 1.3)*, Technical report TR-CS-96-03, Computer Science Department, Australian National University, Canberra, 1996.

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

[31] N. S. Mendelsohn, S. H. Y. Hung, *On the Steiner systems* $S(3, 4, 14)$ *and* $S(4, 5, 15)$, Utilitas Math. **1** (1972), pp. 5–92.

[32] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, Mass., 1994.

[33] D. A. Spielman, *Faster isomorphism testing of strongly regular graphs*, Proc. 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, 1996, pp. 576–584.

[34] D. R. Stinson, *A comparison of two invariants for Steiner triple systems: fragments and trains*, Ars. Combin. **16** (1983), pp. 69–67.

[35] H. L. de Vries, *Some Steiner quadruple systems* $S(3, 4, 16)$ *such that all* 16 *derived triple systems* $S(2, 3, 15)$ *are isomorphic*, Ars Combin. **24A** (1987), pp. 107–129.

[36] H. S. White, F. N. Cole, and L. D. Cummings, *Complete classification of triad systems of fifteen elements*, Memoirs Nat. Acad. Sci. U.S.A **14** (1919), pp. 1–89.

[37] V. A. Zinoviev and D. V. Zinoviev, *Binary extended perfect codes of length* 16 *obtained by a generalized concatenated construction (in Russian)*, Problemy Peredachi Informatsii **38** (2002), pp. 56–84, translation in Probl. Inf. Transm. **38** (2002), pp. 296–322.

[38] V. A. Zinoviev and D. V. Zinoviev, *Binary perfect codes of length* 15 *obtained by a generalized concatenated construction (in Russian)*, Problemy Peredachi Informatsii **40** (2004), pp. 27–39, translation in Probl. Inf. Transm. **40** (2004), pp. 25–36.

[39] V. A. Zinoviev and D. V. Zinoviev, *Classification of Steiner's quadruple systems of order* 16 *and of rank at most* 13 *(in Russian)*, Problemy Peredachi Informatsii **40** (2004), pp. 48–67, translation in Probl. Inf. Transm. **40** (2004), pp. 337–355.

[40] V. A. Zinoviev and D. V. Zinoviev, *Binary extended perfect codes of length* 16 *and rank* 14, preprint.

[41] V. A. Zinoviev and D. V. Zinoviev, *Classification of Steiner quadruple systems of order* 16 *and rank* 14, preprint.

[42] V. A. Zinoviev and D. V. Zinoviev, *Vasiliev codes of length* $n = 2^m$ *and Steiner systems* $S(n, 4, 3)$ *of rank* $n - m$ *over* F_2, preprint.

[43] V. A. Zinoviev, personal communication.