Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 78
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 78
Espoo 2003 HUT-TCS-A78

A CENSUS OF STEINER TRIPLE SYSTEMS AND SOME RELATED COMBINATORIAL OBJECTS

Petteri Kaski

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

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 78
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 78
Espoo 2003 HUT-TCS-A78

A CENSUS OF STEINER TRIPLE SYSTEMS AND SOME RELATED COMBINATORIAL OBJECTS

Petteri Kaski

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

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

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

© Petteri Kaski

ISBN 951-22-6645-8
ISSN 1457-7615

2003

ABSTRACT: A *Steiner triple system* of order v ($\mathrm{STS}(v)$) is a set of triples, or *blocks*, constructed over a set of v *points*, such that every pair of distinct points occurs in a unique block.

Previously, a complete classification of the $\mathrm{STS}(v)$ up to isomorphism was known only for $v \leq 15$. In this work, we classify by computer search the next open case, $v = 19$. The classification proceeds in two stages. First, we construct an initial set of 25-block seed configurations. Then, using an algorithm for the exact cover problem, we determine all completions of the seeds to STS(19). Isomorph rejection on the STS(19) is carried out using the graph canonical labelling package *nauty* supplemented with a vertex invariant based on Pasch configurations.

We conclude that there are 11,084,874,829 nonisomorphic STS(19) and study a number of their properties. In particular, the number of anti-Pasch STS(19) is 2,591 and the number of STS(19) with a nontrivial automorphism group is 164,758.

We also develop an independent algorithm for classifying STS(19) with a prescribed group of automorphisms. We then use this algorithm to classify the STS(19) with a nontrivial automorphism group. The results obtained in this partial classification match those obtained in the main search.

Finally, we show that the main classification algorithm can be used with minor modifications to classify certain related combinatorial structures, such as latin squares and one-factorizations of complete graphs. We estimate the performance of the algorithm in classifying one-factorizations of the complete graph on 12 vertices.

KEYWORDS: classification algorithm, isomorph rejection, one-factorization, Pasch configuration, Steiner triple system

CONTENTS

1 Introduction **1**

2 Background **3**
2.1 Definitions and notation . . . 3
Basic notation . . . 3
Order and partitions . . . 4
Group theory . . . 4
Graphs . . . 6
Incidence structures and designs . . . 6
2.2 Theoretical results . . . 9
Basic results on Steiner triple systems . . . 9
Subsystems of Steiner triple systems . . . 11
Block graphs of Steiner triple systems . . . 12
Pasch configurations in Steiner triple systems . . . 14
2.3 Isomorph-free exhaustive generation . . . 15

3 Auxiliary algorithms **16**
3.1 Exact cover . . . 16
Algorithm DLX . . . 17
3.2 Graph isomorphism . . . 20
Canonical representatives and canonical labelling . . . 20
Canonical labelling of graphs with *nauty* . . . 21
Canonical labelling of incidence structures . . . 22
3.3 Algorithms for permutation groups . . . 23

4 Constructing the STS(19) **23**
4.1 Conventions and assumptions . . . 24
4.2 Seeds and the top-level algorithm . . . 24
4.3 Generation of seeds . . . 27
4.4 Isomorph rejection . . . 28
Parallelization . . . 29
Elimination of hash table queries . . . 30
Implementation details . . . 32
4.5 The search . . . 36

5 Prescribing a group of automorphisms **37**
5.1 Automorphisms of STS(19) . . . 37
5.2 The construction algorithm . . . 39
5.3 Starting points for the search . . . 39
5.4 The search . . . 42

6 The Steiner triple systems of order 19 **42**
6.1 A census of the STS(19) . . . 43
6.2 Systems with a nontrivial automorphism group . . . 44
6.3 Pasch configurations . . . 45
6.4 Discussion . . . 48

7 Construction of other structures **52**
7.1 One-factorizations of K_{2n} 53
7.2 Group divisible designs with block size 3 55
Latin squares 56
Generalized Steiner triple systems 56
7.3 A performance estimate 57

References **60**

1 INTRODUCTION

A *Steiner triple system* of order v (STS(v)) is a set of triples, or *blocks*, constructed over a set of v *points*, such that every pair of distinct points occurs in a unique block. Two STS(v) are *isomorphic* if there exists a bijection between the point sets that maps blocks onto blocks.

As early as in 1847, Kirkman [50][1] settled the existence problem for STS(v). Namely, a Steiner triple system of order v exists if and only if $v \equiv 1 \pmod 6$ or $v \equiv 3 \pmod 6$.

The task of classifying the nonisomorphic STS(v) is much harder. Indeed, despite extensive research, a complete classification of the nonisomorphic STS(v) is known only for a handful of the smallest admissible parameter values v. This is largely because $N(v)$, the number of nonisomorphic STS(v), grows very rapidly with increasing v. More specifically, Wilson [86] (see also [25, 27]) has proved that $N(v) \geq (e^{-5}v)^{v^2/6}$.

For the two smallest nontrivial values, $v = 7$ and $v = 9$, the STS(v) are unique up to isomorphism. For $v = 13$, the STS(v) partition into two isomorphism classes. The existence of these classes was shown by Zulauf [87]. De Pasquale [21] and Brunel [10] established that these two classes form the complete classification. For $v = 15$, there are 80 isomorphism classes, a result first obtained by Cole, Cummings, and White [20, 85] in 1917 after a very laborous manual calculation. Their result was verified in a computer search conducted by Hall and Swift [40] in 1955. A complete listing of the STS(v) for $v \leq 15$ appears in, for example, [17, 59].

For $v = 19$ the exact number of nonisomorphic STS(v) has been unknown until now due to their huge number. Lower bounds and estimates for $N(19)$ have been computed by a number of researchers. These include Stinson and Ferch [78], who obtained $N(19) \geq 2{,}395{,}687$ and estimated $N(19) \approx 10^9$. McKay [59] improved the lower bound to $N(19) \geq 1.1 \cdot 10^9$, and, using a sophisticated estimation method, obtained a (correct) estimate $1.1 \cdot 10^{10} \leq N(19) \leq 1.2 \cdot 10^{10}$ [63].

While a complete classification of the STS(19) has been lacking, research has focused on STS(19) with special properties. Such partial classifications of STS(19) include at least the following. Bays [4] enumerated the four STS(19) having a cyclic automorphism. Denniston [22] classified the 184 nonisomorphic STS(19) with a reversal, that is, an automorphism with nine 2-cycles and one fixed point. Phelps and Rosa [67] classified the 10 nonisomorphic STS(19) having a 2-rotational automorphism, that is, an automorphism with two 9-cycles and one fixed point. Stinson and Seah [79] classified the 284,457 STS(19) with a subsystem of order 9. Colbourn, Magliveras, and Stinson [16] classified all the STS(19) that admit a nontrivial automorphism. (Unfortunately, the results in [16] are partly in error as we shall demonstrate.)

In this report, we develop a search algorithm for classifying STS(v) and use it to classify the STS(19) up to isomorphism by computer search. As a result of the search, which required approximately two years of total CPU

[1] A comprehensive reference to Steiner triple systems with a historical introduction is the book by Colbourn and Rosa [17]; the nineteenth century and early twentieth century references in this report are quoted from Colbourn and Rosa. Ref. [20] is the only exception to this rule.

time, we conclude that

$$N(19) = 11{,}084{,}874{,}829$$

and that the number of distinct STS(19) over a fixed point set is

$$1{,}348{,}410{,}350{,}618{,}155{,}344{,}199{,}680{,}000.$$

This number can be computed from the search data in two different ways, which supports the correctness of the classification. To gain further confidence in the classification, we classify the STS(19) with a nontrivial full automorphism group using an independent algorithm. Chapter 6 describes the classification in more detail. In particular, we correct the erroneous results in [16] and study the number of Pasch configurations in STS(19).

There are a number of reasons that motivate a complete classification of the STS(19). First, a complete classification enables in principle the study of any property of the STS(19). (In practice, the large number of nonisomorphic STS(19) may still make the study of some properties computationally infeasible.) Second, the study of a complete classification of a collection of combinatorial objects often opens up lines of research concerning the structure of a larger class of objects, often in the form of a counterexample to a conjecture. In this sense, the STS(19) are sufficiently "rich" in numbers and structure compared with the systems of lesser order. For example, the classification of the STS(19) enabled the recent discovery of nonisomorphic STS(19) with equivalent point codes [47]. For $v \leq 15$, no such STS(v) exist [82]. We expect that this will not be the last such discovery.

The rest of this report is organized as follows. Chapter 2 contains a brief review of the background material such as the terminology used and theoretical results from the literature.

Chapter 3 covers the auxiliary algorithms required by the search algorithms in the subsequent Chapters 4 and 5. The exact cover algorithm we employ is Algorithm DLX due to Knuth [52]. The isomorph rejection scheme requires computation of the automorphism group and canonical labelling of block graphs of STS(19). For this task we use the package *nauty* due to McKay [61] supplemented with a vertex invariant based on the Pasch configurations of the underlying STS(19).

Chapter 4 contains a detailed description of the classification algorithm for STS(19) and the implementation of the computer search. Our approach to classifying the STS(19) is to regard the construction of STS as an instance of the exact cover problem, in which the task is to cover the $\binom{19}{2} = 171$ pairs of points in all possible ways using 57 triples from a set of $\binom{19}{3} = 969$ triples. We start the exact cover search from a restricted collection of 14,648 partial solutions with 25 triples, which we call *seeds*, and which are known to occur (up to isomorphism) in every STS(19). By starting the search from the seeds we obtain the necessary symmetry reduction in the search space that enables the construction of a representative from every isomorphism class of STS(19) in a manageable amount of time. (With no symmetry reduction we would have to search through all the distinct STS(19), which is obviously infeasible.) We filter isomorphic STS(19) from the output of the exact cover algorithm using two tests, which are motivated by a general framework for isomorph-free exhaustive generation [63].

Chapter 5 describes an independent algorithm for classifying the STS(19) with a nontrivial automorphism group. The approach taken is essentially that of [16], but our method of constructing and performing isomorph rejection on the STS(19) is different. Namely, we formulate the construction problem again as an exact cover problem, in which we cover orbits of pairs with orbits of triples, where the orbits are given by a prescribed group of automorphisms. Careful design of the starting points of the search is again required to make the search feasible. The use of an exact cover algorithm in constructing STS(19) with a prescribed group of automorphisms can be seen as a special case of the technique pioneered by Kramer and Mesner [55].

We conclude this report in Chapter 7 by demonstrating that the classification approach derived for STS(19) can be used essentially without change to classify other combinatorial structures such as latin squares, one-factorizations of complete graphs, and generalized Steiner triple systems. Whether this approach will result in more efficient algorithms than the ones already in existence for these structures remains a subject of future work. Initial estimates on the time required to classify the nonisomorphic one-factorizations of the complete graph on 12 vertices suggest a factor 2.5 performance improvement compared with the algorithm used by Dinitz, Garnick, and McKay [23].

2 BACKGROUND

This chapter summarizes the relevant background material used in the main subject matter of this report. The chapter is organized into two parts.

The first part, Sections 2.1 and 2.2, contains the standard mathematical definitions and theoretical results used in the main subject matter. We assume familiarity with basic mathematical concepts and some knowledge of group theory. Any definitions not given here can be found in the following references. Textbooks in combinatorics include [12, 83]. A comprehensive reference to Steiner triple systems is [17]. Standard design theory references are [5, 14]. Textbooks on group theory include [42, 72]; permutation groups are discussed in [13, 24].

The second part, Section 2.3, contains a brief discussion on algorithms for isomorph-free exhaustive generation of combinatorial structures. Textbooks on combinatorial algorithms include [56, 71].

2.1 Definitions and notation

This section briefly summarizes the standard mathematical definitions and notation used in this report.

Basic notation

The cardinality of a finite set X is denoted by $|X|$. Let X, Y be sets. We write $X \times Y$ for the Cartesian product of X and Y, that is, the set of all ordered pairs (x, y) with $x \in X$ and $y \in Y$. We write $f : X \to Y$ to indicate that f is a function of X into Y. We write either $f : x \mapsto y$ or $f(x) = y$ to indicate that f maps $x \in X$ to $y \in Y$. The restriction of a function $f : X \to Y$ into a set $Z \subseteq X$ is denoted by $f|_Z$.

Order and partitions

Lexicographic order. Let X be a nonempty set with a total order $<$ and let $E, F \subseteq X$. Then, the *lexicographic order* on the set of all nonempty subsets of X is defined by $E < F$ if and only if there exists an $x \in X$ such that

(i) $x \in F$; and

(ii) $x \notin E$; and

(iii) for all $y \in X$ such that $y < x$ we have $x \in E$ if and only if $x \in F$.

We write $E \leq F$ if and only if either $E = F$ or $E < F$.

Partition, ordered partition. A *partition* of a nonempty set V is a set of pairwise disjoint nonempty subsets of V whose union is V. An *ordered partition* of V is a sequence $(V_1, \ldots, V_n)$ of nonempty subsets of V such that $\{V_1, \ldots, V_n\}$ is a partition of V. We denote by $\Pi(V)$ the set of all ordered partitions of V.

The elements of a partition are called its *cells*. The partition that consists of only the cell V is called the *unit* partition. An ordered partition is *discrete* if all of its cells are singleton sets.

Group theory

Group, subgroup. A finite *group* consists of a finite set G and a mapping $\cdot : G \times G \to G$, called the *group operation*, that satisfies the following three axioms:

(G1) $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$ for all $g_1, g_2, g_3 \in G$;

(G2) there exists an element $\mathbf{I} \in G$ such that $g \cdot \mathbf{I} = \mathbf{I} \cdot g = g$ for all $g \in G$;

(G3) for all $g \in G$ there exists an element $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = \mathbf{I}$.

We omit the group operation in what follows and write simply $g_1 g_2$ in place of $g_1 \cdot g_2$.

A subset $H \subseteq G$ of a group G is a *subgroup* of G if H is a group under the group operation of G restricted to H. We write $H \leq G$ to indicate that H is a subgroup of G.

Permutation, permutation group. Let X be a finite nonempty set. A *permutation* of X is a bijection of X onto itself. The *degree* of a permutation of X is $|X|$. Let α, β be two permutations of X. Their *composition* $\alpha\beta$, defined for all $x \in X$ by $\alpha\beta(x) = \alpha(\beta(x))$, is a permutation of X.

The set of all permutations of X forms a group with the composition of permutations as the group operation. This group is called the *symmetric group* on X and we denote it by $\mathrm{Sym}(X)$. A *permutation group* on X is a subgroup of $\mathrm{Sym}(X)$. The *degree* of a permutation group on X is $|X|$.

Let $\alpha \in \mathrm{Sym}(X)$ and $\beta \in \mathrm{Sym}(Y)$, $X \cap Y = \emptyset$. The *sum* $\alpha + \beta$ is the permutation of $X \cup Y$ defined by

$$(\alpha + \beta)(z) = \begin{cases} \alpha(z) & \text{if } z \in X; \text{ and} \\ \beta(z) & \text{if } z \in Y. \end{cases}$$

Similarly, if $G \leq \mathrm{Sym}(X)$ and $H \leq \mathrm{Sym}(Y)$ are permutation groups, then the sum $G + H$ is the permutation group

$$G + H = \{\alpha + \beta \ : \ \alpha \in G, \beta \in H\} \leq \mathrm{Sym}(X \cup Y).$$

A permutation $\alpha \in \mathrm{Sym}(X)$ *fixes* $x \in X$ if $\alpha(x) = x$; otherwise α *moves* x. Two permutations $\alpha, \beta \in \mathrm{Sym}(X)$ are *disjoint* if every point moved by one fixed by the other.

A permutation $\alpha \in \mathrm{Sym}(X)$ is a *cycle* if there exist a positive integer k and distinct $x_1, \ldots, x_k \in X$ such that

$$\alpha(x_1) = x_2, \quad \alpha(x_2) = x_3, \quad \cdots, \quad \alpha(x_{k-1}) = x_k, \quad \alpha(x_k) = x_1 \qquad (1)$$

and $\alpha(x) = x$ for all $x \in X \setminus \{x_1, \ldots, x_k\}$. A cycle is *trivial* if $k = 1$. The integer k is called the *length* of the cycle; a cycle of length k is a *k-cycle*.

We denote a k-cycle that satisfies (1) by $(x_1, x_2, \ldots, x_k)$. The identity permutation is denoted by $\mathbf{I}$.

A permutation $\alpha \in \mathrm{Sym}(X)$ factors uniquely (up to ordering of the factors) into a product of pairwise disjoint cycles such that there is exactly one trivial cycle in the product for each $x \in X$ fixed by α. The *cycle type* of α is $a_1^{n_1} a_2^{n_2} \cdots a_m^{n_m}$ if $\sum_{i=1}^{m} n_i a_i = |X|$ and the factorization of α into disjoint cycles contains exactly n_i cycles of length a_i for all $i \in \{1, \ldots, m\}$.

Generators. Let G be a group and let $S \subseteq G$. Then, the group *generated* by S is the intersection of all subgroups $H \leq G$ such that $S \subseteq H$. If $H \leq G$ is the group generated by S, then we say that S is a set of *generators* for H. We denote by $\langle S \rangle$ the group generated by S.

Group action, orbit, stabilizer. Let G be a group and let X be a finite nonempty set. An *action* of G on X is a mapping $\gamma : G \times X \to X$ that satisfies the following two axioms:

(A1) $\gamma(\mathbf{I}, x) = x$ for all $x \in X$;

(A2) $\gamma(g_1 g_2, x) = \gamma(g_1, \gamma(g_2, x))$ for all $g_1, g_2 \in G$ and $x \in X$.

Whenever it is not necessary to explicitly specify the group action (or the action is clear from the context), we will write either $g(x)$ or simply gx in place of $\gamma(g, x)$ and say only that $g \in G$ *acts* on $x \in X$. In the case of a permutation group $G \leq \mathrm{Sym}(X)$ the action is always assumed to be the natural permutation action on X given by $\gamma(\alpha, x) = \alpha(x)$ for all $\alpha \in G$ and $x \in X$.

A group action on X induces an elementwise action on sets or tuples built from the elements of X. For example, the action induced on the set of all subsets X is given by $g(E) = \{g(x) \ : \ x \in E\}$ for all $g \in G$ and all $E \subseteq X$.

Let G act on X and let $x \in X$. The *orbit* of x under G is the set

$$Gx = \{g(x) \ : \ g \in G\}.$$

The subgroup

$$G_x = \{g \in G \ : \ g(x) = x\}$$

is called the *stabilizer* of x in G.

A group action of G is *transitive* on X if G has only one orbit on X. The action is *k-transitive* on X if the action induced on the set of all k-tuples of distinct elements of X is transitive.

Graphs

Graph, subgraph. A *graph* is a pair (V, E) consisting of a set V of *vertices* and a set E, disjoint from V, of 2-subsets of V called *edges*. We denote the vertex set of a graph X by $V(X)$ and the edge set by $E(X)$. A graph with vertex set V is called a graph *on* V.

A vertex v is *incident* with an edge e if $v \in e$. The two vertices incident with an edge are its *endvertices* or *ends*, and an edge *joins* its ends. Two vertices v, w joined by an edge are *adjacent* or *neighbors*.

The number of vertices of a graph is its *order*. The *degree* of a vertex v is the number of neighbors of v. A graph X is *regular* of degree d if the degree of every vertex is d.

A graph Y is a *subgraph* of a graph X if $V(Y) \subseteq V(X)$ and $E(Y) \subseteq E(X)$.

Isomorphism. Two graphs X, Y are *isomorphic* if there exists a bijection $\varphi : V(X) \to V(Y)$ such that, for all $u, v \in V(X)$,

$$\{u, v\} \in E(X) \qquad \Leftrightarrow \qquad \{\varphi(u), \varphi(v)\} \in E(Y).$$

Such a map φ is an *isomorphism* of X onto Y; if $X = Y$ then φ is an *automorphism* of X.

The set of all automorphisms of X forms a group with the composition of permutations as the group operation. This group is called the *(full) automorphism group* of X. We denote the full automorphism group of X by $\mathrm{Aut}(X)$.

One-factor, one-factorization. A *one-factor* of a graph X is a subset $F \subseteq E(X)$ such that each vertex $v \in V(X)$ is incident with exactly one edge in F. Alternatively, a one-factor of X is a 1-regular subgraph Y such that $V(Y) = V(X)$. A *one-factorization* of a graph X is a partition of $E(X)$ into one-factors.

Two sets of one-factors of a graph X, $\mathcal{F}$ and $\mathcal{F}'$, are *isomorphic* if there exists a permutation φ of $V(X)$ that maps the one-factors in $\mathcal{F}$ onto one-factors in $\mathcal{F}'$. Such a permutation φ is an *isomorphism* of $\mathcal{F}$ onto $\mathcal{F}'$.

Complete graph, clique. A graph is *complete* if all of its vertices are pairwise adjacent. We denote a complete graph on n vertices by K_n. A *clique* in a graph X is a complete subgraph of X. A clique is *maximal* if it is not a subgraph of a clique that has strictly larger order. A *maximum* clique is a clique of maximum order, taken over all cliques in the graph.

Strongly regular graph. A *strongly regular graph* $\mathrm{srg}(n, d, \lambda, \mu)$ is a graph of order n that is regular of degree d and that has the following two properties:

(S1) For any two adjacent vertices x, y, there are exactly λ vertices adjacent to both x and y.

(S2) For any two nonadjacent vertices x, y, there are exactly μ vertices adjacent to both x and y.

Incidence structures and designs

Incidence structure, induced substructure. An *incidence structure* is a triple $(\mathcal{P}, \mathcal{B}, \mathcal{I})$, where $\mathcal{P}$ is a nonempty set of *points*, $\mathcal{B}$ is a nonempty set (disjoint from $\mathcal{P}$) of *blocks*, and $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{B}$ is the incidence relation; a point

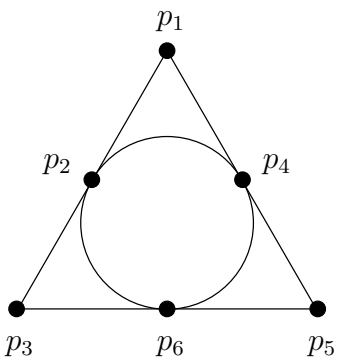
p is *incident* with a block B if $(p, B) \in \mathcal{I}$. Two blocks *intersect* if they are incident with a common point. Conversely, two blocks are *disjoint* if they are not incident with a common point. For an incidence structure $\mathcal{X}$ we write $\mathcal{P}(\mathcal{X}), \mathcal{B}(\mathcal{X}), \mathcal{I}(\mathcal{X})$ for the point set, the block set, and the incidence relation of $\mathcal{X}$, respectively.

An incidence structure is *simple* if no two distinct blocks are incident with the same set of points.

Example 2.1 An incidence structure that we shall encounter often in this report is the *Pasch configuration* (alternatively, a *fragment* or *quadrilateral*), which is the following incidence structure with six points $p_1, p_2, p_3, p_4, p_5, p_6$ and four blocks B_1, B_2, B_3, B_4:

	p_1	p_2	p_3	p_4	p_5	p_6
B_1	•	•	•			
B_2	•			•	•	
B_3		•		•		•
B_4			•		•	•

An alternative pictorial representation is given below.



A set of blocks $\mathcal{D} \subseteq \mathcal{B}(\mathcal{X})$ in an incidence structure $\mathcal{X}$ *induces* the incidence structure $\mathcal{X}'$ defined by

$$\mathcal{P}(\mathcal{X}') = \{p \in \mathcal{P}(\mathcal{X}) \ : \ (p, B) \in \mathcal{I}(\mathcal{X}) \text{ for some } B \in \mathcal{D}\}, \qquad \mathcal{B}(\mathcal{X}') = \mathcal{D},$$

and

$$(p, B) \in \mathcal{I}(\mathcal{X}') \qquad \Leftrightarrow \qquad (p, B) \in \mathcal{I}(\mathcal{X})$$

for all $p \in \mathcal{P}(\mathcal{X}')$ and $B \in \mathcal{B}(\mathcal{X}')$.

Isomorphism. Two incidence structures, $\mathcal{X}$ and $\mathcal{Y}$, are *isomorphic* if there exists a bijection $\varphi : \mathcal{P}(\mathcal{X}) \cup \mathcal{B}(\mathcal{X}) \to \mathcal{P}(\mathcal{Y}) \cup \mathcal{B}(\mathcal{Y})$ such that $\varphi(\mathcal{P}(\mathcal{X})) = \mathcal{P}(\mathcal{Y})$, $\varphi(\mathcal{B}(\mathcal{X})) = \mathcal{B}(\mathcal{Y})$, and for all $p \in \mathcal{P}(\mathcal{X})$ and $B \in \mathcal{B}(\mathcal{X})$ we have

$$(p, B) \in \mathcal{I}(\mathcal{X}) \qquad \Leftrightarrow \qquad (\varphi(p), \varphi(B)) \in \mathcal{I}(\mathcal{Y}).$$

Such a bijection φ is an *isomorphism* of $\mathcal{X}$ onto $\mathcal{Y}$; if $\mathcal{X} = \mathcal{Y}$ then φ is an *automorphism* of $\mathcal{X}$.

The *(full) automorphism group* of $\mathcal{X}$, denoted by $\mathrm{Aut}(\mathcal{X})$, is the group formed by the set of all automorphisms of $\mathcal{X}$ with composition of permutations as the group operation. Clearly, $\mathrm{Aut}(\mathcal{X}) \leq \mathrm{Sym}(\mathcal{P}(\mathcal{X})) + \mathrm{Sym}(\mathcal{B}(\mathcal{X}))$.

A *point automorphism* of $\mathcal{X}$ is the restriction of an automorphism of $\mathcal{X}$ to $\mathcal{P}(\mathcal{X})$; the *point automorphism group* is the group formed by all point automorphisms. Block automorphisms and the block automorphism group are defined analogously.

Dual incidence structure. Let $\mathcal{X}$ be an incidence structure. The *dual* of $\mathcal{X}$ is the incidence structure obtained by exchanging the roles of points and blocks. More formally, the dual of $\mathcal{X}$ is the incidence structure $\mathcal{X}'$, where $\mathcal{P}(\mathcal{X}') = \mathcal{B}(\mathcal{X})$, $\mathcal{B}(\mathcal{X}') = \mathcal{P}(\mathcal{X})$, and for all $p \in \mathcal{P}(\mathcal{X})$ and $B \in \mathcal{B}(\mathcal{X})$,

$$(B, p) \in \mathcal{I}(\mathcal{X}') \qquad \Leftrightarrow \qquad (p, B) \in \mathcal{I}(\mathcal{X}).$$

Point graph, block graph, incidence graph. The *point graph* of an incidence structure is the graph whose vertices are the points of the incidence structure: any two points are joined by an edge if and only if the points are incident with a common block.

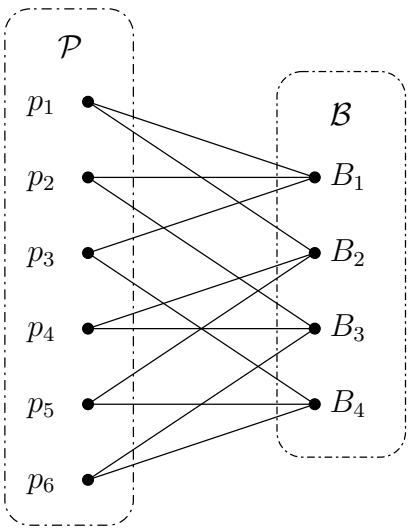
The *block graph* of an incidence structure is the graph whose blocks are the points of the incidence structure: any two blocks are joined by an edge if and only if the blocks intersect. In other words, the block graph of an incidence structure is the point graph of the dual incidence structure.

The *incidence graph* of an incidence structure $\mathcal{X}$ is the graph X with $V(X) = \mathcal{P}(\mathcal{X}) \cup \mathcal{B}(\mathcal{X})$ and edge set defined by

$$\{p, B\} \in E(X) \qquad \Leftrightarrow \qquad (p, B) \in \mathcal{I}(\mathcal{X}) \tag{2}$$

for all $p \in \mathcal{P}(\mathcal{X})$ and $B \in \mathcal{B}(\mathcal{X})$; these are the only edges in $E(X)$, that is, no two vertices from either $\mathcal{P}(\mathcal{X})$ or $\mathcal{B}(\mathcal{X})$ are adjacent in X.

Example 2.2 The incidence graph of the Pasch configuration (Example 2.1) is displayed below.



Partial geometry. A *partial geometry* $\mathrm{pg}(k, r, t)$ is an incidence structure with the following four properties:

(P1) Any two distinct points are incident with at most one common block.

(P2) Every block is incident with exactly k points.

(P3) Every point is incident with exactly r blocks.

(P4) Let B be a block and p a point such that p is not incident with B. Then, there exist exactly t blocks that are incident with p and that intersect B.

Example 2.3 The Pasch configuration in Example 2.1 is a partial geometry $\mathrm{pg}(3, 2, 1)$.

Example 2.4 An $\mathrm{STS}(v)$ is a partial geometry $\mathrm{pg}(3, (v-1)/2, 3)$.

Designs. A t-(v, k, λ) *design* is an incidence structure with the following three properties:

(T1) The point set consists of v points.

(T2) Every block is incident with exactly k points.

(T3) Every t-subset of points is incident with exactly λ blocks.

Example 2.5 An $\mathrm{STS}(v)$ is a 2-$(v, 3, 1)$ design.

Group divisible designs. Let K be a finite nonempty set of positive integers and let λ be a positive integer. A *group divisible design* of index λ and order v (a (K, λ)-GDD) is an incidence structure with the following three properties:

(D1) The point set consists of v points.

(D2) For every block $B \in \mathcal{B}$, $|B| \in K$, where $|B|$ denotes the number of points incident with B.

(D3) There exists a partition $\mathcal{G}$, the cells of which are called *groups*, of the point set such that every pair of distinct points occurs either in exactly λ blocks, in which case the points belong to different groups, or no block contains the point pair, in which case the points belong to the same group.

If $K = \{k\}$, then we write (k, λ)-GDD in place of (K, λ)-GDD. Furthermore, if $\lambda = 1$, then we write either K-GDD or k-GDD.

The sizes of the groups in $\mathcal{G}$ form the *(group) type* of the GDD. More specifically, if $v = \sum_{i=1}^{m} n_i a_i$ and $\mathcal{G}$ contains n_i groups of size a_i, then the group type of the GDD is $a_1^{n_1} a_2^{n_2} \cdots a_m^{n_m}$. A GDD is *uniform* if all groups have the same size. A *transversal design* $\mathrm{TD}(k, n)$ is a k-GDD of type n^k.

Example 2.6 The Pasch configuration in Example 2.1 is a $\mathrm{TD}(3, 2)$. The partition into groups is $\{\{p_1, p_6\}, \{p_2, p_5\}, \{p_3, p_4\}\}$.

2.2 Theoretical results

Basic results on Steiner triple systems

All results in this section are elementary and well-known. In the language of incidence structures, a *Steiner triple system* of *order* v ($\mathrm{STS}(v)$) is a 2-$(v, 3, 1)$ design.

Theorem 2.7 *Every point of an* $\mathrm{STS}(v)$ *is incident with exactly* r *blocks, and the total number of blocks in an* $\mathrm{STS}(v)$ *is* b*, where*

$$r = (v-1)/2, \qquad b = v(v-1)/6.$$

Proof. Let x be an arbitrary point. There are $v-1$ unordered pairs of distinct points that contain x. Associated with each such pair is the unique block that contains the pair. These blocks form a list in which each block of the form $\{x, y, z\}$ occurs twice, once for the pair $\{x, y\}$ and once for the pair $\{x, z\}$. Thus, x occurs in a total of $r = (v-1)/2$ blocks. For the second equality we obtain $3b = vr$ by counting the total occurences of points in the blocks in two different ways. □

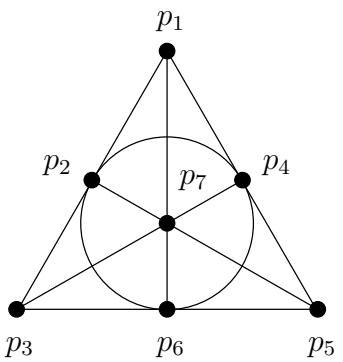
Since both r and b must be integers, we obtain the following necessary existence conditions.

Corollary 2.8 *An* $\mathrm{STS}(v)$ *exists only if* $v \equiv 1 \pmod 6$ *or* $v \equiv 3 \pmod 6$.

These conditions are also sufficient; see [17] for numerous recursive and direct constructions of $\mathrm{STS}(v)$. The smallest nontrivial values for which an $\mathrm{STS}(v)$ exists are $v \in \{7, 9, 13, 15, 19, 21, 25, 27, \ldots\}$.

We give examples of Steiner triple systems of small order.

Example 2.9 The $\mathrm{STS}(7)$ is perhaps most easily described by the following figure.



Example 2.10 The $\mathrm{STS}(9)$ can be constructed from a 3×3 matrix with nine distinct entries by listing the rows, columns, forward diagonals, and back diagonals of the matrix.

Example 2.11 There are two nonisomorphic $\mathrm{STS}(13)$. The first is perhaps most easily described over the point set $\mathbb{Z}_{13} = \{0, 1, \ldots, 12\}$. The blocks are

$$\mathbb{Z}_{13} + \{1, 3, 9\}, \qquad \mathbb{Z}_{13} + \{2, 5, 6\},$$

where addition is performed elementwise modulo 13. The second $\mathrm{STS}(13)$ is shown in the following table, where each column gives a block over $\mathbb{Z}_{13}$.

0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	3	3	3	4	4	4	5	5	5	6
1	3	5	7	9	11	3	4	6	9	10	3	4	6	7	8	6	7	8	6	8	10	7	8	9	7
2	4	6	8	10	12	5	7	8	11	12	9	5	10	12	11	11	10	12	12	9	11	11	10	12	9

Example 2.21 describes one of the 80 nonisomorphic STS(15). For a complete listing of all the STS(v) for $v \leq 15$, see [17, 59].

Up to isomorphism we may regard a Steiner triple system either as a set of subsets of the point set (in which case the blocks are *unlabelled*) or as an incidence structure (in which case the blocks are *labelled*). We shall alternate between these two representations and use whichever is more convenient for the task at hand.

It is perhaps not immediate that a change of representation does not affect the automorphism group. Namely, the automorphism group of a block unlabelled STS is the point automorphism group of the block labelled STS. These two are isomorphic as the following elementary theorem demonstrates.

Theorem 2.12 *Let $\mathcal{X}$ be a simple incidence structure. Then,*

(i) every point automorphism of $\mathcal{X}$ has a unique extension to an automorphism of $\mathcal{X}$; and

(ii) the point automorphism group of $\mathcal{X}$ and the automorphism group of $\mathcal{X}$ are isomorphic.

Proof. A point automorphism has by definition an extension to an automorphism of $\mathcal{X}$, so for (i) it suffices to prove that this extension is unique. Let φ and φ' be automorphisms of $\mathcal{X}$ such that $\varphi|_{\mathcal{P}(\mathcal{X})} = \varphi'|_{\mathcal{P}(\mathcal{X})}$. Let $B \in \mathcal{B}(\mathcal{X})$ be arbitrary. Since φ and φ' are automorphisms, we have

$$(\varphi(p), \varphi(B)) \in \mathcal{I}(\mathcal{X}) \quad \Leftrightarrow \quad (p, B) \in \mathcal{I}(\mathcal{X}) \quad \Leftrightarrow \quad (\varphi'(p), \varphi'(B)) \in \mathcal{I}(\mathcal{X})$$

for all $p \in \mathcal{P}(\mathcal{X})$. Because no two distinct blocks of a simple incidence structure are incident with the same set of points, we must have $\varphi(B) = \varphi'(B)$. Because B was arbitrary, $\varphi = \varphi'$.

To establish (ii) it suffices to check that the map that takes a point automorphism to its unique extension is a group homomorphism. This is straightforward. □

Subsystems of Steiner triple systems

A *subsystem* of order w in an STS(v) is a subset of blocks that induces an STS(w).

In Section 5.1 we shall require the following result on the possible orders of a subsystem that an STS(v) can admit.

Theorem 2.13 **[17, Lemma 6.1]** *If an* STS(v) *contains a subsystem of order w, then either $v \geq 2w + 1$ or $v = w$.*

Proof. The claim is trivial when $v = w$, so suppose $0 < w < v$. Let the points of the subsystem be $\{s_1, \ldots, s_w\}$ and let the other points of the STS(v) be $\{p_1, \ldots, p_{v-w}\}$. Fix the point p_1 and consider the blocks that are incident with p_1. Since no pair of the form $\{s_i, s_j\}$ occurs in these blocks, there exist w blocks of the form $\{p_1, s_i, p_{j_i}\}$. Since there are exactly $v - w - 1$ pairs of the form $\{p_1, p_j\}$, we have $v - w - 1 \geq w$. □

Block graphs of Steiner triple systems

This section contains a number of results that relate the block graph of an $\mathrm{STS}(v)$ to the underlying $\mathrm{STS}(v)$. These results will become relevant in Section 4.4, where we use block graphs for performing isomorph rejection on the $\mathrm{STS}(19)$ generated in the main search. All the results in this section are well known in the sense that they are straightforward consequences of more generic results and techniques due to Bose [7] and Bruck [9]. The applicability of the results of Bose and Bruck to isomorphism testing of graphs is noted already in [64]; see also [74] and the references therein.

It is well known that the block graph of an $\mathrm{STS}(v)$ has the structure of a strongly regular graph. This can be either proven directly or as a consequence of the following theorems of Bose [7].

Theorem 2.14 [7, Theorem 3.1] *The dual of a partial geometry* $\mathrm{pg}(k, r, t)$ *is a partial geometry* $\mathrm{pg}(r, k, t)$.

Theorem 2.15 [7, Theorem 4.1] *The point graph of a partial geometry* $\mathrm{pg}(k, r, t)$ *is a strongly regular graph* $\mathrm{srg}(n, d, \lambda, \mu)$, *where*

$$\begin{aligned} n &= k(1 + (r-1)(k-1)/t), \qquad & d &= r(k-1), \\ \lambda &= (k-2) + (r-1)(t-1), \qquad & \mu &= rt. \end{aligned} \tag{3}$$

A strongly regular graph $\mathrm{srg}(n, d, \lambda, \mu)$ that satisfies (3) for some positive integers k, r, t is a *pseudo-geometric graph* (k, r, t).

Theorem 2.16 [7, Theorem 9.3] *Let* X *be a pseudo-geometric graph* (k, r, t) *and let*

$$2k > r(r-1) + t(r+1)(r^2 - 2r + 2).$$

Then, X *is the point graph of a partial geometry* $\mathrm{pg}(k, r, t)$.

It is straightforward to check that an $\mathrm{STS}(v)$ is equivalent to a partial geometry $\mathrm{pg}(3, r, 3)$, where $r = (v-1)/2$. Thus, using the above theorems, we obtain:

Theorem 2.17 *The block graph of an* $\mathrm{STS}(v)$ *is a strongly regular graph* $\mathrm{srg}(v(v-1)/6, 3(v-3)/2, (v+3)/2, 9)$. *Conversely, a strongly regular graph with these parameters is the block graph of an* $\mathrm{STS}(v)$ *for all admissible* $v > 67$.

The following lemma shows that cliques in the block graph of an $\mathrm{STS}(v)$ have small order unless all the blocks in the clique share a common point. Thus, the cliques of large order in the block graph can be identified with the points of the underlying $\mathrm{STS}(v)$. This observation enables the proofs of Theorems 2.19 and 2.20, which provide the required link between the block graph and the underlying $\mathrm{STS}(v)$ for purposes of isomorph rejection.

Lemma 2.18 *A family of pairwise intersecting blocks in an* $\mathrm{STS}(v)$ *such that the blocks have no common point has size at most* 7. *If equality holds, the blocks form a subsystem of order* 7.

Proof. Let S be a family of pairwise intersecting blocks in an STS(v) with no common point. Let p be a point of the STS(v) incident with at least one of the blocks in S. Suppose p is incident with exactly r blocks in S and denote them by $B_1, \ldots, B_r$. By the structure of an STS, the blocks $B_1, \ldots, B_r$ with p removed are pairwise disjoint. Since S contains a block which intersects $B_1, \ldots, B_r$ but is not incident with p, we must have $r \leq 3$. The number of blocks that intersect any fixed block $B \in S$ is therefore at most $3(3-1) = 6$, so $|S| \leq 7$. For $|S| = 7$ the blocks that intersect any fixed block $B \in S$ partition into three classes of two blocks each according to the point of intersection. Since blocks from distinct classes must intersect pairwise, a short case-by-case analysis shows that the only possibility is that the blocks in S induce a sub-STS(7). □

Theorem 2.19 *Let $\mathcal{X}$ be an* STS(v)*, $v \geq 19$, and let X be the block graph of $\mathcal{X}$. Then,* Aut(X) *is equal to the block automorphism group of $\mathcal{X}$.*

Proof. We show first that the block automorphism group of $\mathcal{X}$ is a subgroup of Aut(X). Let $\varphi \in \mathrm{Aut}(\mathcal{X})$ and denote the restriction of φ to $\mathcal{B}(\mathcal{X})$ by ψ. Let $B_i, B_j \in V(X)$. By definition of a block graph, $\{B_i, B_j\} \in E(X)$ if and only if there exists a $p \in \mathcal{P}(\mathcal{X})$ such that $(p, B_i) \in \mathcal{I}(\mathcal{X})$ and $(p, B_j) \in \mathcal{I}(\mathcal{X})$. Since φ is an automorphism, the latter holds if and only if there exists a $q \in \mathcal{P}(\mathcal{X})$ such that $(q, \varphi(B_i)) \in \mathcal{I}(\mathcal{X})$ and $(q, \varphi(B_j)) \in \mathcal{I}(\mathcal{X})$, that is, if and only if $\{\psi(B_i), \psi(B_j)\} \in E(X)$. Thus, $\psi \in \mathrm{Aut}(X)$.

We must still show that an arbitrary $\psi \in \mathrm{Aut}(X)$ is a block automorphism of $\mathcal{X}$. Observe that with every edge $\{B_i, B_j\} \in E(X)$ there is an associated $p \in \mathcal{P}(X)$, namely the point with which both B_i and B_j are incident. Let Y be a maximal clique in X. Either all edges of Y have the same point p associated with them (in which case Y has order $r = (v-1)/2 \geq 9$) or at least two distinct points are associated with edges of Y. In the latter case, Y has order at most 7 by Lemma 2.18. Thus, the maximum cliques in X are in a one-to-one correspondence with the points of $\mathcal{X}$. Denote the maximum clique of X associated with the point p by Y_p. Since an automorphism $\psi \in \mathrm{Aut}(X)$ must permute the maximum cliques of X, we obtain a permutation σ of $\mathcal{P}(\mathcal{X})$ by requiring $\psi(Y_p) = Y_{\sigma(p)}$ for all $p \in \mathcal{P}(\mathcal{X})$.

Define a map φ of $\mathcal{P}(\mathcal{X}) \cup \mathcal{B}(\mathcal{X})$ into itself by setting $\varphi(p) = \sigma(p)$ for all $p \in \mathcal{P}(\mathcal{X})$ and $\varphi(B) = \psi(B)$ for all $B \in \mathcal{B}(\mathcal{X})$. Clearly, $\varphi \in \mathrm{Sym}(\mathcal{P}(\mathcal{X})) + \mathrm{Sym}(\mathcal{B}(\mathcal{X}))$. Now,

$$(p, B) \in \mathcal{I}(\mathcal{X}) \;\Leftrightarrow\; B \in V(Y_p) \;\Leftrightarrow\; \psi(B) \in V(Y_{\sigma(p)}) \;\Leftrightarrow\; (\varphi(p), \varphi(B)) \in \mathcal{I}(\mathcal{X})$$

for all $p \in \mathcal{P}(\mathcal{X})$ and $B \in \mathcal{B}(\mathcal{X})$, so $\varphi \in \mathrm{Aut}(\mathcal{X})$. This shows that ψ is a block automorphism of $\mathcal{X}$. □

Theorem 2.20 *Let $\mathcal{X}$ and $\mathcal{Y}$ be* STS(v)*, $v \geq 19$. Then, $\mathcal{X}$ and $\mathcal{Y}$ are isomorphic if and only if their block graphs are isomorphic. Moreover, $\mathcal{X}$ is reconstructible up to isomorphism from its block graph.*

Proof. Isomorphic STS(v) have isomorphic block graphs because an isomorphism of the STS(v) gives an isomorphism of the block graphs by restriction to the block set. Let ψ be an isomorphism from the block graph of $\mathcal{X}$ onto

the block graph of $\mathcal{Y}$. For every $p \in \mathcal{P}(\mathcal{X})$, denote by Y_p the maximum clique that corresponds to p in the block graph of $\mathcal{X}$. Similarly, for every $p' \in \mathcal{P}(\mathcal{Y})$, denote by $Y'_{p'}$ the corresponding maximum clique. Since ψ must map maximum cliques onto maximum cliques, we obtain a bijection $\varphi : \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{Y})$ by requiring $Y'_{\varphi(p)} = \psi(Y_p)$ for all $p \in \mathcal{P}(\mathcal{X})$. Now, φ can be extended to an isomorphism from $\mathcal{X}$ onto $\mathcal{Y}$ by setting $\varphi(B) = \psi(B)$ for all $B \in \mathcal{B}(\mathcal{X})$. The following chain of equivalences shows that φ is an isomorphism:

$$(p, B) \in \mathcal{I}(\mathcal{X}) \Leftrightarrow B \in V(Y_p) \Leftrightarrow \psi(B) \in V(Y'_{\varphi(p)}) \Leftrightarrow (\varphi(p), \varphi(B)) \in \mathcal{I}(\mathcal{Y}).$$

Reconstructibility is clear: label the maximum cliques in X with v points, and set each block to be incident with the labels of the three maximum cliques to which the block belongs. □

Theorem 2.19 is tight in the sense that the claim does not hold for $v = 15$ as the following example demonstrates.

Example 2.21 Let the point set of an STS(15) consist of the 15 edges in the complete graph K_6. Take as blocks all the sets of three edges that form either a one-factor or a triangle [83, Problem 19A (i)].

The full automorphism group of the resulting STS(15) has order 20160; the full automorphism group of the corresponding block graph has order 40320. (Both groups are straightforward to construct using, for example, *nauty* [61] and techniques from Section 3.2.)

For $v \leq 15$ it can be checked that Theorem 2.20 holds in the restricted sense that two STS(v) are isomorphic if and only if their block graphs are isomorphic. Reconstructing the STS from its block graph using maximum cliques is not possible for small v; for example, the block graph of the STS(7) is the complete graph K_7.

Pasch configurations in Steiner triple systems

A *configuration* in an STS(v) is an incidence structure induced by a set of blocks. One such example is the Pasch configuration from Example 2.1.

An STS(v) is *anti-Pasch* if it contains no Pasch configurations.

Theorem 2.22 [38] *Anti-Pasch* STS(v) *exist for all* $v \equiv 1, 3 \pmod 6$ *except when* $v = 7$ *or* $v = 13$.

Pasch configurations will play an important role in the isomorph rejection scheme described in Section 4.4. This is because Pasch configurations are the only configurations with four pairwise intersecting blocks whose number may vary in an STS(v) in the following sense:

Theorem 2.23 *Each block of an* STS(v) *occurs in exactly* $(v-3)(v^2-12v+99)/16$ *configurations of four pairwise intersecting blocks that are not Pasch configurations.*

Proof. Consider any set of four pairwise intersecting blocks in an STS(v). A short case-by-case analysis shows that, unless the blocks form a Pasch configuration, there exists a unique point, x, that is incident with at least three of

the blocks. Fix any block B of an STS(v). We count the sets of four blocks of the above type in which B occurs by splitting the count into subcases as follows.

Case A. The point x is incident with three blocks, including B. First, there are three possibilities to choose an x incident with B. Second, there are $r-1$ choices for a block B_1 that contains x. Third, there are four choices for a block B_2 that is not incident with x and that intersects both B and B_1. The block B_3 that is incident with x and that intersects B_2 is unique. Since the choices for B_1 and B_3 can be interchanged, the total number of sets of four blocks that contain B in this subcase is $6(r-1)$.

Case B. The point x is incident with three blocks, not including B. There are $v-3$ choices for x. The three other blocks are uniquely determined as the blocks incident with both x and one of the three points incident with B.

Case C. The point x is incident with all four blocks. There are $3\binom{r-1}{3}$ sets of four blocks that contain B in which x is incident with all four blocks. Namely, there are three ways to choose x and $\binom{r-1}{3}$ ways to choose the other 3 blocks after x has been fixed.

Since evidently the subcases do not overlap, we have that B occurs in exactly

$$6(r-1)+(v-3)+3\binom{r-1}{3}=\frac{(v-3)(v^2-12v+99)}{16}$$

sets of four blocks that intersect pairwise but do not form a Pasch configuration. □

We remark that the above theorem can be obtained as a simple corollary of the results in [37]. Configurations in STS(v) are surveyed in [17, Ch. 13].

2.3 Isomorph-free exhaustive generation

A substantial literature exists on the generation of combinatorial objects using computer search. Computational methods in design theory are surveyed in [33, 73]. Algorithmic aspects of combinatorial designs are surveyed in [18].

The exhaustive construction of certain combinatorial objects such as STS(v), t-(v,k,λ) designs, and error-correcting codes seems to be a hard problem in the sense that no efficient exhaustive construction algorithms are known in the general case, although numerous direct and recursive constructions exist; see [17] for constructions of STS(v). For example, even the existence of a 2-$(22,8,4)$ design has not yet been settled, although the problem is in principle solvable by exhaustive search. A theoretical discussion of notions of efficiency and algorithms for efficient exhaustive generation of various combinatorial structures, such as graphs of fixed order, appears in [34, 35].

While more efficient exhaustive generation methods are lacking, the principal tool in algorithmic exhaustive generation is usually backtrack search [36] in one form or another. In this report, we use a clever backtrack search algorithm due to Knuth [52] in solving the exact cover problems associated with the construction of STS(19).

To make an exhaustive backtrack search practical, the consideration of isomorphic partial solutions in the search must be avoided as much as possible. This is because the number of isomorphic partial solutions is typically exponential in the size of the objects being generated. For example, asymptotically almost all isomorphism classes of STS(v) contain $v!$ distinct STS(v) [2].

The procedure of performing elimination of isomorphic solutions was named *isomorph rejection* by Swift [81]. In its simplest form, isomorph rejection involves testing the generated complete objects for isomorphism. This alone is often a nontrivial issue when there are too many nonisomorphic objects to fit in the main memory of the computer used. To make a search feasible in practice, isomorph rejection must usually be performed also (i) prior to the search by a careful choice of starting points for the search; and (ii) during the search by explicit isomorphism computations.

A number of generic techniques have been developed for performing isomorph rejection in exhaustive generation algorithms. These include [28, 49, 63, 69]. Perhaps the most widely applicable of the techniques is the one introduced by McKay [63]; this is also the technique behind the isomorph rejection scheme used by the STS(19) classification algorithm in Chapter 4. For reasons of space we will not give a detailed description of the model in [63], which uses a very abstract setting to cover as many application domains as possible. A readable informal exposition of the techniques for isomorph rejection, including McKay's technique, is given in [8].

3 AUXILIARY ALGORITHMS

This chapter describes the auxiliary problems and their solution algorithms that are used in the STS(19) classification algorithms described in Chapters 4 and 5.

The organization of this chapter is as follows. Section 3.1 gives a description of Algorithm DLX [52], which we use to solve instances of the exact cover problem associated with the construction of STS(19). Section 3.2 briefly discusses the isomorphism problem for graphs and STS(v). We also give a black-box description of the graph canonical labelling package *nauty* [60, 61], which we use to compute automorphism groups and canonical labelling for block graphs and incidence structures. Section 3.3 gives pointers to permutation group algorithms, which are required in preprocessing stages of the classification algorithms.

3.1 Exact cover

The *exact cover problem* is to determine, given a collection $F_1, \ldots, F_m$ of subsets of a finite set E as input, whether there exists a subcollection that partitions E. In the search version of the problem we are also asked to produce one such subcollection (or all of them) whenever these exist.

The exact cover problem is NP-complete [31], and hence generally considered intractable in the sense that no algorithm whose running time is bounded by a polynomial in the length of the input instance (e.g., a poly-

nomial in $m \cdot |E|$) is likely to exist.

Algorithm DLX

We employ Algorithm DLX [52] in solving the exact cover problem. Algorithm DLX is essentially the straightforward backtrack search that attempts to cover the elements of E one at a time by using the available subsets $F_1, \ldots, F_m$. The tricks that make the algorithm run fast are (i) a heuristic that minimizes the branching factor at each level of the search; and (ii) clever use of doubly linked circular lists which enables fast backtracking.

In describing the algorithm it is convenient to view an exact cover problem instance $E, F_1, \ldots, F_m$ as a $m \times |E|$ matrix A with entries in $\{0, 1\}$. Each row of the matrix corresponds to one subset F_i, and each column of the matrix corresponds to an element of E. The entry at row i, column j of the matrix is 1 if $j \in F_i$; otherwise the entry is 0. A solution to the problem instance is now a collection of rows of A such that every column is *covered* exactly once, that is, for every column there exists a unique row in the collection with a 1 in that column.

Example 3.1 Let $E = \{e_1, \ldots, e_7\}$ and

$$F_1 = \{e_3, e_5, e_6\}, \quad F_2 = \{e_1, e_4, e_7\}, \quad F_3 = \{e_2, e_3, e_6\},$$
$$F_4 = \{e_1, e_4\}, \quad F_5 = \{e_2, e_7\}, \quad F_6 = \{e_4, e_5, e_7\}$$

be an instance of the exact cover problem. The matrix A that corresponds to this instance is

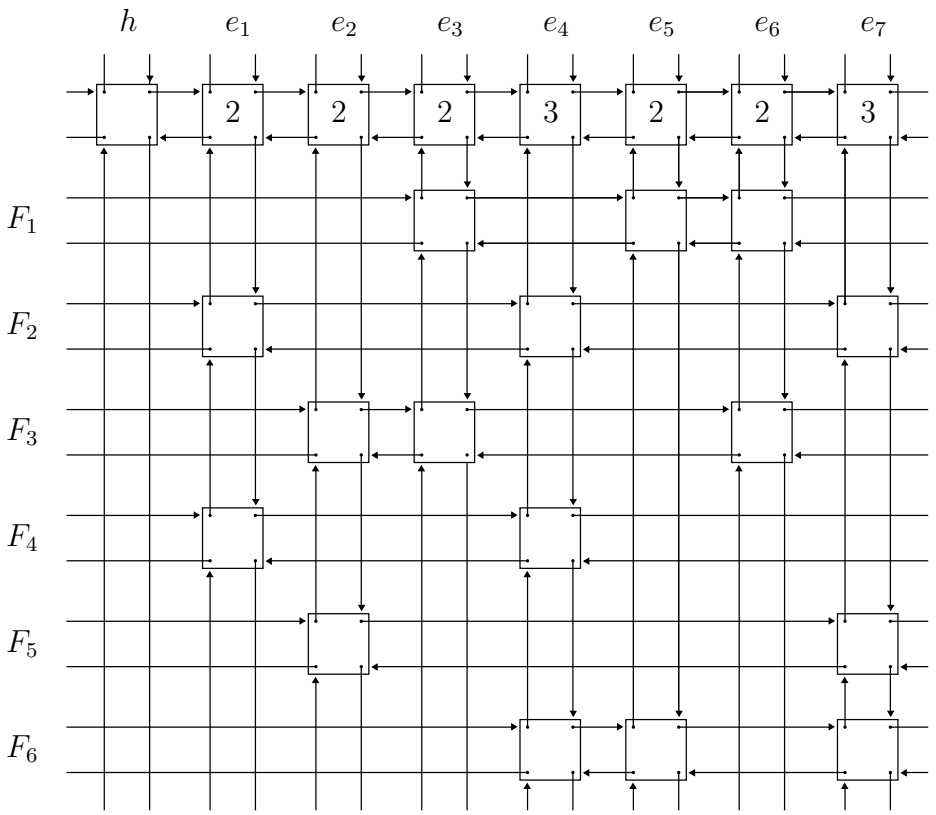
$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

(An exact cover is given by the subsets F_1, F_4, F_5.)

The global data structure manipulated by Algorithm DLX consists of multiple circular doubly linked lists. Each 1 in the matrix A corresponds to a list entry x with five fields $L[x]$, $R[x]$, $U[x]$, $D[x]$, $C[x]$. Rows of the matrix are doubly linked as circular lists via the L and R fields ("left" and "right"); columns are doubly linked as circular lists via the U and D fields ("up" and "down"). Each column list includes a special entry called the *column header*. The C field of each list entry points to the column header of the column in which the entry lies. A column header contains an additional field, $S[x]$ ("size"), which is used to keep track of the number of rows linked to the column list.

The uncovered columns are linked to a circular list via the L and R of their column headers. The column header list also contains a special entry, the *root*, h, which is used to access the column header list.

Example 3.2 The global data structure for Algorithm DLX initialized with the instance of Example 3.1 is depicted below.



The arrows represent the R, U, L, D links in the list entries depicted by boxes. The column headers appear on the topmost row.

Algorithm 1 gives a pseudocode description of Algorithm DLX. Algorithm 2 contains pseudocode for the operations of covering and uncovering a column.

The covering and uncovering operations contain the two tricks in the algorithm. Consider the first line of procedure *cover* in Algorithm 2. The operations

$$L[R[c]] \leftarrow L[c], \qquad R[L[c]] \leftarrow R[c]$$

clearly remove the column header c from the uncovered column list. The elegant observation for backtracking is that the operations

$$L[R[c]] \leftarrow c, \qquad R[L[c]] \leftarrow c$$

suffice to insert c back to the list. Thus, it suffices only to keep track of the elements deleted from a list to enable their insertion back to the list. This eliminates essentially all of the bookkeeping usually required by backtracking since all information required to "rewind" the global data structure back to the earlier state is present on the L, R, U, D fields of the deleted list entries.

The second trick is the minimum branching degree heuristic. At each level of the search, a column with the minimum number of linked rows is selected for covering. Compared with a similar algorithm with no heuristic (that is, the next uncovered column in some fixed order is always selected), the minimum branching degree heuristic typically reduces the total number of search tree nodes by a factor that is exponential in the number of levels in the tree [52].

Algorithm 1 Algorithm DLX [52].

procedure *search*$(k$: integer$)$
  If $R[h] = h$, report the current solution $Q_1, \ldots, Q_{k-1}$ and return.
  Otherwise choose an uncovered column c with $S[c]$ minimum.
  Cover column c.
  For each $r \leftarrow D[c], D[D[c]], \ldots,$ while $r \neq c$,
    Set $Q_k \leftarrow r$;
    For each $j \leftarrow R[r], R[R[r]], \ldots,$ while $j \neq r$,
      Cover column $C[j]$;
    search$(k+1)$;
    For each $j \leftarrow L[r], L[L[r]], \ldots,$ while $j \neq r$,
      Uncover column $C[j]$.
  Uncover column c and return.

initial invocation
  search(1).

Algorithm 2 Covering and uncovering a column [52].

procedure *cover*$(c$: column header$)$
  Set $L[R[c]] \leftarrow L[c]$ and $R[L[c]] \leftarrow R[c]$.
  For each $i \leftarrow D[c], D[D[c]], \ldots,$ while $i \neq c$,
    For each $j \leftarrow R[i], R[R[i]], \ldots,$ while $j \neq i$,
      Set $U[D[j]] \leftarrow U[j]$ and $D[U[j]] \leftarrow D[j]$;
      Set $S[C[j]] \leftarrow S[C[j]] - 1$.

procedure *uncover*$(c$: column header$)$
  For each $i \leftarrow U[c], U[U[c]], \ldots,$ while $i \neq c$,
    For each $j \leftarrow L[i], L[L[i]], \ldots,$ while $j \neq i$,
      Set $S[C[j]] \leftarrow S[C[j]] + 1$;
      Set $U[D[j]] \leftarrow j$ and $D[U[j]] \leftarrow j$.
  Set $L[R[c]] \leftarrow c$ and $R[L[c]] \leftarrow c$.

3.2 Graph isomorphism

The *graph isomorphism problem* is to determine, given two graphs X, Y as input, whether they are isomorphic. In the search version of the problem we are also asked to produce an isomorphism from X to Y whenever the graphs are isomorphic.

The graph isomorphism problem is of fundamental importance in combinatorial computation since the isomorphism problem of most combinatorial structures can be reduced to testing isomorphism of two graphs derived from these structures. Indeed, it often is the case that the isomorphism problem for a class of combinatorial objects is *equivalent* to the graph isomorphism problem in the sense that polynomial time computable reductions exist in both directions [6].

The computational complexity of the graph isomorphism problem has attracted a vast amount of research (see for example [41, 53, 70] and the references therein). It is currently believed that the graph isomorphism problem is neither NP-complete nor admits a polynomial time algorithm, although polynomial time isomorphism algorithms exist for many families of graphs.

The computational complexity of the $\mathrm{STS}(v)$ isomorphism problem in relation to the graph isomorphism problem is unresolved [17]. It is clear that the isomorphism problem for $\mathrm{STS}(v)$ is reducible to testing the incidence graphs of the $\mathrm{STS}(v)$ for isomorphism, but it is unknown whether graph isomorphism can be reduced to testing two $\mathrm{STS}(v)$ for isomorphism. It is known that testing isomorphism of 2-$(v, 3, \lambda)$ designs is polynomial time equivalent to graph isomorphism [19]. On the other hand, an isomorphism algorithm operating in subexponential $O(v^{\log v + f(k,\lambda)})$ time exists for $\mathrm{STS}(v)$ [64] and 2-(v, k, λ) designs with k, λ bounded [3]. For $\mathrm{STS}(v)$ with no subsystem, an $O(v^5)$ algorithm exists [77].

A problem related to the graph isomorphism problem is the *graph automorphism (generator) problem*, which, given a graph X as input, asks for a set of generators for the full automorphism group $\mathrm{Aut}(X)$.

The graph isomorphism and automorphism problems are equivalent in hardness in the sense that a polynomial time algorithm for one exists if and only if the other has a polynomial time algorithm [58].

Canonical representatives and canonical labelling

Most general-purpose graph isomorphism algorithms are based on the idea of computing a *code* [70] (or *certificate* [56, 65]) for each input graph such that two graphs are isomorphic if and only if they have the same code. In practice, this certificate is a graph that is isomorphic to the input graph [54, 61]. Such algorithms that determine the *canonical representative* of a family of isomorphic graphs are useful also for other purposes besides testing isomorphism, as we shall see later in Section 4.4.

An algorithm for computing the canonical representative of a graph takes as input a graph X and outputs a graph C, isomorphic to X, such that the same graph C is output no matter which graph from the isomorphism equivalence class of C is input to the algorithm; the graph C is the *canonical representative* of its isomorphism equivalence class. An isomorphism of X onto C is a *canonical labelling* of X.

In practice, we shall be more interested in obtaining a canonical labelling for the input graph X than the actual canonical representative. For future convenience we shall adopt a more fixed setting, which will be given in the following definitions.

We work over a fixed finite nonempty vertex set V. A permutation $\varphi \in \mathrm{Sym}(V)$ acts on a graph X on V in the obvious way, that is, $\varphi(X)$ is the graph with $V(\varphi(X)) = V$ and edge set defined by

$$\{v, w\} \in E(X) \qquad \Leftrightarrow \qquad \{\varphi(v), \varphi(w)\} \in E(\varphi(X)).$$

Let $\mathcal{G}(V)$ be a set of graphs on V such that for all $\varphi \in \mathrm{Sym}(V)$ and $X \in \mathcal{G}(V)$, we have $\varphi(X) \in \mathcal{G}(V)$. In other words, $\mathcal{G}(V)$ is a set of graphs closed under permutation of the vertex set V.

We say that a function $\kappa : \mathcal{G}(V) \to \mathrm{Sym}(V)$ is a *canonical labelling map* for $\mathcal{G}(V)$ if, for all $X \in \mathcal{G}(V)$ and $\varphi \in \mathrm{Sym}(V)$, we have

$$\kappa_X(X) = \kappa_{\varphi(X)}(\varphi(X)). \tag{4}$$

In other words, κ assigns to every graph $X \in \mathcal{G}(V)$ a canonical labelling $\kappa_X \in \mathrm{Sym}(V)$, which takes X to the canonical representative $\kappa_X(X)$.

Canonical labelling of graphs with *nauty*

The *nauty* package by Brendan McKay [61] computes for an arbitrary graph a canonical labelling and generators for the full automorphism group. The mathematical basis of the algorithm is described in [60].

For our purposes it is almost sufficient to regard *nauty* as a black box that for a given input graph X simply outputs a canonical labelling κ_X and generators for $\mathrm{Aut}(X)$. To get good performance from *nauty* in the canonical labelling of block graphs of $\mathrm{STS}(v)$, and to enable the derivation of a canonical labelling map for incidence structures, we will require a slightly expanded view.

Some further preliminary definitions are necessary. We continue to work over a fixed finite set of vertices V and a set of graphs $\mathcal{G}(V)$, closed under permutations of V. Fix an arbitrary total order on V and label the elements of V as $v_1, v_2, \ldots, v_{|V|}$ so that $v_i < v_j$ holds if and only if $i < j$. For each ordered partition $\pi = (V_1, \ldots, V_m) \in \Pi(V)$, let $c(\pi) = (W_1, \ldots, W_m)$, where

$$\begin{aligned} W_1 &= \{v_1, \ldots, v_{|V_1|}\}, \\ W_2 &= \{v_{|V_1|+1}, \ldots, v_{|V_1|+|V_2|}\}, \\ &\vdots \\ W_m &= \{v_{|V|-|V_m|+1}, \ldots, v_{|V|}\}. \end{aligned}$$

A permutation $\varphi \in \mathrm{Sym}(V)$ acts on an ordered partition $\pi = (V_1, \ldots, V_m) \in \Pi(V)$ so that

$$\varphi(\pi) = (\varphi(V_1), \ldots, \varphi(V_m)).$$

In particular, for all $\varphi \in \mathrm{Sym}(V)$ and $\pi \in \Pi(V)$, $c(\pi) = c(\varphi(\pi))$.

The canonical labelling map computed by *nauty* can now be described as follows. A function $\kappa : \mathcal{G}(V) \times \Pi(V) \to \mathrm{Sym}(V)$ is a canonical labelling map with *initial partition constraint* if for all $X \in \mathcal{G}(V)$, $\varphi \in \mathrm{Sym}(V)$, and $\pi \in \Pi(V)$, we have

$$\kappa_{X,\pi}(X) = \kappa_{\varphi(X),\varphi(\pi)}(\varphi(X)) \qquad \text{and} \qquad \kappa_{X,\pi}(\pi) = c(\pi). \tag{5}$$

The group of automorphisms computed by *nauty* is analogously constrained by the initial partition $\pi \in \Pi(V)$. For a graph $X \in \mathcal{G}(V)$ and an initial partition $\pi \in \Pi(V)$, the group of automorphisms computed by *nauty* is

$$\mathrm{Aut}_\pi(X) = \{\alpha \in \mathrm{Aut}(X) \ : \alpha(\pi) = \pi\}. \tag{6}$$

Clearly, (5) reduces to (4), and (6) reduces to the full automorphism group of X if we take π to be the unit partition. The following theorem shows that a similar situation arises if we produce the initial partition using a vertex invariant. A function $\iota : \mathcal{G}(V) \to \Pi(V)$ is a *vertex invariant* for $\mathcal{G}(V)$ if, for all $X \in \mathcal{G}(V)$ and $\varphi \in \mathrm{Sym}(V)$,

$$\iota(\varphi(X)) = \varphi(\iota(X)). \tag{7}$$

Theorem 3.3 *Let $\kappa : \mathcal{G}(V) \to \mathrm{Sym}(V)$ be a canonical labelling map with initial partition constraint and let $\iota : \mathcal{G}(V) \to \Pi(V)$ be a vertex invariant for $\mathcal{G}(V)$. Then, the map $\hat{\kappa} : X \mapsto \kappa_{X,\iota(X)}$ is a canonical labelling map for $\mathcal{G}(V)$. Moreover, $\mathrm{Aut}_{\iota(X)}(X) = \mathrm{Aut}(X)$ for all $X \in \mathcal{G}(V)$.*

Proof. Let $X \in \mathcal{G}(V)$ and $\varphi \in \mathrm{Sym}(V)$. By (5) and (7), we have

$$\begin{aligned}\hat{\kappa}_X(X) &= \kappa_{X,\iota(X)}(X) = \kappa_{\varphi(X),\varphi(\iota(X))}(\varphi(X)) \\ &= \kappa_{\varphi(X),\iota(\varphi(X))}(\varphi(X)) = \hat{\kappa}_{\varphi(X)}(\varphi(X)).\end{aligned}$$

Thus, (4) holds for $\hat{\kappa}$. Since for any $\alpha \in \mathrm{Aut}(X)$ we have $\alpha(X) = X$, it must be that $\iota(X) = \iota(\alpha(X)) = \alpha(\iota(X))$. Thus, $\mathrm{Aut}_{\iota(X)}(X) = \mathrm{Aut}(X)$. □

Canonical labelling of incidence structures

In addition to graphs, we shall also perform isomorph rejection on incidence structures. For this purpose we require the analogous concepts of a canonical representative and canonical labelling for incidence structures.

We work over a fixed nonempty set of points $\mathcal{P}$ and a fixed nonempty set of blocks $\mathcal{B}$ (disjoint from $\mathcal{P}$). Let $\mathcal{S}(\mathcal{P},\mathcal{B})$ be a set of incidence structures on $\mathcal{P},\mathcal{B}$. We assume that $\mathcal{S}(\mathcal{P},\mathcal{B})$ is closed under permutation of the points $\mathcal{P}$ and the blocks $\mathcal{B}$, where a permutation $\varphi \in \mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$ acts on an incidence structure $\mathcal{X} \in \mathcal{S}(\mathcal{P},\mathcal{B})$ so that $\varphi(\mathcal{X}) \in \mathcal{S}(\mathcal{P},\mathcal{B})$ is the incidence structure defined by

$$(\varphi(p),\varphi(B)) \in \mathcal{I}(\varphi(\mathcal{X})) \qquad \Leftrightarrow \qquad (p,B) \in \mathcal{I}(\mathcal{X}) \tag{8}$$

for all $p \in \mathcal{P}$ and $B \in \mathcal{B}$.

A function $\jmath : \mathcal{S}(\mathcal{P},\mathcal{B}) \to \mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$ is a *canonical labelling map* for $\mathcal{S}(\mathcal{P},\mathcal{B})$ if, for all $\mathcal{X} \in \mathcal{S}(\mathcal{P},\mathcal{B})$ and $\varphi \in \mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$ we have

$$\jmath_{\mathcal{X}}(\mathcal{X}) = \jmath_{\varphi(\mathcal{X})}(\varphi(\mathcal{X})). \tag{9}$$

We now construct a canonical labelling map for $\mathcal{S}(\mathcal{P},\mathcal{B})$ from a canonical labelling map for graphs. This is a well-known construction, see, for example, [33].

Let κ be a canonical labelling map with initial partition constraint for the set of all graphs on $\mathcal{P} \cup \mathcal{B}$. We assume that the set $\mathcal{P} \cup \mathcal{B}$ is ordered so that $p < B$ holds for all $p \in \mathcal{P}$ and $B \in \mathcal{B}$. Let $\mathcal{X} \in \mathcal{S}(\mathcal{P},\mathcal{B})$ and let $I_{\mathcal{X}}$ be the incidence graph of $\mathcal{X}$. The incidence graph $I_{\mathcal{X}}$ and κ can be used to canonically label an incidence structure $\mathcal{X}$ as follows:

Theorem 3.4 *The map $\jmath : \mathcal{X} \mapsto \kappa_{I_\mathcal{X},(\mathcal{P},\mathcal{B})}$ is a canonical labelling map for $\mathcal{S}(\mathcal{P},\mathcal{B})$. Moreover,* $\mathrm{Aut}(\mathcal{X}) = \mathrm{Aut}_{(\mathcal{P},\mathcal{B})}(I_\mathcal{X})$.

Proof. Let $\mathcal{X} \in \mathcal{S}(\mathcal{P},\mathcal{B})$ and let $\varphi \in \mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$. Observe that $\kappa_{I_\mathcal{X},(\mathcal{P},\mathcal{B})} \in \mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$ by (5) and the assumed ordering of $\mathcal{P} \cup \mathcal{B}$. Now, for all $p \in \mathcal{P}$ and $B \in \mathcal{B}$,

$$\begin{aligned}
(p,B) \in \mathcal{I}(\jmath_\mathcal{X}(\mathcal{X})) \quad &\Leftrightarrow \quad (\jmath_\mathcal{X}^{-1}(p), \jmath_\mathcal{X}^{-1}(B)) \in \mathcal{I}(\mathcal{X}) \\
&\Leftrightarrow \quad \{\kappa^{-1}_{I_\mathcal{X},(\mathcal{P},\mathcal{B})}(p), \kappa^{-1}_{I_\mathcal{X},(\mathcal{P},\mathcal{B})}(B)\} \in E(I_\mathcal{X}) \\
&\Leftrightarrow \quad \{\kappa^{-1}_{\varphi(I_\mathcal{X}),(\mathcal{P},\mathcal{B})}(p), \kappa^{-1}_{\varphi(I_\mathcal{X}),(\mathcal{P},\mathcal{B})}(B)\} \in E(\varphi(I_\mathcal{X})) \\
&\Leftrightarrow \quad (\jmath^{-1}_{\varphi(\mathcal{X})}(p), \jmath^{-1}_{\varphi(\mathcal{X})}(B)) \in \mathcal{I}(\varphi(\mathcal{X})) \\
&\Leftrightarrow \quad (p,B) \in \mathcal{I}(\jmath_{\varphi(\mathcal{X})}(\varphi(\mathcal{X}))).
\end{aligned}$$

The first and last equivalence follow from (8); the second and fourth equivalence are a consequence of (2); the third equivalence is implied by (5) and $\varphi(\mathcal{P}) = \mathcal{P}$, $\varphi(\mathcal{B}) = \mathcal{B}$.

A permutation $\alpha \in \mathrm{Sym}(\mathcal{P} \cup \mathcal{B})$ is an automorphism of $\mathcal{X}$ if and only if $\alpha(\mathcal{P}) = \mathcal{P}$; $\alpha(\mathcal{B}) = \mathcal{B}$; and for all $p \in \mathcal{P}$ and $B \in \mathcal{B}$,

$$(p,B) \in \mathcal{I}(\mathcal{X}) \qquad \Leftrightarrow \qquad (\alpha(p), \alpha(B)) \in \mathcal{I}(\mathcal{X}).$$

By (2) an equivalent definition is to require $\alpha \in \mathrm{Aut}(I_\mathcal{X})$, $\alpha(\mathcal{P}) = \mathcal{P}$, and $\alpha(\mathcal{B}) = \mathcal{B}$. Thus, $\mathrm{Aut}(\mathcal{X}) = \mathrm{Aut}_{(\mathcal{P},\mathcal{B})}(I_\mathcal{X})$. □

3.3 Algorithms for permutation groups

Several algorithms in this report require computation with permutation groups. For example, the isomorph rejection strategy in Section 4.4 requires the construction of the point automorphism group of an incidence structure from a set of generators output by *nauty*. Also, the algorithm in Section 5.2 requires the construction of orbits of a group.

For reasons of space we omit the detailed description of these auxiliary algorithms from this report because they are neither performance-critical nor central to the main subject matter.

A comprehensive introduction to computation with permutation groups, which is more than adequate for the applications required in this report, is given in [11]. More advanced material appears in [29, 30, 44]. Computational complexity aspects of permutation groups are studied in [41, 57].

4 CONSTRUCTING THE STS(19)

This chapter describes the classification algorithm for the STS(19) and the organization of the computer search which resulted in the classification. The results of the classification appear in Chapter 6.

We divide the construction process for the STS(19) into two stages. In the first stage, we classify the *seeds* for the main search. This preliminary stage is described in Section 4.3. In the second stage, we compute for each seed the extensions of the seed into an STS(19) with the help of Algorithm DLX. This

stage of the classification is executed in parallel on a network of workstations. Whenever an STS(19) is discovered, we perform isomorph rejection on it using *nauty*. The isomorph rejection phase is described in Section 4.4. If the isomorph rejection step accepts the STS(19) as the representative of its isomorphism equivalence class, then we record some of its properties such as the full automorphism group order and the number of Pasch configurations. The organization of the search is briefly discussed in Section 4.5.

4.1 Conventions and assumptions

Before we describe the structure of the algorithm in more detail, let us fix some conventions. Throughout this chapter we will regard the STS(19) as incidence structures constructed using a fixed point set $\mathcal{P} = \{p_1, p_2, \ldots, p_{19}\}$ and a fixed block set $\mathcal{B} = \{B_1, B_2, \ldots, B_{57}\}$. (Recall from Section 2.2 that an STS(19) has $b = 57$ blocks and that every point is incident with exactly $r = 9$ blocks.) We allow the point and block sets of an incidence structure to be proper subsets of $\mathcal{P}$ and $\mathcal{B}$.

We assume that $\mathcal{P}$ is totally ordered so that $p_i < p_j$ if and only if $i < j$. This order on $\mathcal{P}$ induces a lexicographic order on the set of all nonempty subsets of $\mathcal{P}$ and, recursively, on the set of all subsets of nonempty subsets of $\mathcal{P}$.

We let a permutation $\varphi \in \mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$ act on an incidence structure $\mathcal{X}$ with $\mathcal{P}(\mathcal{X}) \subseteq \mathcal{P}$ and $\mathcal{B}(\mathcal{X}) \subseteq \mathcal{B}$ so that $\mathcal{P}(\varphi(\mathcal{X})) = \varphi(\mathcal{P}(\mathcal{X}))$, $\mathcal{B}(\varphi(\mathcal{X})) = \varphi(\mathcal{B}(\mathcal{X}))$, and

$$(p, B) \in \mathcal{I}(\mathcal{X}) \qquad \Leftrightarrow \qquad (\varphi(p), \varphi(B)) \in \mathcal{I}(\varphi(\mathcal{X}))$$

for all $p \in \mathcal{P}$ and $B \in \mathcal{B}$.

The orbits of $\mathrm{Sym}(\mathcal{P}) + \mathrm{Sym}(\mathcal{B})$ on the set of incidence structures over $\mathcal{P}, \mathcal{B}$ (and their subsets) clearly correspond to the isomorphism equivalence classes of such incidence structures.

4.2 Seeds and the top-level algorithm

For lack of a better name, a *seed* is an incidence structure $\mathcal{S}$ with $\mathcal{P}(\mathcal{S}) = \mathcal{P}$ and $\mathcal{B}(\mathcal{S}) \subset \mathcal{B}$, $|\mathcal{B}(\mathcal{S})| = 25$, that satisfies the following three properties:

(i) Every block is incident with exactly three points.

(ii) Every pair of distinct points is incident with at most one block.

(iii) There exists a block that is incident with exactly one point from every other block.

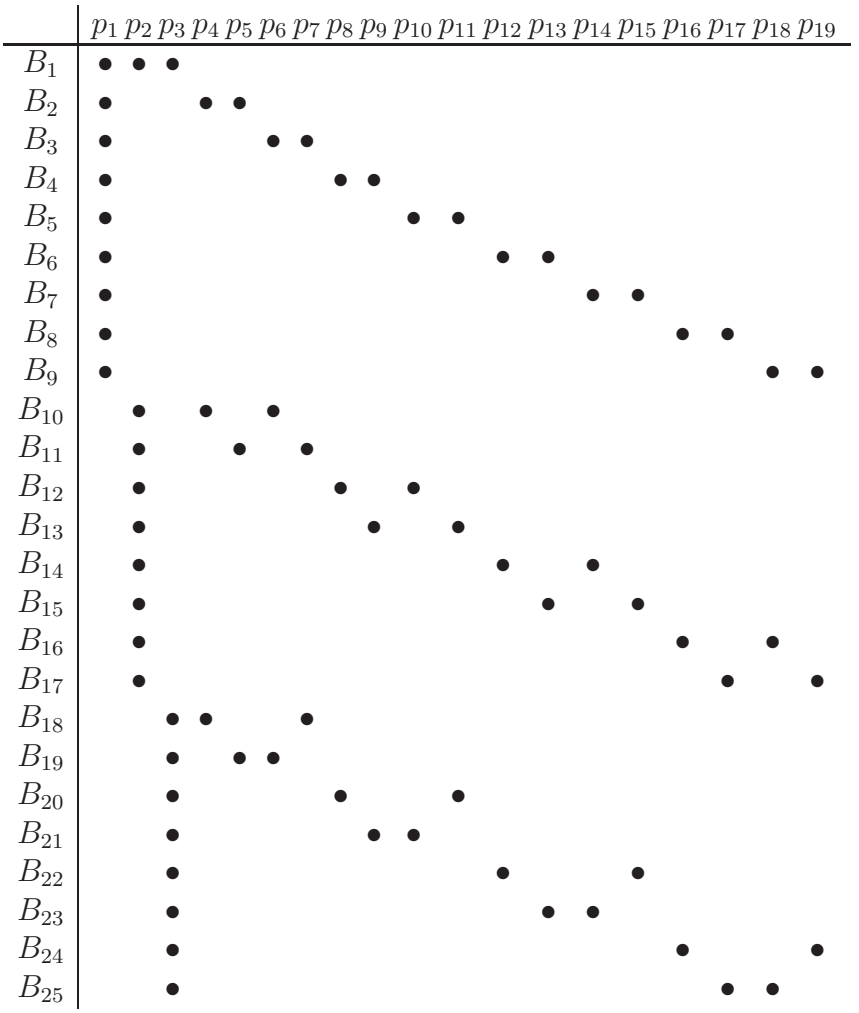
Every STS(19) contains 57 substructures that are seeds. Namely, a seed is formed by fixing an arbitrary block B_i of an STS(19) and taking the incidence structure induced by B_i and the blocks that intersect B_i. We say that this incidence structure is the seed *induced* by the block B_i.

To see that the induced incidence structure is a seed, observe that each of the three points incident with B_i is incident with 8 other blocks. These three sets of 8 blocks each must be pairwise disjoint since each pair of distinct

points is incident with exactly one block. Thus, we obtain a set of 25 blocks, which clearly satisfy (i)–(iii).

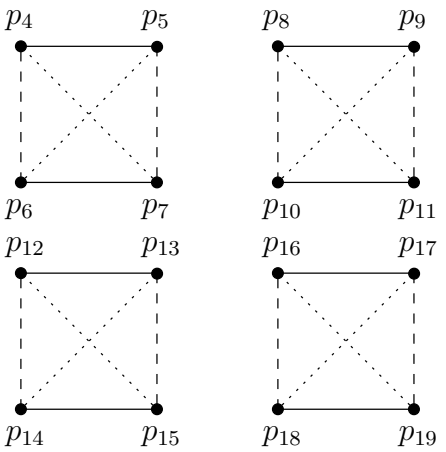
A seed may alternatively be regarded as a set of three disjoint one-factors of the complete graph K_{16}. Namely, the block in (iii) is easily seen to be unique: two blocks that intersect in p can intersect at most 4 common blocks that do not contain p. Thus, the remaining 24 blocks partition into 3 sets of 8 blocks each based on the point shared with the block in (iii). Each set of blocks constitutes a one-factor of K_{16} if we remove the common point from all the blocks. The three resulting one-factors are disjoint by (ii). The converse of this construction is straightforward.

Example 4.1 An example of a seed.



Up to isomorphism there are 14,648 seeds; these were classified using the algorithm described in the next section. To classify the STS(19), we run the main search algorithm (Algorithm 3) for each of the 14,648 nonisomorphic seeds.

Example 4.2 The three disjoint one-factors of K_{16} that correspond to the seed in Example 4.1 are depicted below.



Algorithm 3 Top-level search algorithm.

procedure *extseed*($\mathcal{S}$: incidence structure)
    Initialize Algorithm DLX based on the seed $\mathcal{S}$.
    Run Algorithm DLX.
    For each solution $Q_1, \ldots, Q_{32}$ reported by Algorithm DLX,
        Construct from $\mathcal{S}$ and $Q_1, \ldots, Q_{32}$ an STS(19), $\mathcal{X}$;
        Perform isomorph rejection on $\mathcal{X}$ and $\mathcal{S}$;
        If $\mathcal{X}$ is not rejected, report $\mathcal{X}$ as a solution.

The main search algorithm uses Algorithm DLX in the following way to create (up to labelling of the blocks) all STS(19) that contain the seed $\mathcal{S}$ as a substructure. Recall that in an STS(19) every pair of distinct points is incident with exactly one block. An STS(19) is therefore a solution to the problem of covering the $\binom{19}{2} = 171$ unordered pairs of points with 57 triples from the set of all $\binom{19}{3} = 969$ triples. Thus, to create the extensions of $\mathcal{S}$ with Algorithm DLX, we determine from $\mathcal{S}$ the $171 - 25 \cdot 3 = 96$ pairs that still need to be covered; these become the columns to be covered by Algorithm DLX. Similarly, each triple that does not intersect a block of $\mathcal{S}$ in more than one point defines a row. Each row covers precisely the $\binom{3}{2} = 3$ columns (pairs of points) that occur in the corresponding triple.

To produce an STS(19) from a seed $\mathcal{S}$ and a completion $Q_1, \ldots, Q_{32}$ reported by Algorithm DLX, we must label the 57 triples in the solution with $B_1, \ldots, B_{57}$. In practice, we can pick almost an arbitrary labelling; the only requirement that we place on the labelling is that block B_1 induces the seed $\mathcal{S}$. We state this requirement explicitly since it is needed in subsequent correctness proofs.

Requirement L. For each STS(19), $\mathcal{X}$, generated by the main search algorithm as an extension of a seed $\mathcal{S}$, we require that the block B_1 in $\mathcal{X}$ induces the seed $\mathcal{S}$.

Since 57 seeds occur in every STS(19), our algorithm gives us a complete classification of the STS(19) in the sense that Algorithm DLX generates at least one representative from every isomorphism class of STS(19). The isomorph rejection part of the algorithm then takes care that exactly one representative from every isomorphism class is output.

4.3 Generation of seeds

In this section we describe the algorithm that was used to generate the 14,648 nonisomorphic seeds for the main search.

In principle, any suitable algorithm can be used for generating the seeds since the seed classification requires only a minute fraction of the total CPU time required for the classification. The approach described in this section was chosen because it is straightforward to describe and implement. Moreover, the correctness of the algorithm is obvious.

Recall from Section 4.2 that a seed corresponds up to isomorphism to three disjoint one-factors of K_{16}. Now, the union of any two disjoint one-factors in a graph is a collection of cycles of even length. Thus, up to isomorphism the disjoint pairs of one-factors in K_{16} are characterized by the partitions of 16 into even parts, each of which is at least 4. We obtain 7 isomorphism classes:

$$16, \quad 12+4, \quad 10+6, \quad 8+8, \quad 8+4+4, \quad 6+6+4, \quad 4+4+4+4.$$

For example, the one-factors induced by any two of the points p_1, p_2, p_3 in Example 4.1 correspond to the case $4+4+4+4$. This can be easily observed by looking at any pair of one-factors in the corresponding figure in Example 4.2.

It remains to extend the 7 isomorphism classes in all possible ways by a disjoint one-factor and reject isomorphs. Even a brute-force search through all possible one-factors suffices here because K_{16} has only $15 \cdot 13 \cdot 9 \cdot 7 \cdot 5 \cdot 3 = 2{,}027{,}025$ distinct one-factors. In practice, the number is smaller because 16 of the 120 possible edges cannot be used since they occur in the two fixed one-factors. A generation algorithm for the distinct one-factors of K_{16} is straightforward to implement. (See [56] for examples of generation algorithms for elementary combinatorial structures.)

We perform isomorph rejection using a hash table that stores the canonical representative seed from each isomorphism class encountered. First, we transform a set $\mathcal{F} = \{F_1, F_2, F_3\}$ of disjoint one-factors of K_{16} on $\{p_4, \ldots, p_{19}\}$ into a seed whose blocks are

$$\{\{p_1, p_2, p_3\}\} \cup \bigcup_{i=1}^{3} \{\{p_i\} \cup e \ : \ e \in F_i\}.$$

Then, we compute the canonical representative of the seed with the help of *nauty* and the canonical labelling map $\jmath$ from Section 3.2. If the canonical representative does not occur in the hash table, we accept the seed as the representative of its isomorhism class; otherwise we reject it.

In this way 14,648 nonisomorphic seeds were obtained in a little over one hour of CPU time on a Linux workstation with a 450-MHz Pentium II CPU,

which is perfectly adequate compared with the time required by the main search.

We verified that the seed classification is correct using two independent algorithms, which we will not describe in significant detail here for reasons of space. The second algorithm generates the seeds block by block using backtrack search with an isomorph rejection step after each added block. The third algorithm utilizes a correspondence between one-factorizations of K_{16} and certain error-correcting codes. Each coordinate in these codes corresponds to a one-factor. By stopping the coordinatewise code generation algorithm in [46, Ch. 5] after three generated coordinates, we obtain a classification of the seeds in less than three minutes of CPU time at the cost of a significantly more complex algorithm.

4.4 Isomorph rejection

The most involved part of the STS(19) classification algorithm is the elimination of isomorphic STS(19) output by the main search algorithm. There are three major issues that need to be addressed.

First, the main search on different seeds must be conducted in parallel because of the considerable resource requirements. This presents a difficulty since the parallel runs should preferably be independent of each other, whereby no comparisons between isomorphism class representatives encountered in distinct runs are allowed. Second, the search is to be conducted in part on computers that do not have enough main memory to store the millions of isomorphism class representatives potentially encountered as extensions of a single seed. Third, isomorphism testing must be fast, since there are, as we now know, in the order of $7.1 \cdot 10^{11}$ STS(19) that must be tested for isomorphism.

Luckily enough, essentially all of the above difficulties are solved by a recent algorithm framework for isomorph-free exhaustive generation [63]. For reasons of speed, we shall work with block graphs of STS(19). By Theorems 2.19 and 2.20, this is essentially equivalent to working with the STS(19) themselves. Our basic isomorph rejection strategy is to use *nauty* to compute a canonical representative for the block graph of a generated STS(19), followed by a hash table query to test whether the canonical representative was encountered earlier in the search.

We can parallelize the search on different seeds by performing a test that assigns a unique *parent* seed for each isomorphism class of STS(19). This test is described in Section 4.4.

As mentioned above, the basic strategy of storing the isomorphism class representatives in a hash table will not suffice since there exist seeds for which the number of representatives to be stored is in the range of millions. For such seeds we perform an additional test, which together with the parent test suffices for complete isomorph rejection and does not require the storage of isomorphism class representatives. This test is described in Section 4.4.

In Section 4.4 we describe the implementation tricks that we used to make the tests fast enough. Of central importance is a vertex invariant based on Pasch configurations that speeds up the operation of *nauty* on block graphs and enables fast rejection of STS(19) in the parent test.

Parallelization

To enable parallelization, we must guarantee that algorithm runs performed on different seeds do not output isomorphic STS(19). This can be accomplished by using the canonical labelling and the automorphism group of the block graph output by *nauty* to test that the STS(19) originates from the correct *parent* seed. The following test is motivated by the general theory in [63].

The intuition behind the test is as follows. Recall that every STS(19) contains 57 seeds. Now, for each isomorphism class of STS(19), we have to fix (up to isomorphism) one of these seeds as the parent seed. This can be done using a canonical labelling map. Namely, since the canonical representative is unique for every isomorhism class, we can set (say) the seed induced by block B_1 in the canonical representative to be the parent seed. The test now checks whether there exists an isomorphism from the canonical representative to the generated STS(19) that maps the parent seed in the canonical representative to the seed from which the STS(19) was generated. In practice, the test is a bit more involved because we use block graphs in place of the STS. Furthermore, the test does not guarantee complete isomorph rejection in the sense that isomorphic STS may still pass through the test if they have the same parent. This incompleteness will be corrected with a second test, which is described in the next section.

The parent test is as follows. Let $\mathcal{X}$ be an STS(19) and let X be the block graph of $\mathcal{X}$. Furthermore, let κ be a canonical labelling map for block graphs of STS(19).

The parent test. Under the assumptions above, we say that $\mathcal{X}$ *passes* the parent test relative to seed $\mathcal{S}$ if the seed induced by block B_1 in $\mathcal{X}$ is isomorphic to $\mathcal{S}$ and

$$B_1 \in \{\alpha(\kappa_X^{-1}(B_1)) \ : \ \alpha \in \mathrm{Aut}(X)\}. \tag{10}$$

Otherwise we say that $\mathcal{X}$ is *rejected* in the parent test relative to $\mathcal{S}$.

We must now make the intuition rigorous and prove that the parent test behaves as intended. This amounts to checking the following items:

(i) For every isomorphism class of STS(19), at least one representative generated by the main search algorithm will pass the parent test relative to some seed.

(ii) Whenever two isomorphic STS(19) pass the parent test relative to some seeds, the seeds must be isomorphic.

Theorem 4.3 *Let $\mathcal{X}$, $\mathcal{X}'$ be two isomorphic* STS(19) *such that $\mathcal{X}$ passes the parent test relative to a seed $\mathcal{S}$ and $\mathcal{X}'$ passes the parent test relative to a seed $\mathcal{S}'$. Then, $\mathcal{S}$ and $\mathcal{S}'$ are isomorphic.*

Proof. Let X and X' be block graphs of $\mathcal{X}$ and $\mathcal{X}'$, respectively. By (10) there exist $\alpha \in \mathrm{Aut}(X)$ and $\beta \in \mathrm{Aut}(X')$ such that

$$\alpha(\kappa_X^{-1}(B_1)) = B_1 = \beta(\kappa_{X'}^{-1}(B_1)). \tag{11}$$

Because $\mathcal{X}$ and $\mathcal{X}'$ are isomorphic, the block graphs X and X' are isomorphic. Thus, $\kappa_X(X) = \kappa_{X'}(X')$, that is, $\kappa_{X'}^{-1}\kappa_X$ is an isomorphism of X onto X'. Moreover, since α and β are automorphisms, $\varphi = \beta\kappa_{X'}^{-1}\kappa_X\alpha^{-1}$ is an

isomorphism of X onto X' that satisfies $\varphi(B_1) = B_1$ by (11). By Theorem 2.19, φ extends to an isomorphism of $\mathcal{X}$ onto $\mathcal{X}'$. Since $\mathcal{S}$ and $\mathcal{S}'$ are the seeds induced by block B_1 in $\mathcal{X}$ and $\mathcal{X}'$, respectively, the extended φ gives an isomorphism of $\mathcal{S}$ onto $\mathcal{S}'$. This is because an isomorphism must map blocks that intersect onto blocks that intersect; the seed induced by a block B consists of precisely the blocks that intersect B. □

Theorem 4.4 *For every isomorphism class of* STS(19)*, at least one representative generated by the main search algorithm will pass the parent test relative to some seed.*

Proof. Let $\mathcal{X}$ be an arbitrary STS(19) and let X be its block graph. Put $B_p = \kappa_X^{-1}(B_1)$. Let $\mathcal{S}$ be the seed induced by block B_p in $\mathcal{X}$. Now, $\mathcal{S}$ is isomorphic to exactly one seed, $\mathcal{S}'$, considered in the main search.

Let φ be an isomorphism of $\mathcal{S}$ onto $\mathcal{S}'$. Extend φ to a permutation of $\mathcal{P} \cup \mathcal{B}$ in an arbitrary way and put $\mathcal{X}' = \varphi(\mathcal{X})$. Since $\mathcal{X}'$ contains $\mathcal{S}'$ as a substructure, at least one of the possible $\mathcal{X}'$ is output by the main search algorithm when it is initialized with the seed $\mathcal{S}'$. Requirement L for the main search algorithm (recall Section 4.2) implies that block B_1 in $\mathcal{X}'$ induces the seed $\mathcal{S}'$. Since φ maps $\mathcal{S}$ to $\mathcal{S}'$, we must have $\varphi(B_p) = B_1$ because the block that intersects all the other blocks in a seed is unique. Let X' be the block graph of $\mathcal{X}'$. Clearly, $X' = \varphi|_{\mathcal{B}}(X)$. Since X and X' are isomorphic, $\kappa_X^{-1}\kappa_{X'}$ is an isomorphism of X' onto X. Consequently, $\alpha = \varphi|_{\mathcal{B}}\kappa_X^{-1}\kappa_{X'} \in \mathrm{Aut}(X')$. Thus, $\mathcal{X}'$ will pass the parent test relative to $\mathcal{S}'$ since $\alpha\kappa_{X'}^{-1}(B_1) = \varphi|_{\mathcal{B}}\kappa_X^{-1}(B_1) = \varphi|_{\mathcal{B}}(B_p) = B_1$. □

Elimination of hash table queries

To eliminate the need to store isomorphism class representatives in a hash table, we employ an additional test, based on automorphisms of a seed. This test is also motivated by the general theory in [63].

The intuition behind the second test is in the following observation.

Theorem 4.5 *Let $\mathcal{S}$ be a seed and let $\mathcal{X}, \mathcal{X}'$ be* STS(19) *for which $\mathcal{S}$ is the seed induced by the block B_1. If $\mathcal{X}$ and $\mathcal{X}'$ are isomorphic and both pass the parent test relative to $\mathcal{S}$, then there exists an isomorphism of $\mathcal{X}$ onto $\mathcal{X}'$ that can be restricted to an automorphism of $\mathcal{S}$.*

Proof. Let X and X' be the block graphs of $\mathcal{X}$ and $\mathcal{X}'$. Exactly as in the proof of Theorem 4.3, we obtain an isomorhism φ from X to X' such that $\varphi(B_1) = B_1$. By Theorem 2.19, φ has an extension to an isomorphism from $\mathcal{X}$ to $\mathcal{X}'$. Because $\varphi(B_1) = B_1$, φ must map $\mathcal{S}$ to $\mathcal{S}$. □

Thus, the automorphisms of $\mathcal{S}$ can be employed to select exactly one representative for each isomorphism class of STS(19) among those STS(19) in the class that pass the parent test relative to $\mathcal{S}$. In practice, we shall select the lexicographic minimum element from every orbit of the seed automorphism group.

There is one technicality that we have to resolve before we proceed to describe the second test. Namely, in Section 4.1 we chose to treat the blocks as labelled. For purposes of the present test, we have to treat the blocks as

unlabelled. Formally, for every STS(19), $\mathcal{X}$, we put

$$u(\mathcal{X}) = \{\{p \in \mathcal{P}(\mathcal{X}) \ : \ (p, B) \in \mathcal{I}(\mathcal{X})\} \ : \ B \in \mathcal{B}(\mathcal{X})\}.$$

In other words, $u(\mathcal{X})$ is the block unlabelled version of $\mathcal{X}$.

We now describe the second test. Let $\mathcal{S}$ be a seed and let $\mathcal{X}$ be an STS(19) such that the seed induced by the block B_1 is $\mathcal{S}$. An automorphism $\varphi \in \mathrm{Aut}(\mathcal{S})$ acts on $u(\mathcal{X})$ by permutation of the points, that is, for all triples $\{p_i, p_j, p_k\} \subseteq \mathcal{P}$,

$$\{p_i, p_j, p_k\} \in u(\mathcal{X}) \qquad \Leftrightarrow \qquad \{\varphi(p_i), \varphi(p_j), \varphi(p_k)\} \in \varphi(u(\mathcal{X})).$$

The automorphism test. Under the assumptions above, we say that $\mathcal{X}$ *passes* the automorphism test if

$$\forall \varphi \in \mathrm{Aut}(\mathcal{S}) \qquad u(\mathcal{X}) \leq \varphi(u(\mathcal{X})), \tag{12}$$

where $\leq$ is the lexicographic order induced by the order on $\mathcal{P}$. Otherwise, we say that $\mathcal{X}$ is *rejected* in the automorphism test.

Note that all generated STS(19) pass the automorphism test trivially when the full automorphism group of the seed is trivial. This is very convenient because most seeds have a trivial full automorphism group; see Table 1 in the next section.

Checking the correctness of the automorphism test amounts to verifying the following items:

(i) If $\mathcal{X}$ and $\mathcal{X}'$ are distinct isomorphic STS(19) generated from a seed $\mathcal{S}$ such that both $\mathcal{X}$ and $\mathcal{X}'$ pass the parent test relative to $\mathcal{S}$, then either $\mathcal{X}$ or $\mathcal{X}'$ is rejected in the automorphism test; and

(ii) For every isomorphism class of STS(19), at least one representative generated by the main search algorithm will pass the parent test and the automorphism test.

Item (i) is proven by Theorem 4.5 provided that we have $u(\mathcal{X}) \neq u(\mathcal{X}')$ whenever $\mathcal{X}$ and $\mathcal{X}'$ are distinct STS output by Algorithm DLX. This assumption clearly holds for the main search algorithm as described in Section 4.2.

To see that Item (i) holds, suppose that $\mathcal{X}$ and $\mathcal{X}'$ are distinct isomorphic STS(19) that both pass the parent test relative to $\mathcal{S}$. Then, by Theorem 4.5 there exists a $\varphi \in \mathrm{Aut}(\mathcal{S})$ such that $\varphi(u(\mathcal{X})) = u(\mathcal{X}')$. Since $\mathcal{X}$ and $\mathcal{X}'$ are distinct, we have either $u(\mathcal{X}) < u(\mathcal{X}')$ or $u(\mathcal{X}) > u(\mathcal{X}')$. Thus, either $\mathcal{X}$ or $\mathcal{X}'$ is rejected in the automorphism test.

For Item (ii) it suffices to prove the following claim.

Theorem 4.6 *Let $\mathcal{S}$ be a seed and let $\mathcal{X}$ be an* STS(19) *for which the seed induced by the block B_1 is $\mathcal{S}$. Let φ be an arbitrary extension of an automorphism of $\mathcal{S}$ into a permutation of $\mathcal{P} \cup \mathcal{B}$. Then, $\mathcal{X}$ passes the parent test relative to $\mathcal{S}$ if and only if $\varphi(\mathcal{X})$ passes the parent test relative to $\mathcal{S}$.*

Proof. It suffices to prove the "only if" direction. Let X be the block graph of $\mathcal{X}$ and X' the block graph of $\varphi(\mathcal{X})$. Because $\mathcal{X}$ passes the parent test relative to $\mathcal{X}$, there exists an $\alpha \in \mathrm{Aut}(X)$ such that $\alpha\kappa_X^{-1}(B_1) = B_1$.

Since φ maps $\mathcal{S}$ to $\mathcal{S}$, the seed induced by block B_1 in $\varphi(\mathcal{X})$ is $\mathcal{S}$ and $\varphi(B_1) = B_1$. We must still show that (10) holds. Clearly, $\varphi|_{\mathcal{B}}$ is an isomorphism from X to X'. Thus, $\beta = \varphi|_{\mathcal{B}}\alpha\kappa_X^{-1}\kappa_{X'} \in \mathrm{Aut}(X')$. But then,

$$\beta\kappa_{X'}^{-1}(B_1) = \varphi|_{\mathcal{B}}\alpha\kappa_X^{-1}(B_1) = \varphi|_{\mathcal{B}}(B_1) = B_1$$

and $\varphi(\mathcal{X})$ passes the parent test relative to $\mathcal{S}$. □

We now argue that Item (ii) holds. From Theorem 4.4 we know that at least one representative from every isomorphism class of STS(19) will pass the parent test relative to some seed. For an arbitrary isomorphism class, let $\mathcal{X}$ be this representative. The previous theorem implies that if $\mathcal{X}$ passes the parent test relative to $\mathcal{S}$, then so does the lexicographic minimum of its orbit under $\mathrm{Aut}(\mathcal{S})$, which *will* pass the automorphism test. This minimum representative is generated by the main search algorithm, which establishes (ii).

Implementation details

In this section we describe the implementations of the parent test and the automorphism test. We start with the parent test.

We use *nauty* in computing the canonical labelling for block graphs for the purposes of the parent test. The use of a vertex invariant is required to guarantee good performance from *nauty* on these strongly regular graphs. For this purpose, we define a vertex invariant based on the Pasch configurations of the underlying STS(19). It will also turn out that this invariant provides a quick way to reject many STS(19) in the parent test without constructing the block graph or running *nauty*, which provides a significant performance gain. The use of Pasch configurations as an isomorphism invariant is not new [32, 76].

The vertex invariant is as follows. Let $\mathcal{X}$ be an STS(v) and let X be the block graph of $\mathcal{X}$. For a block $B \in \mathcal{B}(\mathcal{X})$, denote by $P(B)$ the number of Pasch configurations in $\mathcal{X}$ that contain the block B. Suppose $\{P(B) : B \in \mathcal{B}(\mathcal{X})\} = \{p_1, \ldots, p_m\}$, $p_1 > \cdots > p_m$. Put $\iota_P(X) = (V_1, \ldots, V_m)$, where $V_i = \{B \in \mathcal{B}(\mathcal{X}) : P(B) = p_i\}$ for all $i \in \{1, \ldots, m\}$. In other words, ι_P partitions the vertex set of the block graph so that the first cell contains the blocks that have the maximum number of occurences in Pasch configurations, the second cell contains the blocks that have the next largest number of occurences in Pasch configurations, and so on.

It is not immediately clear that ι_P is well-defined because the definition uses the underlying STS(v) from which the block graph was derived.

Theorem 4.7 *For $v \geq 19$, the function ι_P is a well-defined vertex invariant for the set of block graphs of* STS(v).

Proof. Let X be the block graph of an STS(v), $\mathcal{X}$, and let $\varphi \in \mathrm{Sym}(\mathcal{B}(\mathcal{X}))$. Let $\mathcal{X}'$ be any STS(v) such that its block graph $X' = \varphi(X)$. It suffices to verify (7) for ι_P. (To establish well-definedness, take $\varphi = \mathbf{I}$.) Put $\iota_P(X) = (V_1, \ldots, V_m)$ and $\iota_P(X') = (V'_1, \ldots, V'_{m'})$. By Theorems 2.19 and 2.20, φ has an extension to an isomorphism from $\mathcal{X}$ to $\mathcal{X}'$. Thus, $P(B) = P(\varphi(B))$

for all $B \in \mathcal{B}(\mathcal{X})$, which implies $m = m'$ and

$$\begin{aligned}\varphi(V_i) &= \{\varphi(B) \ : \ P(B) = p_i, \ B \in \mathcal{B}(\mathcal{X})\} \\ &= \{\varphi(B) \ : \ P(\varphi(B)) = p_i, \ B \in \mathcal{B}(\mathcal{X})\} \\ &= \{B' \ : \ P(B') = p_i, \ B' \in \mathcal{B}(\mathcal{X}')\} = V_i'\end{aligned}$$

for all $i \in \{1, \ldots, m\}$. Thus, (7) holds for ι_P. □

By Theorem 2.23, the Pasch configuration invariant ι_P coincides on block graphs of STS(v) with the vertex invariant that partitions the vertices of a graph according to the number of 4-cliques in which a vertex occurs. The Pasch configuration invariant is much faster to compute, however, since by Theorem 2.23 every block of an STS(19) occurs in 232 configurations of four pairwise intersecting blocks that do not form a Pasch configuration.

The following observation enables quick rejection of STS(19) in the parent test when ι_P is used.

Theorem 4.8 *Let X be a block graph of an* STS(19)*. If the canonical labelling map κ in the parent test is constrained with the initial partition $\iota_P(X)$, then X passes the parent test (relative to any seed) only if B_1 occurs in the first cell of $\iota_P(X)$.*

Proof. Suppose $\iota_P(X) = (V_1, \ldots, V_m)$ and $c(\iota_P(X)) = (W_1, \ldots, W_m)$, where $c(\iota_P(X))$ is defined as in Section 3.2 (subject to the total order $B_1 < B_2 < \cdots < B_{57}$). Since ι_P is a vertex invariant, any automorphism $\alpha \in \mathrm{Aut}(X)$ satisfies $\alpha(\iota_P(X)) = \iota_P(X)$ by (7). So, since $\kappa^{-1}_{X,\iota_P(X)}(W_1) = V_1$ by (5), we have $\alpha\kappa^{-1}_{X,\iota_P(X)}(W_1) = V_1$ for all $\alpha \in \mathrm{Aut}(X)$. Because $B_1 \in W_1$, X will not pass the parent test unless $B_1 \in V_1$. □

In other words, if the vertex invariant ι_P is used in computing canonical labelling for block graphs, then a block graph X will not pass the parent test unless the number of Pasch configurations in which the block B_1 occurs is the maximum taken over all blocks of the STS(19).

This observation translates into the following algorithm for performing the parent test. Let $\mathcal{X}$ be an STS(19) that satisfies Requirement L.

(i) Starting from $i = 1$, compute for every block B_i the number of Pasch configurations $P(B_i)$ in which the block occurs.

(ii) If $P(B_i) > P(B_1)$ for some i, then reject the STS(19).

(iii) Construct the block graph, X, of $\mathcal{X}$.

(iv) Construct the initial partition $\pi = \iota_P(X)$ from the array P. (Partition $\mathcal{B}$ into maximal cells of blocks B with $P(B)$ constant, sort the cells into order of decreasing $P(B)$.)

(v) Input X and π to *nauty*.

(vi) Using the automorphism orbits of X and the canonical labelling κ_X output by *nauty*, test whether B_1 and $\kappa_X^{-1}(B_1)$ are in the same automorphism orbit. If so, accept the STS(19); otherwise reject the STS(19).

Since Step (i) of the algorithm requires a very fast implementation, we shall outline it a bit more. Internally, the algorithm represents an STS(19) as an array of 57 integers, where the ith element of the array contains the lexicographic rank of the 3-subset of points incident with B_i. (Ranking and unranking algorithms for subsets are considered in [56].) This representation allows rapid testing of whether two blocks intersect, which is useful also for Step (iii) of the algorithm. For the intersection test we maintain a look-up table of $\binom{19}{3} = 969$ words. The ith word in the table contains the bit representation of the 3-subset that has lexicographic rank i, that is, 19 bits of the word correspond to the points; the three bits that correspond to the elements of the rank i 3-subset are set to 1, all the other bits of the word are set to 0. Now, blocks B_i and B_j intersect if and only if the bitwise-and of the corresponding look-up table words is nonzero.

The number of Pasch configurations in which B_i occurs can be computed with the help of the fast intersection test as follows. A Pasch configuration can only be formed by B_i and three blocks, B_j, B_k, B_ℓ, that intersect B_i so that no two of the three intersect B_i in the same point. Such a set B_j, B_k, B_ℓ forms a Pasch configuration with B_i if and only if the three blocks intersect pairwise but there exists no point common to all three blocks. The look-up table helps here also, since three blocks are incident with a common point if and only if the bitwise-and of the three corresponding look-up table words is nonzero. An alternative algorithm for finding the Pasch configurations in a Steiner triple system is considered in [75].

We now turn to the implementation of the automorphism test. Let $\mathcal{S}$ be the seed relative to which the automorphism test is performed. First, we require a list of the elements of $\mathrm{Aut}(\mathcal{S})$, which we compute from the automorphism group generators output by *nauty* on the incidence graph $I_\mathcal{S}$. (Recall Section 3.2 and Theorem 3.4.) We expand the generators into the full group using the reduced representation algorithm of Jerrum [43]. In practice, even the naïve straightforward algorithm of computing multiplicative closure of the generators suffices since the groups have small order and the group needs to be generated only once in the initialization phase of the algorithm.

Speed is of essence also in the automorphism test (in particular since the automorphism test will be performed before the parent test), so we will sketch the implementation in more detail. Let $\mathcal{X}$ be the STS(19) that is to be tested and let $\varphi \in \mathrm{Aut}(\mathcal{S})$. Recall that $\mathcal{X}$ is represented internally as an array of 57 integers, where the ith element of the array contains the lexicographic rank of the 3-subset of points incident with block B_i. We represent $u(\mathcal{X})$ (and respectively, $\varphi(u(\mathcal{X}))$) by a bit vector of length $\binom{19}{3} = 969$, where bit position i is set to one if and only if a block of $\mathcal{X}$ is incident with the 3-subset of points that has lexicographic rank i. The bit representation for $\varphi(u(\mathcal{X}))$ is constructed analogously. Here we take advantage of the fact that we can precompute the permutation that φ induces on the lexicographic ranks of 3-subsets of $\mathcal{P}$, and use this permutation in constructing the bit vector for $\varphi(u(\mathcal{X}))$ from the array representation of $\mathcal{X}$. Lexicographic comparison between $u(\mathcal{X})$ and $\varphi(u(\mathcal{X}))$ can be computed by comparing the bit vectors word by word, which is reasonably fast since at most 31 word comparison operations are required if the word length is 32 bits or more.

Because the automorphism test must consider every $\varphi \in \mathrm{Aut}(\mathcal{S})$, the test is slow to perform on those seeds for which the automorphism group is large. Moreover, the memory requirement is 969 words for each element of $\mathrm{Aut}(\mathcal{S})$ in the present implementation. To avoid this difficulty, we use a hash table of canonical representatives for isomorph rejection for seeds with a large automorphism group.

To sum up, the isomorph rejection algorithms for seeds with small and large automorphism groups, respectively, are given in pseudocode as Algorithms 4 and 5.

Algorithm 4 Isomorph rejection for seeds with $\mathrm{Aut}(\mathcal{S})$ small.

procedure *smallreject*($\mathcal{X}$: incidence structure)
  For every $\varphi \in \mathrm{Aut}(\mathcal{S})$,
    If $\varphi(u(\mathcal{X})) < u(\mathcal{X})$, reject $\mathcal{X}$ and return.
  For every $i = 1, 2, \ldots, 57$,
    Set $P(B_i) \leftarrow$ the number of Pasch configurations
      in $\mathcal{X}$ that contain B_i;
    If $P(B_i) > P(B_1)$, reject $\mathcal{X}$ and return.
  Construct from P the initial partition π.
  Construct the block graph, X, of $\mathcal{X}$.
  Run *nauty* on X, π.
  If B_1 and $\kappa_{X,\pi}^{-1}(B_1)$ are **not** on the same orbit under $\mathrm{Aut}(X)$,
    reject $\mathcal{X}$ and return.
  Accept $\mathcal{X}$ and return.

Algorithm 5 Isomorph rejection for seeds with $\mathrm{Aut}(\mathcal{S})$ large.

procedure *largereject*($\mathcal{X}$: incidence structure)
  For every $i = 1, 2, \ldots, 57$,
    Set $P(B_i) \leftarrow$ the number of Pasch configurations
      in $\mathcal{X}$ that contain B_i;
    If $P(B_i) > P(B_1)$, reject $\mathcal{X}$ and return.
  Construct from P the initial partition π.
  Construct the block graph, X, of $\mathcal{X}$.
  Run *nauty* on X, π.
  If B_1 and $\kappa_{X,\pi}^{-1}(B_1)$ are **not** on the same orbit under $\mathrm{Aut}(X)$,
    reject $\mathcal{X}$ and return.
  Construct the canonical representative $\kappa_{X,\pi}(X)$.
  If $\kappa_{X,\pi}(X)$ occurs in the hash table, reject $\mathcal{X}$ and return.
  Insert $\kappa_{X,\pi}(X)$ into the hash table.
  Accept $\mathcal{X}$ and return.

Table 1: The seeds for STS(19).

$\lvert\mathrm{Aut}(\mathcal{S})\rvert$	Seeds
1	11,706
2	2,218
3	14
4	412
6	20
8	127
12	13
16	50
24	16
32	25
36	3
40	1
48	5
64	9

$\lvert\mathrm{Aut}(\mathcal{S})\rvert$	Seeds
72	2
96	7
120	1
128	6
192	2
256	4
288	1
512	2
768	1
1,536	1
1,728	1
36,864	1
Total	14,648

4.5 The search

We conclude this chapter by giving a brief description on how the classification of STS(19) was carried out in practice.

First, the 14,648 nonisomorphic seeds were classified using the algorithm described in Section 4.3. Table 1 lists the full automorphism group order for the seeds. The seed classification took a little over one hour on a Linux workstation with a 450-MHz Pentium II CPU. We applied the `cellquads` vertex invariant on levels 0 and 1 of the *nauty* search tree in computing canonical labelling for incidence graphs.

The main search was distributed using the batch system `autoson` [62] to a network of 65 IBM Intellistation Pro workstations with 450-MHz Pentium II CPUs and 15 other workstations with CPUs ranging from 1GHz Athlon Thunderbird to 200-MHz Pentium. All of the workstations ran the Linux operating system.

Each `autoson` job executed the main search algorithm on one of the 14,648 seeds. The CPU time required by each job was recorded using the Linux library function `times(2)`. Each job recorded the search statistics and the STS(19) of interest on a separate file. The recorded statistics were

(i) the total number of extensions a seed has to an STS(19); and

(ii) the number of STS(19) that pass the isomorph rejection; and

(iii) for each STS(19) accepted by the isomorph rejection phase, the full automorphism group order and the number of Pasch configurations.

In case an accepted STS(19) was anti-Pasch or the full automorphism group had order at least 12, the STS(19) was saved on a separate file.

For the 11 seeds whose automorphism group had order greater than 200, Algorithm 5 was used for isomorph rejection. The maximum number of

block graphs that had to be stored in the hash table was 100,813. Algorithm 4 was used for isomorph rejection on all the other seeds.

The total time requirement for the main search was about two years of CPU time, or about two and a half weeks real time with the available computational resources. The CPU time was fairly evenly distributed on the different seeds in the sense that there were no seeds whose CPU time requirement was prohibitively large: the slowest job took 14 hours of CPU time and the fastest job took 7 minutes. (Note that these execution times have not been scaled to uniform processor speed.)

All of the algorithms were implemented in the C programming language and compiled using the GNU C compiler.

5 PRESCRIBING A GROUP OF AUTOMORPHISMS

In this chapter, we first describe an algorithm for classifying the STS(19) with a nontrivial full automorphism group. We then apply the algorithm to verify that the classification of these STS(19) obtained as part of the main search is correct. The primary motivation for the material in this chapter was to resolve the observed discrepancy for automorphism group orders 2 and 3 between the main search data and the results reported by Colbourn, Magliveras, and Stinson [16]. This discrepancy will be discussed in more detail in the following chapter.

Our approach to classifying the STS(19) with a nontrivial automorphism group is based on results in [16] concerning the automorphisms that an STS(19) can admit. Namely, each STS(19) with a nontrivial automorphism group must admit at least one of the so-called *basic* automorphisms as an automorphism.

To classify the STS(19) with a nontrivial automorphism group, it suffices to construct, for each basic automorphism α, all STS(19) that admit $\langle\alpha\rangle$ as a group of automorphisms. This construction problem can be formulated as an exact cover problem, which we solve using Algorithm DLX followed by an isomorph rejection step. To make the exact cover search feasible, we must again perform isomorph rejection prior to the search. This amounts to starting the search from a select collection of partial solutions.

The organization of this chapter is as follows. Section 5.1 describes the results in [16] on automorphisms of STS(19). Section 5.2 outlines the classification algorithm for STS(19) that admit a given group of automorphisms. Section 5.3 describes the starting points of the search for different basic automorphism types. The implementation of the search is described in Section 5.4.

5.1 Automorphisms of STS(19)

In this section we follow Colbourn, Magliveras, and Stinson [16] in determining a collection of *basic* automorphisms such that any STS(19) with a nontrivial automorphism must admit at least one automorphism from this collection.

Let $\alpha \neq \mathbf{I}$ be a permutation of degree 19 with cycle type $a_1^{n_1} \cdots a_t^{n_t}$. Let ℓ

be the least common multiple of $a_1, \ldots, a_t$ and let p be any prime that divides at least one of $a_1, \ldots, a_t$. Then $\alpha^{\ell/p}$ has cycle type $1^n p^m$, where $n + pm = 19$ and m is the number of a_i that are divisible by the maximum power of p that divides ℓ. Thus, any STS(19) that admits a nontrivial automorphism α must also admit an automorphism with cycle type $1^n p^m$, where p is prime and $n + pm = 19$.

The following observation and Theorem 2.13 show that $n \in \{0, 1, 3, 7, 9\}$.

Lemma 5.1 [16] *The set of points fixed by any automorphism of a Steiner triple system induces a subsystem.*

Proof. Let $\mathcal{X}$ be an STS and let $\alpha \in \mathrm{Aut}(\mathcal{X})$. The claim is obvious for $\alpha = \mathbf{I}$. Otherwise, let $\mathcal{Q}$ be the set of all points fixed by α. Now, no block $\{x, y, z\}$ with $x, y \in \mathcal{Q}$ and $z \notin \mathcal{Q}$ can exist because then $\{x, y, \alpha(z)\}$ would be a block, which is a contradiction since $\alpha(z) \neq z$. Thus, $\mathcal{Q}$ induces a subsystem. □

The case $n = 9$ is impossible as is demonstrated by the following [16]: If there existed a nontrivial automorphism fixing nine points of an STS(19), then the deletion of these nine points yields a one-factorization of K_{10} with an automorphism fixing every one-factor. The following lemam shows that no such one-factorization exists.

Lemma 5.2 [**16, Lemma 1.1**] *For $n \geq 2$ there exists a one-factorization of K_{2n} having a nontrivial automorphism that fixes every one-factor if and only if n is even.*

Proof. Suppose that $\mathcal{F} = \{F_1, \ldots, F_{2n-1}\}$ is a one-factorization of K_{2n}, that n is odd, and that α is an automorphism that fixes every one-factor F_i. First, suppose that α fixes a vertex x. Then, since every one-factor is fixed, every edge incident with x is fixed. Hence, $\alpha = \mathbf{I}$. Thus, an automorphism $\alpha \neq \mathbf{I}$ that fixes every one-factor must have type t^u for some $t \geq 2$, $tu = 2n$. Suppose $t > 2$. Let $(x_1, x_2, \ldots, x_t)$ be a t-cycle of α. Then, since $\{x_1, x_2\} \in F_i$ for some i, we have that $\{\alpha(x_1), \alpha(x_2)\} = \{x_2, x_3\} \in F_i$, which is impossible. Hence, $t = 2$ and $u = n$ is the only remaining possibility. Suppose $\alpha = (x_1, x_2)(x_3, x_4) \cdots (x_{2n-1}, x_{2n})$. Every edge in the one-factorization $\mathcal{F}$ is either fixed or moved by α. In the latter case, the edge lies on an orbit of length two. Because each one-factor has an odd number of edges since n is odd, any one-factor must contain an odd number of fixed edges. But there are $2n - 1$ one-factors in $\mathcal{F}$ and only n fixed edges. Hence, also the case $t = 2$, $u = n$ is impossible. For the converse part of the proof, see [16]. □

Hence, all STS(19) that admit a nontrivial automorphism must admit an automorphism with cycle type $1^n p^m$, where p is prime, $n + pm = 19$, and $n \in \{0, 1, 3, 7\}$. Automorphisms of this type are called *basic* automorphisms. It is easy to see that there are six types of basic automorphisms:

$$19^1, \quad 1^1 2^9, \quad 1^1 3^6, \quad 1^3 2^8, \quad 1^7 2^6, \quad 1^7 3^4.$$

To construct all STS(19) with a nontrivial full automorphism group, it suffices to construct all STS(19) that admit at least one basic automorphism as an automorphism.

5.2 The construction algorithm

In this section we describe an algorithm that generates up to isomorphism all STS(19) with a prescribed group of automorphisms.

The construction of designs with a prescribed group of automorphisms as described in this section is not new. Indeed, block orbit by block orbit construction of t-(v, k, λ) designs with a prescribed group of automorphisms can be traced back to the seminal article of Kramer and Mesner [55]. (See [33] and the references therein for further developments of the technique.)

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_{19}\}$ be a fixed set of 19 points and let $G \leq \mathrm{Sym}(\mathcal{P})$ be the desired group of automorphisms. (In practice, we set $G = \langle\alpha\rangle$ for each basic automorphism α, see Section 5.3.) The key observation is that all STS(19) over $\mathcal{P}$ that have G as a group of automorphisms are unions of orbits of G on the set of all 3-subsets of $\mathcal{P}$. Thus, to construct up to isomorphism all STS(19) that have (a group that is permutation isomorphic to) G as a group of automorphisms, it suffices to consider all unions of 3-subset orbits of G such that every 2-subset occurs in exactly one 3-subset.

We formulate the construction problem for such STS(19) again as an exact cover problem, in which the task is to cover the 2-subsets of points with orbits of 3-subsets. Those 3-subset orbits that cover a 2-subset more than once can clearly be rejected from consideration. Moreover, instead of covering all 2-subsets, it suffices to cover orbits of 2-subsets. This is because any 3-subset orbit either covers all the 2-subsets or none of the 2-subsets in a 2-subset orbit. Algorithms that compute the orbits of a permutation group can be found in the references given in Section 3.3.

We use Algorithm DLX to generate all the solutions to the orbit covering problem, and perform isomorph rejection on the resulting STS(19) using a hash table that contains canonical representatives of the STS(19) generated so far. As a side effect of the computation of the canonical representative, we obtain generators for the full automorphism group of the STS(19).

Algorithm DLX features no isomorph rejection, so care must be taken to initialize the algorithm so that redundant symmetry in the search space is eliminated. The starting points of the search for different basic automorphism types are described in the next section.

5.3 Starting points for the search

In this section we describe the starting points of the search that we used for different basic automorphism types.

Type 19^1. Without loss of generality we may assume that the automorphism of type 19^1 is

$$\alpha = (p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{17}, p_{18}, p_{19}).$$

In this case it is not necessary to fix any blocks since the search is fast enough even with no symmetry reduction. In the following cases, however, a number of blocks must be fixed to speed up the search.

Type $1^1 2^9$. We may assume that

$$\alpha = (p_2, p_3)(p_4, p_5)(p_6, p_7)(p_8, p_9)(p_{10}, p_{11})(p_{12}, p_{13})(p_{14}, p_{15})(p_{16}, p_{17})(p_{18}, p_{19}).$$

Since the pairs $\{p_{2i}, p_{2i+1}\}$ for $i \in \{1, \ldots, 9\}$ must each occur in exactly one block, an STS(19) that admits α as an automorphism must contain the triples $\{p_1, p_{2i}, p_{2i+1}\}$ for $i \in \{1, \ldots, 9\}$. (A block of the form $\{p_j, p_{2i}, p_{2i+1}\}$ with $j > 1$ cannot occur in an STS(19) that admits α as an automorphism since then $\{\alpha(p_j), p_{2i}, p_{2i+1}\}$ would also be a block, which is impossible because the pair $\{p_{2i}, p_{2i+1}\}$ would then occur in two distinct blocks.)

Since $\{p_1, p_2, p_3\}$ is a block, the blocks incident with p_2 form a one-factor of K_{16} on $\{p_4, \ldots, p_{19}\}$ that is disjoint from the one-factor $\{\{p_{2i}, p_{2i+1}\} : i \in \{2, \ldots, 9\}\}$ formed by the blocks incident with p_1. This pair of disjoint one-factors forms a collection of even-length cycles of length at least 4. It is straightforward to check that cycles of length 4 cannot occur because α is an automorphism. Thus, there are three alternatives for the cycle structure, namely one 16-cycle; one 10-cycle and one 6-cycle; and two 8-cycles. Without loss of generality the second one-factor may hence be fixed as one of the following:

$$\{p_4, p_6\}, \{p_5, p_8\}, \{p_7, p_{10}\}, \{p_9, p_{12}\}, \{p_{11}, p_{14}\}, \{p_{13}, p_{16}\}, \{p_{15}, p_{18}\}, \{p_{17}, p_{19}\};$$
$$\{p_4, p_6\}, \{p_5, p_8\}, \{p_7, p_{10}\}, \{p_9, p_{12}\}, \{p_{11}, p_{13}\}, \{p_{14}, p_{16}\}, \{p_{15}, p_{18}\}, \{p_{17}, p_{19}\};$$
$$\{p_4, p_6\}, \{p_5, p_8\}, \{p_7, p_{10}\}, \{p_9, p_{11}\}, \{p_{12}, p_{14}\}, \{p_{13}, p_{16}\}, \{p_{15}, p_{18}\}, \{p_{17}, p_{19}\}.$$

Each row gives an alternative one-factor; the point p_2 has been omitted from each block for brevity.

Type $1^1 3^6$. We may assume that

$$\alpha = (p_2, p_3, p_4)(p_5, p_6, p_7)(p_8, p_9, p_{10})(p_{11}, p_{12}, p_{13})(p_{14}, p_{15}, p_{16})(p_{17}, p_{18}, p_{19}).$$

Any block that contains the point p_1 must occur in an orbit of length 3. Moreover, the two other points in such a block must occur in different 3-cycles. Thus, we may include the blocks $\{p_1, p_2, p_5\}, \{p_1, p_8, p_{11}\}, \{p_1, p_{14}, p_{17}\}$.

Type $1^3 2^8$. We may assume that

$$\alpha = (p_4, p_5)(p_6, p_7)(p_8, p_9)(p_{10}, p_{11})(p_{12}, p_{13})(p_{14}, p_{15})(p_{16}, p_{17})(p_{18}, p_{19}).$$

The points p_1, p_2, p_3 form a subsystem by Lemma 5.1, so we may include the block $\{p_1, p_2, p_3\}$. Now the pairs $\{p_{2i}, p_{2i+1}\}$, $i \in \{2, \ldots, 9\}$, must occur in exactly one block, so we have that for every $i \in \{2, \ldots, 9\}$ there exists a $j \in \{1, 2, 3\}$ such that $\{p_j, p_{2i}, p_{2i+1}\}$ is a block.

It is easy to see that the aforementioned blocks are the only orbits of length 1 under $\langle\alpha\rangle$; all the other orbits have length 2. Thus, each of the points p_1, p_2, p_3 must occur in an odd number of orbits of length 1, one of which is the block $\{p_1, p_2, p_3\}$. So, disregarding the block $\{p_1, p_2, p_3\}$, we obtain a division into 4 cases, based on how the points p_1, p_2, p_3 are incident with the 8 orbits of length 1.

Case 8. We may assume that point p_1 is incident with all the 8 orbits of length 1. In this case we may also include the blocks $\{p_2, p_{4i}, p_{4i+2}\}$, where $i \in \{1, \ldots, 4\}$.

Case $6+2$. We may assume that point p_1 is incident with 6 orbits of length 1 and point p_2 is incident with 2 orbits of length 1. So, assuming that p_1 is incident with the length-1 orbits $\{p_1, p_{2i}, p_{2i+1}\}$ for $i \in \{2, \ldots, 7\}$, we may also include the blocks $\{p_2, p_{16}, p_{17}\}$, $\{p_2, p_{18}, p_{19}\}$, $\{p_1, p_{16}, p_{18}\}$, and the blocks $\{p_2, p_{4i}, p_{4i+2}\}$ for $i \in \{1, 2, 3\}$.

Case $4+4$. We may assume that points p_1 and p_2 are both incident with 4 orbits of length 1. Assuming that p_1 is incident with the length-1 orbits $\{p_1, p_{2i}, p_{2i+1}\}$ for $i \in \{2, \ldots, 5\}$ and that p_2 is incident with the length-1 orbits $\{p_2, p_{2i}, p_{2i+1}\}$ for $i \in \{6, \ldots, 9\}$, we may also include the blocks $\{p_1, p_{4i}, p_{4i+2}\}$ for $i \in \{3, 4\}$, and the blocks $\{p_2, p_{4i}, p_{4i+2}\}$ for $i \in \{1, 2\}$.

Case $4+2+2$. We may assume that point p_1 is incident with 4 orbits of length 1 and that points p_2, p_3 are both incident with 2 orbits of length 1. We may further assume that the length-1 orbits are $\{p_1, p_{2i}, p_{2i+1}\}$ for $i \in \{2, \ldots, 5\}$, $\{p_2, p_{2i}, p_{2i+1}\}$ for $i \in \{6, 7\}$, and $\{p_3, p_{2i}, p_{2i+1}\}$ for $i \in \{7, 8\}$. By including more blocks we obtain four alternatives, all of which must be considered. We may assume that either

(i) $\{p_1, p_{12}, p_{16}\}$, $\{p_1, p_{14}, p_{18}\}$ are blocks, in which case we may further assume that either $\{p_2, p_4, p_6\}$, $\{p_2, p_8, p_{10}\}$; or $\{p_2, p_4, p_6\}$, $\{p_2, p_8, p_{16}\}$, $\{p_2, p_{10}, p_{18}\}$ are blocks; or that

(ii) $\{p_1, p_{12}, p_{14}\}$, $\{p_1, p_{16}, p_{18}\}$ are blocks, in which case we may further assume that either $\{p_2, p_4, p_6\}$, $\{p_2, p_8, p_{10}\}$, $\{p_2, p_{16}, p_{19}\}$; or $\{p_2, p_4, p_6\}$, $\{p_2, p_8, p_{16}\}$, $\{p_2, p_{10}, p_{18}\}$ are blocks.

Type $\mathbf{1^7 2^6}$. We may assume

$$\alpha = (p_8, p_9)(p_{10}, p_{11})(p_{12}, p_{13})(p_{14}, p_{15})(p_{16}, p_{17})(p_{18}, p_{19}).$$

The fixed points $p_1, \ldots, p_7$ form a subsystem, which we may assume to be

$$\begin{gathered}\{p_1, p_2, p_3\}, \{p_1, p_4, p_5\}, \{p_1, p_6, p_7\}, \{p_2, p_4, p_6\}, \\ \{p_2, p_5, p_7\}, \{p_3, p_4, p_7\}, \{p_3, p_5, p_6\}.\end{gathered} \tag{13}$$

As in the previous case, a consideration of the orbits of length 1 shows that each fixed point of α must be incident with an even number of the 6 orbits of length 1 that do not belong to the subsystem (13). We obtain a division into 3 subcases:

Case 6. Since the full automorphism group of the STS(7) is 2-transitive, we may assume that p_1 is the point incident with all the 6 orbits of length 1. Furthermore, we may fix the blocks $\{p_2, p_{4i}, p_{4i+2}\}$ for $i \in \{2, 3, 4\}$.

Case $4+2$. By 2-transitivity we may assume that p_1 is incident with 4 orbits of length 1 and p_2 is incident with 2 orbits of length 1. Assuming that p_1 is incident with orbits $\{p_1, p_{2i}, p_{2i+1}\}$ for $i \in \{4, 7\}$ and that p_2 is incident with orbits $\{p_2, p_{2i}, p_{2i+1}\}$ for $i \in \{8, 9\}$, we may include the block $\{p_1, p_{16}, p_{18}\}$ and the blocks $\{p_2, p_{4i}, p_{4i+2}\}$ for $i \in \{2, 3\}$.

Case $2+2+2$. The full automorphism group of the STS(7) has two orbits on the set of 3-subsets of $\{p_1, \ldots, p_7\}$. For the fixed subsystem (13) we may select $\{p_1, p_2, p_3\}$ and $\{p_1, p_2, p_4\}$ to be the representatives of these orbits. From these we obtain two alternatives, both of which must be considered. Namely, we include the blocks $\{p_1, p_{2i}, p_{2i+1}\}$ for $i \in \{4, 5\}$, $\{p_2, p_{2i}, p_{2i+1}\}$ for $i \in \{6, 7\}$, and $\{p_j, p_{2i}, p_{2i+1}\}$ for $i \in \{8, 9\}$, where either $j = 3$ or $j = 4$. By including further blocks we split the alternatives $j = 3, 4$ into two. Namely, having fixed either $j = 3, 4$, we may include either the blocks $\{p_1, p_{12}, p_{14}\}$, $\{p_1, p_{16}, p_{18}\}$ or the blocks $\{p_1, p_{12}, p_{16}\}$, $\{p_1, p_{14}, p_{18}\}$. Thus, in total four alternatives need to be considered.

Type $1^7 3^4$. We may assume

$$\alpha = (p_8, p_9, p_{10})(p_{11}, p_{12}, p_{13})(p_{14}, p_{15}, p_{16})(p_{17}, p_{18}, p_{19}).$$

We may again include the sub-STS(7) as in (13). This leaves $57 - 7 = 50$ blocks unspecified, all of which must occur in orbits of length 1 or 3. Since $50 = 3 \cdot 16 + 2$ is not divisible by 3, there exist at least two orbits of length 1. We may thus include the blocks $\{p_8, p_9, p_{10}\}$ and $\{p_{11}, p_{12}, p_{13}\}$. Now, we may without loss of generality assume that the blocks incident with the pairs $\{p_{14}, p_{15}\}$ and $\{p_{17}, p_{18}\}$ are $\{p_8, p_{14}, p_{15}\}$ and $\{p_{11}, p_{17}, p_{18}\}$. This is because exactly 8 orbits of length 3 are incident with the points p_8, p_9, p_{10}, of which 7 orbits are incident with one of the points $p_1, \ldots, p_7$. The blocks in the one remaining orbit of length 3 must therefore cover a pair of points from some 3-cycle of α. Since the same argument applies to the points p_{11}, p_{12}, p_{13}, we may without loss of generality fix the above mentioned blocks.

5.4 The search

The classification of the STS(19) with a nontrivial automorphism group took approximately one and a half days of CPU time on a Linux workstation with a 450-MHz Pentium II CPU. It took 23 hours of CPU time to run the classification algorithm from the 19 starting points specified in Section 5.3. This resulted in a collection of 216,857 STS(19), from which the isomorphic STS(19) were removed in 10 hours of CPU time with an unoptimized algorithm. In total 164,758 nonisomorphic STS(19) were obtained in this way.

All of the algorithms were implemented in the C programming language and compiled using the GNU C compiler.

6 THE STEINER TRIPLE SYSTEMS OF ORDER 19

This chapter contains results on STS(19) extracted from the main search data and from the independent classification of the STS(19) with a nontrivial automorphism group.

For the STS(19) with full automorphism group order 2, 3, and 8, our results do not agree with those obtained by Colbourn, Magliveras, and Stinson [16]. We are nevertheless confident that our results are the correct ones, because our results for nontrivial automorphism groups can be obtained in two different ways as described in Chapters 4 and 5. Moreover, the total number of distinct STS(19) can be computed in two different ways from the main search data.

The results in this chapter are organized as follows. Section 6.1 gives the total number of nonisomorphic and distinct STS(19). Section 6.2 analyzes the STS(19) with a nontrivial automorphism group and the automorphisms that an STS(19) can admit; the aim is to correct the tables in [16]. Section 6.3 gives the recorded statistics on the number of Pasch configurations in STS(19). From the statistics we obtain the number of anti-Pasch STS(19) and the maximum number of Pasch configurations that an STS(19) can admit. Section 6.4 concludes the chapter with a short discussion of future work

Table 2: The STS(19).

$\lvert\mathrm{Aut}(\mathcal{X})\rvert$	STS(19)	Anti-Pasch
1	11,084,710,071	2,538
2	149,522	1
3	12,728	41
4	2,121	0
6	182	5
8	101	0
9	19	4
12	37	0
16	13	0
18	11	0
19	1	0
24	11	0
32	3	0
54	2	0
57	2	1
96	1	0
108	1	0
144	1	0
171	1	1
432	1	0
Total	11,084,874,829	2,591

on the STS(19) and limitations of the present classification approach.

6.1 A census of the STS(19)

As a result of the main search, we obtain the number of pairwise nonisomorphic STS(19).

Theorem 6.1 *The number of pairwise nonisomorphic* STS(19) *is*

$$N(19) = 11{,}084{,}874{,}829.$$

By Theorem 2.20, this yields the following corollary for the strongly regular block graphs of STS(19).

Corollary 6.2 *There are at least* 11,084,874,829 *pairwise nonisomorphic strongly regular graphs* $\mathrm{srg}(57, 24, 11, 9)$.

Table 2 lists the number of nonisomorphic STS(19) and the number of anti-Pasch STS(19) for each possible full automorphism group order.

The orbit-stabilizer theorem allows us to compute the number of distinct STS(19) from the data in Table 2. Let $\mathcal{X}_1, \ldots, \mathcal{X}_{N(19)}$ be representatives of the isomorphism classes of STS(19). Then, the number of distinct STS(19)

over a fixed point set is

$$\sum_{i=1}^{N(19)} \frac{19!}{|\mathrm{Aut}(\mathcal{X}_i)|}. \tag{14}$$

This yields the following result.

Theorem 6.3 *The number of distinct* STS(19) *is*

$$1{,}348{,}410{,}350{,}618{,}155{,}344{,}199{,}680{,}000.$$

This value can be obtained in an alternative way from the data generated by the main search. Namely, the main search algorithm computes for every seed $\mathcal{S}_i$, $1 \le i \le 14{,}648$, the number of distinct extensions of $\mathcal{S}_i$ to an STS(19). Denote this number by M_i for each seed $\mathcal{S}_i$. Because any STS(19) contains exactly 57 distinct seeds, the orbit-stabilizer theorem gives that

$$\frac{1}{57} \sum_{i=1}^{14{,}648} \frac{19! \cdot M_i}{|\mathrm{Aut}(\mathcal{S}_i)|} \tag{15}$$

is the number of distinct STS(19). (The full automorphism group order for each seed appears in Table 1. Cross-tabulating the values M_i with the values $|\mathrm{Aut}(\mathcal{S}_i)|$ is not feasible here for reasons of space; such a table would require several thousand entries.)

The fact that both (14) and (15) give the same number for distinct STS(19) gives us confidence that the classification is correct. Additional confidence is gained from the fact that our independent classification of STS(19) with a nontrivial full automorphism group gives results identical to those in Table 2 for the number of nonisomorphic STS(19) with a nontrivial full automorphism group.

6.2 Systems with a nontrivial automorphism group

The STS(19) with a nontrivial automorphism group were classified in [16], however, as we noted before, that paper contains a number of errors. In particular, the results for full automorphism groups of order 2 and 3 are incorrect, and the partition of STS(19) into classes in terms of the nonbasic automorphisms that they admit is incorrect for STS(19) with full automorphism group order 8. The aim of this section is to correct these results.

As is shown in [16] (cf. Section 5.1), every STS(19) with a nontrivial automorphism group must admit at least one automorphism of the six *basic* types

$$19^1, \quad 1^1 2^9, \quad 1^1 3^6, \quad 1^3 2^8, \quad 1^7 2^6, \quad 1^7 3^4.$$

For each basic automorphism type, the number of nonisomorphic STS(19) admitting such an automorphism is:

19^1	4
$1^1 2^9$	184
$1^1 3^6$	12,885
$1^3 2^8$	80,645
$1^7 2^6$	72,150
$1^7 3^4$	124

These numbers are inconsistent with those in [16] on automorphism types $1^1 3^6$, $1^3 2^8$, and $1^7 2^6$, where it is erroneously claimed that the respective numbers are 12,021, 80,591, and 80,558. (To prove that the results in [16] cannot be correct, it suffices to check, for example, that the 12,885 STS(19) admitting an automorphism of type $1^1 3^6$ are indeed nonisomorphic and admit an automorphism of the required type.) An electronic listing of all the 164,758 nonisomorphic STS(19) with a nontrivial automorphism group is available from the author upon request.

We now correct the tables in [16]. The basic automorphism structure of STS(19) with a nontrivial automorphism group is summarized in Table 3. The format of Table 3 is identical to [16, Table 1] for ease of reference. The STS(19) are partitioned into classes according to the order of the full automorphism group. Each such class is partitioned further into subclasses according to the types of basic automorphisms that the STS(19) admit. Whenever more than one subclass exists, these are denoted by letters a, b, c, ... For example, the class 4b contains the 662 STS(19) that have full automorphism group order 4 and admit only automorphisms of type $1^3 2^8$ among the basic automorphism types.

In [16] it is also obtained that, in addition to the basic automorphism types, an STS(19) can admit a *nonbasic* automorphism whose type belongs to the list

$$1^1 9^2, \quad 1^1 6^3, \quad 1^1 3^2 6^2, \quad 1^1 2^1 4^4, \quad 1^1 2^1 8^2, \quad 1^3 8^2, \quad 1^3 4^4, \quad 1^3 2^2 6^2, \quad 1^3 2^2 4^3.$$

Together with the basic automorphism types, these are the only types of automorphism an STS(19) can admit.

The classes in Table 3 partition further into subclasses according to the nonbasic automorphisms admitted by an STS(19). This subdivision is given in Table 4. Table 4 is identical to [16, Table 2] with the exception of the class 8a, which partitions into two subclasses instead of the one given in [16]. Namely, among the 84 STS(19) in class 8a there exist two STS(19) whose full automorphism group has order 8 and that admit automorphisms of type $1^3 2^8$, $1^7 2^6$, and $1^3 4^4$. Accordingly, the number of STS(19) that admit an automorphism of type $1^3 4^4$ is 185 instead of the 183 claimed in [16].

6.3 Pasch configurations

Tables 5 to 11 list the number of Pasch configurations in STS(19) for each full automorphism group order. The column "P" in the tables gives the number of Pasch configurations; the column "STS(19)" gives the number of nonisomorphic STS(19) containing this number of Pasch configurations.

From these tables we can extract the number of anti-Pasch STS(19) and the maximum number of Pasch configurations in an STS(19). Previously it was known that there are more than one thousand anti-Pasch STS(19) [66].

Theorem 6.4 *The number of nonisomorphic anti-Pasch* STS(19) *is* 2,591.

The large number of anti-Pasch STS(19) prevents listing them here, however, an electronic listing of the anti-Pasch STS(19) is available from the author upon request.

Table 3: Basic automorphisms.

Order	Class	19^1	1^12^9	1^13^6	1^32^8	1^72^6	1^73^4	STS(19)
432				*	*	*	*	1
171		*		*				1
144				*	*	*		1
108				*	*	*	*	1
96				*	*	*		1
57		*		*				2
54				*		*	*	2
32					*	*		3
24				*	*	*		11
19		*						1
18	a		*	*				1
	b			*	*		*	2
	c			*		*		2
	d			*		*	*	6
16					*	*		13
12	a			*	*			7
	b			*	*	*		8
	c			*		*		12
	d				*	*	*	10
9				*				19
8	a				*	*		84
	b				*			17
6	a		*	*				14
	b			*	*			14
	c			*		*		116
	d				*		*	10
	e					*	*	28
4	a				*	*		839
	b				*			662
	c					*		620
3	a			*				12,664
	b						*	64
2	a		*					169
	b				*			78,961
	c					*		70,392
Total		4	184	12,885	80,645	72,150	124	164,758

Table 4: Nonbasic automorphisms.

Class	$1^1 9^2$	$1^1 6^3$	$1^1 3^2 6^2$	$1^1 2^1 4^4$	$1^1 2^1 8^2$	$1^3 8^2$	$1^3 4^4$	$1^3 2^2 6^2$	$1^3 2^2 4^3$	STS(19)
432			*			*	*	*		1
171	*									1
144			*	*	*		*			1
108			*					*		1
96			*				*			1
57										2
54			*							2
32				*			*			3
24			*							11
19										1
18a		*								1
18b								*		2
18c			*							6
18d			*							2
16				*	*		*			5
16				*						6
16						*	*			1
16							*			1
12a			*							8
12b										7
12c										12
12d								*		10
9	*									9
9										10
8a							*			2
										82
8b				*			*			5
					*		*			10
						*	*			2
6a		*								14
6b										14
6c			*							104
										12
6d								*		10
6e										28
4a										839
4b				*						498
							*			153
										11
4c									*	48
										572
Total	10	15	137	518	16	4	185	24	48	

Table 5: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| = 1$.

P	STS(19)
0	2,538
1	35,742
2	263,580
3	1,314,921
4	4,958,394
5	15,095,241
6	38,479,651
7	84,328,790
8	162,042,722
9	276,885,482
10	426,046,203
11	596,271,490
12	765,950,843
13	910,509,472
14	1,008,606,577
15	1,047,848,142
16	1,027,119,044
17	954,708,823
18	845,586,319
19	716,600,889
20	583,312,837
21	457,752,251

P	STS(19)
22	347,316,148
23	255,585,528
24	182,930,596
25	127,610,069
26	86,994,788
27	58,048,786
28	38,001,524
29	24,453,668
30	15,483,681
31	9660,784
32	5948,963
33	3621,508
34	2183,650
35	1300,661
36	770,041
37	451,540
38	263,545
39	151,688
40	89,084
41	50,804
42	29,632
43	16,852

P	STS(19)
44	10,567
45	5,943
46	3,864
47	2,125
48	1,558
49	715
50	664
51	350
52	316
53	78
54	126
55	68
56	93
57	19
58	56
59	5
60	11
62	17
64	1
66	2
70	2

The maximum number of Pasch configurations in an STS(v) is denoted by $P(v)$. See [80] for a discussion of the function $P(v)$ and [39] for some recent results. For $v \geq 19$ it has been known that $P(19) \geq 84$ with three known designs attaining this value. As a result of the classification, we obtain that $P(19) = 84$.

Theorem 6.5 *The maximum number of Pasch configurations in an* STS(19) *is* 84*; there are three such designs (with automorphism groups of order* 108*,* 144*, and* 432*).*

6.4 Discussion

Clearly, not all the relevant results on STS(19) were produced in this chapter. For example, it is not known whether there exists a weakly 4-chromatic STS(19), nor is it known how many STS(19) admit a subsystem of order 7. The main result of this report is that settling these questions is now possible using exhaustive search, provided that the test for a particular property runs fast enough so that it can be executed for every isomorphism class representative in a reasonable amount of time. For example, one year of CPU time divided uniformly to the $N(19)$ isomorphism class representatives gives $2.84 \cdot 10^{-3}$ seconds of CPU time per representative.

Table 6: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| = 2$.

P	STS(19)
0	1
2	35
4	216
6	794
7	3
8	2,024
9	18
10	4,119
11	63
12	6,506
13	242
14	8,538
15	571
16	9,748
17	1,247

P	STS(19)
18	9,354
19	2,049
20	8,604
21	2,920
22	7,920
23	3,602
24	7,756
25	3,943
26	8,078
27	3,892
28	8,432
29	3,261
30	8,132
31	2,657
32	7,481

P	STS(19)
33	1,751
34	6,277
35	1,247
36	4,994
37	742
38	3,915
39	386
40	2,848
41	203
42	1,814
43	120
44	1,184
45	50
46	687
47	23

P	STS(19)
48	436
49	7
50	261
51	8
52	135
54	121
55	1
56	38
58	28
60	7
62	23
64	1
66	6
70	3

Table 7: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| = 3$.

P	STS(19)
0	41
1	16
2	31
3	240
4	70
5	131
6	602
7	190
8	266
9	1,016
10	350
11	441
12	1,298
13	404
14	556

P	STS(19)
15	1,306
16	416
17	528
18	987
19	352
20	421
21	694
22	224
23	296
24	412
25	156
26	200
27	251
28	86
29	135

P	STS(19)
30	148
31	54
32	67
33	76
34	36
35	43
36	40
37	18
38	18
39	27
40	6
41	11
42	14
43	2
44	6

P	STS(19)
45	4
46	4
47	3
48	16
49	2
50	1
51	1
52	5
54	2
57	1
58	1
59	1
60	5

Table 8: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| = 4$.

P	STS(19)
4	7
8	41
10	1
11	3
12	91
13	5
14	1
16	123
17	11
19	3

P	STS(19)
20	114
21	18
22	14
23	1
24	124
25	11
26	45
28	159
29	9
30	148

P	STS(19)
32	201
33	7
34	198
36	138
37	2
38	154
40	111
41	1
42	96
44	60

P	STS(19)
46	54
48	52
50	51
52	19
54	18
56	4
58	13
60	8
62	5

Table 9: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| = 6$.

P	STS(19)
0	5
6	3
12	3
13	1
15	9
18	4
19	5

P	STS(19)
21	14
24	8
25	4
27	12
30	2
31	4
33	14

P	STS(19)
34	3
36	11
37	4
39	15
42	5
45	11
46	2

P	STS(19)
48	22
51	7
54	1
58	4
60	1
66	6
78	2

Table 10: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| = 8$.

P	STS(19)
8	1
16	4
24	3
34	2
36	5

P	STS(19)
38	6
40	3
42	20
44	4
46	18

P	STS(19)
48	4
50	12
52	5
54	8
56	2

P	STS(19)
58	2
62	2

Table 11: Pasch configurations in STS(19) with $|\mathrm{Aut}(\mathcal{X})| \geq 9$.

$|\mathrm{Aut}(\mathcal{X})| = 9$

P	STS(19)
0	4
9	2
15	4
18	4
21	1
27	1
36	2
48	1

$|\mathrm{Aut}(\mathcal{X})| = 12$

P	STS(19)
7	1
14	1
15	1
18	2
22	1
23	1
26	4
30	1
38	3
42	2
48	8
50	2
54	2
60	4
66	4

$|\mathrm{Aut}(\mathcal{X})| = 16$

P	STS(19)
36	2
44	4
52	2
60	5

$|\mathrm{Aut}(\mathcal{X})| = 18$

P	STS(19)
18	1
39	6
48	2
57	2

$|\mathrm{Aut}(\mathcal{X})| = 19$

P	STS(19)
19	1

$|\mathrm{Aut}(\mathcal{X})| = 24$

P	STS(19)
30	2
40	4
42	4
64	1

$|\mathrm{Aut}(\mathcal{X})| = 32$

P	STS(19)
28	1
44	2

$|\mathrm{Aut}(\mathcal{X})| = 54$

P	STS(19)
57	2

$|\mathrm{Aut}(\mathcal{X})| = 57$

P	STS(19)
0	1
38	1

$|\mathrm{Aut}(\mathcal{X})| = 96$

P	STS(19)
28	1

$|\mathrm{Aut}(\mathcal{X})| = 108$

P	STS(19)
84	1

$|\mathrm{Aut}(\mathcal{X})| = 144$

P	STS(19)
84	1

$|\mathrm{Aut}(\mathcal{X})| = 171$

P	STS(19)
0	1

$|\mathrm{Aut}(\mathcal{X})| = 432$

P	STS(19)
84	1

Naturally the most complete classification of the STS(19) one could hope for would be to catalogue them all on a storage device during the main search, which would then enable the subsequent study of any property of interest by searching the catalogue. As a result of the present work, such a catalogue was prepared for the anti-Pasch STS(19) and the STS(19) with a nontrivial automorphism group. The complete catalogue of all the STS(19) would require disk space in the order of tens or hundreds of gigabytes, which is impractical at least for the time being. (The design of an efficient compression scheme for STS(v) could be a topic of further research.) Meanwhile, it is always possible to rerun the main search provided that sufficient computational resources are available.

Given that a complete classification of the STS(19) can now be produced within a manageable amount of time, the next obvious question is whether the next open case, $v = 21$, can be classified using the approach described in Chapter 4. Unfortunately, the answer seems to be no. For the STS(21), there are 219,104 nonisomorphic 28-block seeds. The running time of the main search algorithm on a few randomly selected seeds was estimated to be more than one year for each seed on a Linux workstation with a 450-MHz Pentium II CPU. A complete classification with the present approach seems thus to require CPU time in the order of hundreds of thousands of years, which is clearly infeasible. The time required for a classification of the STS(21) with a nontrivial automorphism group was not estimated; this remains a topic for future work.

7 CONSTRUCTION OF OTHER STRUCTURES

The classification framework described in Chapter 4 for STS(19) can be used essentially without change to classify certain other combinatorial objects that admit representation up to isomorphism as incidence structures. In this chapter we study some examples of such objects.

For the framework to be directly applicable, the incidence structure representation must have the following four properties:

(C1) The construction problem for a structure can be formulated as an exact cover problem, in which the blocks are used to cover certain subsets of points.

(C2) A *seed* is the structure induced by a block and all blocks that intersect the block. The block that induces a seed is unique, that is, no two blocks intersect the same set of blocks.

(C3) The block automorphism group of a structure is equal to the automorphism group of the block graph.

(C4) Two structures are isomorphic if their block graphs are isomorphic.

Property (C1) guarantees that the exact cover algorithm can be used for exhaustive generation.

Properties (C2) and (C3) guarantee that the parent test and the automorphism test in Section 4.4 work as intended. Indeed, a careful reading of

Sections 4.4 and 4.4 shows that Properties (C2) and (C3), the latter in the form of Theorem 2.19, are the only properties that are required for the correct operation of the tests if Requirement L holds. Note that Property (C3) is required only when block graphs are used for isomorph rejection.

Property (C4) is required by isomorph rejection with canonical representative block graphs and by the Pasch invariant in Section 4.4. The Pasch invariant naturally also requires a constant block size of three. In this chapter we restrict to structures with block size three.

More specifically, we consider three types of objects: one-factorizations of the complete graph K_{2n} [1, 84] and two types of 3-GDDs: latin squares [15] and generalized Steiner triple systems [26]. For each of the objects considered, we will perform the following. First, we derive a representation for the objects using incidence structures. Then, we establish that Properties (C1)–(C4) hold for the representation. This is the content of Sections 7.1 and 7.2.

The material in this chapter is to a large degree a description of work in progress. Initial estimates, presented in Section 7.3, suggest that the classification framework is likely to perform well when compared with existing algorithms [23] on classifying one-factorizations of the complete graph K_{2n}. Estimates have not yet been computed for the other two types of structures considered.

7.1 One-factorizations of K_{2n}

We first derive an incidence structure representation for one-factorizations of K_{2n}. Let $n \geq 1$ and let $\mathcal{F}$ be a one-factorization of K_{2n}. Clearly, $\mathcal{F}$ contains $2n-1$ one-factors $F_1, \ldots, F_{2n-1}$, each of which consists of n edges. Suppose the vertex set of K_{2n} consists of vertices $v_1, \ldots, v_{2n}$.

We represent $\mathcal{F}$ using an incidence structure of the following form. The point set of the incidence structure consists of the vertices $v_1, \ldots, v_{2n}$ and $2n-1$ additional points $f_1, \ldots, f_{2n-1}$, one point for each one-factor. Each block of the incidence structure is associated with an edge of K_{2n}. Namely, for each edge $\{v_i, v_j\}$, the incidence structure contains a block incident with the points $\{f_k, v_i, v_j\}$, where k is determined from $\{v_i, v_j\} \in F_k$.

It is straightforward to check that two one-factorizations of K_{2n} are isomorphic if and only if their incidence structure representations are isomorphic.

The exhaustive construction of one-factorizations of K_{2n} can be formulated as an exact cover problem in which the task is to cover once all $\binom{2n}{2} + 2n(2n-1) = 3n(2n-1)$ pairs of the form

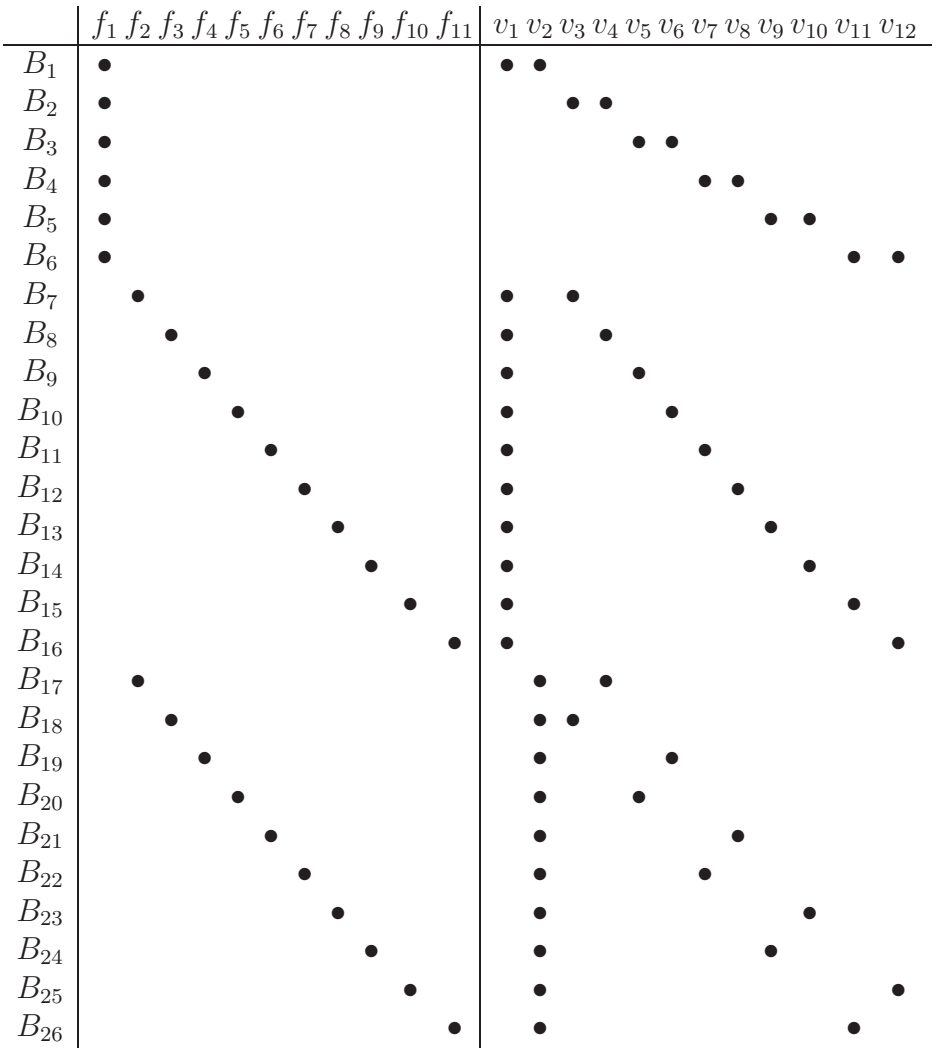
$$\{v_i, v_j\}, \quad 1 \leq i < j \leq 2n; \qquad \{f_i, v_j\}, \quad 1 \leq i \leq 2n-1,\ 1 \leq j \leq 2n$$

using 3-subsets of the form $\{f_k, v_i, v_j\}$, $1 \leq i < j \leq 2n$, $1 \leq k \leq 2n-1$. The solutions of this problem are precisely the incidence structure representations of one-factorizations of K_{2n}. This establishes Property (C1).

A seed is the structure induced by a block $\{f_k, v_i, v_j\}$ and all blocks that intersect it. There are n blocks that contain f_k, $2n-1$ blocks that contain v_i, and $2n-1$ blocks that contain v_j. Thus, a seed consists of $1 + (n-1) + 2(2n-2) = 5n-4$ blocks. The block that induces a seed is easily seen to

be unique for $n \geq 3$. (For $n \leq 2$ all blocks induce the same seed.) Thus, Property (C2) holds for $n \geq 3$. An example of a seed for $n = 6$ is given below.

Example 7.1 A seed for the one-factorizations of K_{12}.



We study next the block graphs of (incidence structure representations of) one-factorizations of K_{2n}. As was done in Section 2.2 for block graphs of STS(v), we establish Properties (C3) and (C4) for a block graph of a one-factorization by studying the maximum cliques in the graph. Since each pair of distinct points is incident with at most one block, we obtain from the proof of Lemma 2.18 that a set of pairwise intersecting blocks with no common point has cardinality at most 7. This can be improved to at most 6 because no point f_i is incident with more than two blocks in such a set. Hence, a sub-STS(7) cannot occur. However, a set of six blocks is possible:

$$\{f_1, v_1, v_2\}, \{f_1, v_3, v_4\}, \{f_2, v_1, v_3\}, \{f_2, v_2, v_4\}, \{f_3, v_1, v_4\}, \{f_3, v_2, v_3\}.$$

Such a set corresponds to a one-factorization of K_4.

Since each pair of points is incident with at most one block, every edge in the block graph can be associated with the unique point common to the two blocks. Thus, a point v_j can be associated with a clique of order $2n-1$ in the block graph. Similarly, a point f_i can be associated with a clique of order n.

By the upper bound just derived, other maximal cliques in the block graph have order at most 6. When $2n-1 > 6$, that is, $n \geq 4$, the cliques associated with the points v_j are maximum and hence can be easily found from a block graph.

For $n \geq 4$, a one-factorization can be recovered up to isomorphism from a block graph as follows. Label the edges of the maximum cliques in the block graph arbitrarily with the points $v_1, \ldots, v_{2n}$. For each pair of distinct points v_i, v_j, locate the block common to the corresponding maximum cliques in the block graph. This block is adjacent to $n-1$ other blocks that do not belong to either of the maximum cliques. These n blocks induce a clique in the block graph that corresponds to a point f_k. If the edges of this clique are unlabelled, label all the edges in this clique with f_k. Continue with the next pair v_i, v_j until all edges of the block graph have been labelled. Finally, reconstruct the blocks of the incidence structure by setting each block to be incident with the three edge labels that occur in the incident edges.

Since any automorphism of a block graph must permute the maximum cliques among themselves, the previous argument also shows that the block automorphism group and the automorphism group of the block graph are equal for $n \geq 4$. Thus, Properties (C3) and (C4) hold for $n \geq 4$.

It is interesting to note that Pasch configurations have a natural interpretation in terms of one-factorizations. Namely, a Pasch configuration in an incidence structure representation of a one-factorization corresponds to two one-factors and four edges, two from each one-factor, that form a 4-cycle. For example, the blocks B_1, B_2, B_7, and B_{17} in Example 7.1 induce a Pasch configuration. Anti-Pasch one-factorizations are thus equivalent to one-factorizations in which no pair of one-factors contains a 4-cycle.

7.2 Group divisible designs with block size 3

For simplicity, we will consider uniform group divisible designs only.

Consider a 3-GDD of group type t^u. The exhaustive construction of such designs can clearly be formulated as an exact cover problem, in which the task is to cover once all distinct pairs of points from different groups using 3-subsets of points such that no two points in a 3-subset are in the same group. This establishes Property (C1).

It is straightforward (cf. Theorem 2.7) to see that in a 3-GDD of type t^u, each point must occur in $r = t(u-1)/2$ blocks and that the number of blocks is $b = t^2u(u-1)/6$.

Since each pair of points occurs in at most one block, the number of blocks that intersect a block of a uniform 3-GDD is $1 + 3(r-1) = 3r-2$. This is the cardinality of a seed. The block that induces a seed is unique for $r > 3$. To see this, observe that any two blocks that intersect in a point p can intersect at most 4 common blocks that do not contain p. The blocks must intersect $2(r-1)$ common blocks that do not contain p so that they induce the same seed. Thus, Property (C2) holds for $r \geq 4$.

We next analyze the block graphs of uniform 3-GDDs. Let M be the maximum cardinality of a set of pairwise intersecting blocks that do not have a common point. The proof of Lemma 2.18 gives $M \leq 7$ with equality if and only if the blocks form a sub-STS(7), which requires $u \geq 7$.

Each point of a uniform 3-GDD forms a clique of order r in the block graph. For $r > M$, these $v = tu$ cliques are the only maximum cliques in the block graph. Hence, a uniform 3-GDD can be reconstructed from its block graph by labelling the maximum cliques arbitrarily with v points and setting each block to be incident with the labels of the three maximum cliques to which it belongs. Furthermore, the group partition $\mathcal{G}$ can be reconstructed using the observation that two maximum cliques in the block graph are vertex-disjoint if and only if the corresponding points are in the same group. An automorphism of the block graph must permute the maximum cliques among themselves. Hence, any automorphism of the block graph can be extended to an automorphism of the underlying 3-GDD using the permutation induced to the maximum cliques (cf. the proof of Theorem 2.19). This establishes properties (C3) and (C4) for $r > M$.

In the following we discuss two examples of 3-GDDs: latin squares and generalized Steiner triple systems.

Latin squares

A *latin square* of order n is a quadruple (R, C, S, L), where R, C, and S are (pairwise disjoint) sets of cardinality n and L is a mapping $L : R \times C \to S$ such that for any two of $i \in R$, $j \in C$, $x \in S$, the equation $L(i, j) = x$ has a unique solution. Elements of R are called *rows*, elements of C are called *columns*, and elements of S are called *symbols* or *entries* of the latin square.

Latin squares are equivalent to many other combinatorial objects (see [15]), among them transversal designs $\mathrm{TD}(3, n)$. A latin square of order n defines a transversal design $\mathrm{TD}(3, n)$ (that is, a 3-GDD of type n^3), where the groups are the sets R, C, S, and the blocks are all triples of the form $\{i, j, x\}$, where $i \in R$, $j \in C$, $x \in S$ such that $L(i, j) = x$.

Conversely, a $\mathrm{TD}(3, n)$ defines a latin square of order n: label the three groups of the GDD arbitrarily as R, C, S, and define the function L by setting $L(i, j) = x$ if and only if $\{i, j, x\}$ is a block of the TD. Since each pair of points from distinct groups occurs in a unique block, L is well-defined and satisfies the required property. (Note that there are six possibilities to label the three groups as R, C, S, so each $\mathrm{TD}(3, n)$ defines six latin squares.)

Two latin squares are called *main class isotopic* if the corresponding transversal designs are isomorphic.

The block graph of a $\mathrm{TD}(3, n)$ is a strongly regular graph $\mathrm{srg}(n^2, 3(n - 1), n - 2, 6)$. A short case-by-case analysis shows that the maximum cardinality of a set of pairwise intersecting blocks with no common point in a $\mathrm{TD}(3, n)$ is $M = 4$, which occurs precisely when the four blocks induce a Pasch configuration.

It is again interesting to note that a Pasch configuration is a $\mathrm{TD}(3, 2)$, that is, a latin square of order 2. Thus, an anti-Pasch $\mathrm{TD}(3, n)$ corresponds to a latin square with no subsquares of order 2. A latin square with no subsquares of order 2 is called an N_2 square [15].

Generalized Steiner triple systems

Generalized Steiner (triple) systems were introduced by Etzion [26].

We require first some standard definitions. Let $Z_q = \{0, 1, \ldots, q-1\}$ and denote by Z_q^n the set of all ordered n-tuples $x = x_1 x_2 \cdots x_n$ (words) over Z_q.

Table 12: The seeds for one-factorizations of K_{12}.

$\lvert\mathrm{Aut}(\mathcal{S})\rvert$	Seeds
1	66
2	166
4	78
8	24
10	3
12	1
16	14
20	4
24	6

$\lvert\mathrm{Aut}(\mathcal{S})\rvert$	Seeds
32	11
48	9
64	1
96	4
128	2
192	1
384	2
7,680	1
Total	393

The *(Hamming) distance* between two codewords $x, y \in Z_q^n$ is $d_H(x, y) = |\{i : x_i \neq y_i\}|$, that is, the number of coordinate positions in which x and y differ. The *(Hamming) weight* of a codeword $x \in Z_q^n$ is $w_H(x) = |\{i : x_i \neq 0\}|$, that is, the number of nonzero coordinates in the codeword or, equivalently, the distance from the all-zero codeword. A *code* of length n over Z_q is a nonempty subset of Z_q^n. The *minimum distance* of a code is the minimum Hamming distance between pairs of distinct codewords.

A 3-GDD of type t^n can be viewed as a code of length n and constant weight 3 over the alphabet Z_q, $q = t + 1$. Namely, we identify the elements of each of the groups $G_1, \ldots, G_n$ in the GDD arbitrarily with the nonzero elements of Z_q. Now, each block of the GDD defines a codeword $x = x_1 \cdots x_n$ so that $x_i = j$ if the block is incident with the point identified with $j \in Z_q \setminus \{0\}$ in G_i; and $x_i = 0$ otherwise, that is, when no point from G_i is incident with the block.

A 3-GDD of type t^n is a *generalized Steiner triple system* if the minimum distance of the corresponding code is 3.

In this case $M \leq 7$, where equality requires $n \geq 7$. A Pasch configuration does not seem to have an immediate natural interpretation.

7.3 A performance estimate

We conclude this chapter by estimating the performance of the classification algorithm in Chapter 4 in classifying the nonisomorphic one-factorizations K_{2n} for $n = 6$, the largest value of n for which a complete classification is known (see [1, 23]).

First, we classify the 26-block seeds using block by block backtrack search with an isomorph rejection step after each added block. In total, there are 393 nonisorphic seeds, which are obtained in less than two minutes of CPU time on a Linux workstation with a 450-MHz Pentium II CPU. The full automorphism group order for each seed is listed in Table 12.

We estimate the CPU time required for a complete classification by running the main search with stochastic pruning. In describing the estimation technique it is convenient to view the entire search as forming a tree. The nodes of the tree are the partial incidence structures considered in the search;

Table 13: CPU time estimates for the main search on one-factorizations of K_{12}.

p	S	T (s)	Estimate
0.20	$\{29, 32, 35\}$	32,440	47 days
0.27	$\{36, 37, 38\}$	77,840	46 days
0.20	$\{39, 40, 41\}$	32,390	47 days
0.50	$\{35, 36, \ldots, 40\}$	61,790	46 days

a node $\mathcal{X}$ is a *child* of a node $\mathcal{Y}$ if $\mathcal{X}$ can be obtained from $\mathcal{Y}$ by addition of one block. A node is on *level* ℓ in the search tree if it consists of ℓ blocks. For example, on level 26 of the tree there are 393 nodes, which correspond to the seeds. Let $S \subseteq \{27, \ldots, 66\}$ be a set of levels on which stochastic pruning is performed and let $0 \leq p \leq 1$ be the probability of accepting a node. Whenever the search reaches a node on one of the levels $\ell \in S$, we flip a biased coin that has probability p of coming up heads and probability $1 - p$ of coming up tails. If the coin comes up heads, then we accept the node and continue the search to the children of the node; otherwise we reject the node and return to the previous level in the search.

In this way we obtain a crude estimate of the time required to run the main search, assuming that the search tree is sufficiently uniform. If T is the time required by the main search with stochastic pruning, then $T/p^{|S|}$ is our estimate for the CPU time requirement of the main search. Note that systematic bias is produced by the fact that the search tree is completely traversed up to the pruning levels during estimation. Thus, the expected fraction of the search tree that is traversed is strictly more than $p^{|S|}$, even when the search tree is completely uniform. (For a more thorough discussion on estimating the efficiency of backtrack algorithms, see [51, 68].)

We conducted four estimation runs, each with a different selection of p and S. The results of the estimation appear in Table 13. All of the estimates were performed on a Linux workstation with a 450-MHz Pentium II CPU. The pseudorandom number generator used in estimation was the Linux standard library function `drand48(3)` initialized with the system time. Based on the estimations in Table 13, it would seem that a classification of the nonisomorphic one-factorizations of K_{12} could be obtained in less than 50 days using the algorithm of Chapter 4.

In comparison, the algorithm used by Dinitz, Garnick, and McKay [23] required a little over 160 MIPS-years (160 years on a computer running at 1 MIPS) to classify the nonisomorphic one-factorizations of K_{12}. Assuming a processor speed of 450 MIPS for the computer used in the estimation, the estimates indicate that a classification could be obtained in less than $50/365 \cdot 450 \approx 62$ MIPS-years with the classification algorithm of Chapter 4. Thus, with an estimated performance improvement of factor 2.5 over an earlier algorithm, the algorithm of Chapter 4 seems promising for use in the classification of other structures as well. The first task is to conduct an independent verification of the classification in [23].

ACKNOWLEDGMENTS

This is the report version of my Licentiate's Thesis [45]. My thesis work started in fall 2001, when Professor Patric Östergård suggested to me an approach for classifying Steiner triple systems with an exact cover algorithm. This approach turned out to be successful, and in December 2001 we had a complete classification of the Steiner triple systems of order 19 [48]. This thesis describes this classification approach and its extension to the classification of other structures.

There are a number of people that I would like to thank for assisting me in my thesis work. First and foremost, I wish to thank my instructor, Professor Patric Östergård, for his guidance and comments. I thank Professor Pekka Orponen for supervising and commenting upon this work, Professor Ilkka Niemelä for providing the supportive research environment at the Laboratory for Theoretical Computer Science, and Harri Haanpää and Tommi Junttila for rewarding discussions. Furthermore, I wish to thank Juhani Markula and Jukka Seppänen at the Computing Centre of Helsinki University of Technology for providing the computing resources without which this work would not have been possible.

The financial support of Helsinki Graduate School in Computer Science and Engineering (HeCSE), the Academy of Finland under Grant No. 47754, and the Foundation of Technology, Helsinki, Finland (Tekniikan Edistämisäätiö) is gratefully acknowledged.

REFERENCES

[1] L. D. Andersen. Factorizations of graphs. In *The CRC Handbook of Combinatorial Designs* (C. J. Colbourn and J. H. Dinitz, eds.), CRC Press, Boca Raton FL, 1996, pp. 653–667.

[2] L. Babai, Almost all Steiner triple systems are asymmetric, *Ann. Discrete Math.* **7** (1980) 37–39.

[3] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proc. 15th ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York NY, 1983, pp. 171–183.

[4] S. Bays, Sur les systèmes cycliques de triples de Steiner différents pour n premier de la forme $6n + 1$, *Comment. Math. Helv.* **4** (1932) 183–194.

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

[6] K. S. Booth and C. J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, Department of Computer Science, University of Waterloo, Ontario, Canada, 1979.

[7] R. C. Bose, Strongly regular graphs, partial geometries and partially balanced designs, *Pacific J. Math.* **13** (1963) 389–419.

[8] G. Brinkmann. Isomorphism rejection in structure generation programs. In *Discrete Mathematical Chemistry (Proc. DIMACS Workshop held at Rutgers University, New Brunswick NJ, March 23–25, 1998)*, Amer. Math. Soc., Providence RI, 2000, pp. 25–38.

[9] R. H. Bruck, Finite nets II. Uniqueness and imbedding, *Pacific J. Math.* **13** (1963) 421–457.

[10] G. Brunel, Sur les deux systèmes de triades de trieze elements, *J. Math. Soc. Sci. Phys. Nat. Bordeaux (6)* **2** (1902) 1–23.

[11] G. Butler. *Fundamental Algorithms for Permutation Groups*. Springer-Verlag, New York NY, 1991.

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

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

[14] C. J. Colbourn and J. H. Dinitz, eds. *The CRC Handbook of Combinatorial Designs*. CRC Press, Boca Raton FL, 1996.

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

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

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

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

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

[20] F. N. Cole, L. D. Cummings, and H. S. White, The complete enumeration of triad systems in 15 elements, *Proc. Natl. Acad. Sci. USA* **3** (1917) 197–199.

[21] V. De Pasquale, Sui sistemi ternari di 13 elementi, *Rend. R. Ist. Lombardo Sci. Lett. (2)* **32** (1899) 213–221.

[22] R. H. F. Denniston, Nonisomorphic reverse Steiner triple systems of order 19, *Ann. Discrete Math.* **7** (1980) 255–264.

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

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

[25] G. P. Egorychev, Proof of the van der Waerden conjecture for permanents, *Sibirsk. Mat. Zh.* **22** (1981) no. 6 , 65–71, 225. In Russian. English translation: The solution of van der Waerden's problem for permanents, *Adv. in Math.* **42** (1981) 299–305.

[26] T. Etzion, Optimal constant weight codes over Z_k and generalized designs, *Discrete Math.* **169** (1997) 55–82.

[27] D. I. Falikman, Proof of the van der Waerden conjecture on the permanent of a doubly stochastic matrix, *Mat. Zametki* **29** (1981) 931–938, 957. In Russian. English translation: Proof of the van der Waerden conjecture on the permanent of a doubly stochastic matrix, *Math. Notes* **29** (1981) 475–479.

[28] I. A. Faradžev. Constructive enumeration of combinatorial objects. In *Problèmes Combinatoires et Théorie des Graphes Colloque Internat. CNRS No. 260*, CNRS, Paris, France, 1978, pp. 131–135.

[29] L. Finkelstein and W. M. Kantor, eds. *Groups and Computation (Proc. DIMACS Workshop held at Rutgers University, New Brunswick NJ, October 7–10, 1991)*, Amer. Math. Soc., Providence RI, 1993.

[30] L. Finkelstein and W. M. Kantor, eds. *Groups and Computation, II (Proc. 2nd DIMACS Workshop held at Rutgers University, New Brunswick NJ, June 7–10, 1995)*, Amer. Math. Soc., Providence RI, 1997.

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

[32] P. B. Gibbons. *Computing Techniques for the Construction and Analysis of Block Designs.* PhD thesis, University of Toronto, Ontario, Canada, 1976.

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

[34] L. A. Goldberg, Efficient algorithms for listing unlabeled graphs, *J. Algorithms* **13** (1992) 128–143.

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

[36] S. W. Golomb and L. D. Baumert, Backtrack programming, *J. ACM* **12** (1965) 516–524.

[37] M. J. Grannell, T. S. Griggs, and E. A. Mendelsohn, A small basis for four-line configurations in Steiner triple systems, *J. Combin. Des.* **3** (1995) 51–59.

[38] M. J. Grannell, T. S. Griggs, and C. A. Whitehead, The resolution of the anti-Pasch conjecture, *J. Combin. Des.* 8 (2000) 300–309.

[39] B. D. Gray and C. Ramsay, On the number of Pasch configurations in a Steiner triple system, *Bull. Inst. Combin. Appl.* **24** (1998) 105–112.

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

[41] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism.* Springer-Verlag, New York NY, 1982.

[42] J. F. Humphreys. *A Course in Group Theory.* Oxford University Press, Oxford, England, 1996.

[43] M. Jerrum, A compact representation for permutation groups, *J. Algorithms* **7** (1986) 60–78.

[44] W. M. Kantor and A. Seress, eds. *Groups and Computation, III (Proc. 3rd Int'l Conference held at The Ohio State University, Columbus OH, June 15–19, 1999)*, Walter de Gruyter & Co., New York NY, 2001.

[45] P. Kaski. *A Census of Steiner Triple Systems and Some Related Combinatorial Objects.* Licentiate's thesis, Department of Computer Science and Engineering, Helsinki University of Technology, 2002.

[46] P. Kaski. Isomorph-free exhaustive generation of combinatorial designs. Laboratory for Theoretical Computer Science Research Report HUT-TCS-A70, Department of Computer Science and Engineering, Helsinki University of Technology, Finland, 2002.

[47] P. Kaski and P. R. J. Östergård, There exist nonisomorphic STS(19) with equivalent point codes, in preparation.

[48] P. Kaski and P. R. J. Östergård, The Steiner triple systems of order 19, *Math. Comp.*, to appear.

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

[50] T. P. Kirkman, On a problem in combinations, *Cambridge and Dublin Math. J.* **2** (1847) 191–204.

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

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

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

[54] W. Kocay. On writing isomorphism programs. In *Computational and Constructive Design Theory* (W. Wallis, ed.), Kluwer, Dordrecht, The Netherlands, 1996, pp. 135–175.

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

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

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

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

[59] 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, eds.), CRC Press, Boca Raton FL, 1996, pp. 3–41.

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

[61] B. D. McKay. *nauty* User's Guide (version 1.5). Tech. Rep. TR-CS-90-02, Computer Science Department, Australian National University, Canberra, Australia, 1990.

[62] B. D. McKay. autoson – a distributed batch system for UNIX workstation networks (version 1.3). Tech. Rep. TR-CS-96-03, Computer Science Department, Australian National University, Canberra, Australia, 1996.

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

[64] G. L. Miller. On the $n^{\log n}$ isomorphism technique. In *Proc. 10th ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York NY, 1978, pp. 51–58.

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

[66] J. P. Murphy. *Steiner Triple Systems and Cycle Structure*. PhD thesis, University of Central Lancashire, Preston, England, 1999.

[67] K. T. Phelps and A. Rosa, Steiner triple systems with rotational automorphisms, *Discrete Math.* **33** (1981) 57–66.

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

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

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

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

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

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

[74] D. A. Spielman. Faster isomorphism testing of strongly regular graphs. In *Proc. 28th ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York NY, 1996, pp. 576–584.

[75] D. R. Stinson, A comparison of two invariants for Steiner triple systems: fragments and trains, *Ars. Combin.* **16** (1983) 69–76.

[76] D. R. Stinson, Hill-climbing algorithms for the construction of combinatorial designs, *Ann. Discrete Math.* **26** (1985) 321–334.

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

[78] D. R. Stinson and H. Ferch, 2 000 000 Steiner triple systems of order 19, *Math. Comp.* **44** (1985) 533–535.

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

[80] D. R. Stinson and Y. J. Wei, Some results on quadrilaterals in Steiner triple systems, *Discrete Math.* **103** (1992) 207–219.

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

[82] V. D. Tonchev and R. S. Weishaar, Steiner triple systems of order 15 and their codes, *J. Statist. Plann. Inference* **58** (1997) 207–216.

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

[84] W. D. Wallis. *One-Factorizations.* Kluwer, Dordrecht, The Netherlands, 1997.

[85] H. S. White, F. N. Cole, and L. D. Cummings, Complete classification of the triad systems on fifteen elements, *Memoirs of the National Academy of Sciences U.S.A.* **14** (1919) 1–89.

[86] R. M. Wilson, Nonisomorphic Steiner triple systems, *Math. Z.* **135** (1974) 303–313.

[87] K. Zulauf. Über Tripelsysteme von 13 Elementen. Dissertation Giessen, Wintersche Buchdruckerei, Darmstadt, Germany, 1897.

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE
RESEARCH REPORTS

HUT-TCS-A65 Harri Haanpää
Computational Methods for Ramsey Numbers. November 2000.

HUT-TCS-A66 Heikki Tauriainen
Automated Testing of Büchi Automata Translators for Linear Temporal Logic.
December 2000.

HUT-TCS-A67 Timo Latvala
Model Checking Linear Temporal Logic Properties of Petri Nets with Fairness Constraints.
January 2001.

HUT-TCS-A68 Javier Esparza, Keijo Heljanko
Implementing LTL Model Checking with Net Unfoldings. March 2001.

HUT-TCS-A69 Marko Mäkelä
A Reachability Analyser for Algebraic System Nets. June 2001.

HUT-TCS-A70 Petteri Kaski
Isomorph-Free Exhaustive Generation of Combinatorial Designs. December 2001.

HUT-TCS-A71 Keijo Heljanko
Combining Symbolic and Partial Order Methods for Model Checking 1-Safe Petri Nets.
February 2002.

HUT-TCS-A72 Tommi Junttila
Symmetry Reduction Algorithms for Data Symmetries. May 2002.

HUT-TCS-A73 Toni Jussila
Bounded Model Checking for Verifying Concurrent Programs. August 2002.

HUT-TCS-A74 Sam Sandqvist
Aspects of Modelling and Simulation of Genetic Algorithms: A Formal Approach.
September 2002.

HUT-TCS-A75 Tommi Junttila
New Canonical Representative Marking Algorithms for Place/Transition-Nets. October 2002.

HUT-TCS-A76 Timo Latvala
On Model Checking Safety Properties. December 2002.

HUT-TCS-A77 Satu Virtanen
Properties of Nonuniform Random Graph Models. May 2003.

HUT-TCS-A78 Petteri Kaski
A Census of Steiner Triple Systems and Some Related Combinatorial Objects. June 2003.

ISBN 951-22-6645-8
ISSN 1457-7615