

Generating Uniformly-Distributed Random
Generalised 2-designs with Block Size 3

School of Mathematical Sciences, Queen Mary University of London

*A thesis submitted in partial fulfillment of the requirements of the Degree of
Doctor of Philosophy*

Andy L. Drizen

December 2012

Abstract

Generalised t -designs, defined by Cameron, describe a generalisation of many combinatorial objects including: Latin squares, 1-factorisations of K_{2n} (the complete graph on $2n$ vertices), and classical t -designs.

This new relationship raises the question of how their respective theory would fare in a more general setting. In 1991, Jacobson and Matthews published an algorithm for generating uniformly distributed random Latin squares and Cameron conjectures that this work extends to other generalised 2-designs with block size 3.

In this thesis, we divide Cameron's conjecture into three parts. Firstly, for constants λ_{RC} , λ_{RS} and λ_{CS} , we study a generalisation of Latin squares, which are $(r \times c)$ grids whose cells each contain λ_{RC} symbols from the set $\{1, 2, \dots, s\}$ such that each symbol occurs λ_{RS} times in each column and λ_{CS} times in each row. We give fundamental theory about these objects, including an enumeration for small parameter values. Further, we prove that Cameron's conjecture is true for these designs, for all admissible parameter values, which provides the first method for generating them uniformly at random.

Secondly, we look at a generalisation of 1-factorisations of the complete graph. For constants λ_{NN} and λ_{NC} , these graphs have n vertices, each incident with λ_{NN} coloured edges, such that each colour appears at each vertex λ_{NC}

times. We successfully show how to generate these designs uniformly at random when $\lambda_{NC} \equiv 0 \pmod{2}$ and $\lambda_{NN} \geq \lambda_{NC}$.

Finally, we observe the difficulties that arise when trying to apply Jacobson and Matthews' theory to the classical triple systems. Cameron's conjecture remains open for these designs, however, there is mounting evidence which suggests an affirmative result.

A function reference for DesignMC, the bespoke software that was used during this research, is provided in an appendix.

Acknowledgements

I am grateful to Queen Mary University of London for having such a vibrant and high-calibre mathematics department. With international travel opportunities, discussions with academics who perform cutting-edge research in their fields, and a social environment which breeds collaboration, Queen Mary has been a good home. In particular, Peter Cameron, Leonard Soicher, Mark Jerrum and my fellow PhD students have been invaluable sources of knowledge, confidence, inspiration and fun.

I am indebted to my secondary school teacher, Liz Rabett, my mother, Kate, and my wife, Anna, for getting me there.

Andy Drizen

London, 2012

Contents

1	Introduction	1
1.1	An Overview	1
1.2	Software Used	5
1.3	Original Content	7
2	Generalised t-designs	9
3	Generating Latin Squares Uniformly at Random	15
3.1	Latin Squares	15
3.1.1	Counting Latin Squares	16
3.1.2	Generating Latin Squares	17
3.2	Markov Chains	22
3.2.1	Example: Random Walk	24
3.3	Jacobson and Matthews' Technique	26
3.3.1	Pair Graphs	32
3.3.2	Proving Connectedness	39
4	Generalised Latin squares	47
4.1	Definition	47
4.1.1	Generalised Squares	49
4.2	Pair Graphs	52
4.2.1	Square Row-Pair Graph Analysis	53

4.3	Generating Squares Uniformly at Random	55
5	Generalised Factorisations	61
5.1	Definition	61
5.1.1	Counting 1-factorisations	62
5.1.2	Generalised Factorisations	63
5.1.3	Generalised λ_{NC} -factorisation of $\lambda_{NN}K_n$	63
5.2	Pair Graphs	65
5.3	Generating Factorisations Uniformly at Random	71
5.4	Experimental Results	74
6	Generalised Triple Systems	77
6.1	Definition	77
6.1.1	Counting Triple Systems	79
6.2	Pair Graphs	81
6.3	Evidence Supporting the ± 1 -move	82
7	Decomposing Latin Rectangles	87
7.1	Decomposing Latin Rectangles	89
8	Conclusion	95
Appendix A DesignMC User Guide		99
A.1	Background	99
A.1.1	Licence	99
A.1.2	Requirements	100
A.1.3	Installation	100
A.1.4	Function Reference	100
Appendix B Latin Squares App		119
B.1	Detailed Tour	119

Bibliography 131

Index 132

Chapter 1

Introduction

1.1 An Overview

A fortunate consequence of working with objects as ubiquitous as Latin squares, 1-factorisations of the complete graph, and triple systems (defined shortly), is that motivating their study, even to non-mathematicians, is relatively easy. Fundamentally, they are abstract mathematical concepts with many beautiful and interesting properties. Their open problems are straightforward to understand and devilishly difficult to resolve, which only add to their allure. Although they appear in many guises in different areas of mathematics (for example, designing experiments, tournament scheduling, cryptography and error correcting codes), their uses also filter through into non-mathematical environments such as marketing, manufacturing, and gaming; anyone who has completed a Sudoku puzzle has worked with a Latin square.

Given the simplicity of the definition of a Latin square, (an $n \times n$ grid in which the numbers $\{1, 2, \dots, n\}$ appear, such that each row and each column contains each number exactly once), one might expect that generating a uniformly-distributed random Latin square would be trivial. However, as we shall discuss

in section 3.1, this is not the case. Naïve approaches (for example, enumeration or hill-climbing) are either too expensive with regard to time or space, or do not actually yield the uniform distribution.

However, in 1991, Jacobson and Matthews developed a method that successfully generates (approximately) uniformly-distributed random Latin squares [29]. The technique, which we will investigate in detail in section 3.3, takes a random walk on graph whose vertices represent not only proper Latin squares, but also objects that are “almost” Latin squares, called “improper” Latin squares.

In chapter 2 we will discuss Generalised t -designs, pioneered by Cameron [6], which offer a new perspective on the relationship between combinatorial objects such as Latin squares, 1-factorisations of the complete graph, and triple systems. Before seeing the formal definition of a generalised t -design, we proceed with a broad outline of the contents of this thesis and the motivation behind it.

Cameron conjectured that it is possible to generate generalisations of the three aforementioned combinatorial objects uniformly at random by using an altered version of the Markov chain that Jacobson and Matthews described.

In chapter 4 we shall discuss generalised Latin squares, which are similar to Latin squares except the constant number of symbols in each cell (denoted λ_{RC}), the constant number of times a symbol occurs in each row (denoted λ_{RS}), and the constant number of times a symbol occurs in each column (denoted λ_{CS}) may all be different values. Much less appears in the literature about these designs, although for some particular parameter values, they have been studied (for example semi-Latin squares [1]).

Our first main result completely resolves one third of Cameron’s conjecture by showing that in the case of generalised Latin squares on n symbols, for any values of λ_{RC} , λ_{RS} , λ_{CS} which yield a square, Jacobson and Matthews’ Markov

chain may be used to generate them uniformly at random.

In chapter 5 we shall recall the definition and a selection of the known theory relating to the traditional 1-factorisation of the complete graph on $2n$ vertices. We shall see that generating these designs uniformly at random is hindered slightly because whereas Latin squares have three types of points (row, columns and symbols); here we only have two (vertices and colours). This seemingly benign difference results in the loss of useful features of the underlying graph. The reason for this is that whilst proving that the Markov chain is connected, we aim to show that we may transform any design to any other design with the same parameters. This is achieved in much the same way as one solves a Rubik's cube, that is, row by row. In proving connectedness for 1-factorisations of K_{2n} , we try to transform one into another colour by colour, however, with only two types of points, so-far insurmountable difficulties arise.

However, for a generalisation of these designs in which the number of edges connecting each pair of vertices (denoted λ_{NN}) and the number of times an edge of each colour appears at each vertex (denoted λ_{NC}) may vary, the situation is markedly improved. In fact, we are able to prove that a generalisation of Jacobson and Matthews' method will manage to generate the designs uniformly at random when $\lambda_{NN} \geq \lambda_{NC}$, $\lambda_{NC} \equiv 0 \pmod{2}$, and a design with these parameters exists. This work, and the reasons for these constraints are detailed in chapter 5.

The final member of this family is the generalised triple system. Note that these are simply the classical 2-designs with block size 3. In this case, whether $\lambda = 1$ (that is, Steiner triple systems) or $\lambda > 1$, the problem of generating uniformly-distributed random generalised triple systems is impervious to current theory. Again, it appears that by losing another type of point, things become much harder. There is mounting evidence that Jacobson and Matthews' technique may work (as over 100 distinct trades can be actioned [15]), but

probably with a significantly different method of proof. This and other evidence is discussed in more detail in chapter 6.

A more concise description of the theory presented in chapters 4-6 can be found in the author's paper [14].

The software that was used in this research (described in the next section) lends itself to many other problems in this area. For example, in chapter 7, experimentation led to a new result in the area of Latin rectangles and transversal decompositions. Namely, for $n = 3, 6, 9, 12$, any $n \times n/3$ Latin rectangle has a transversal decomposition. This also proves the new result that any 12×12 Latin square may be decomposed into 36 partial transversals of length 4.

In chapter 8, we summarise all that we have discovered and talk through some of the most interesting open problems that remain. These problems include generating Steiner triple systems uniformly at random, and the mixing time of Jacobson and Matthews' Markov chain for Latin squares, (or any of the other generalised t -designs).

1.2 Software Used

Throughout this thesis I will make references to software that has been crucial in studying these designs. The most fundamental of these is GAP [20] – a computer algebra program. GAP is a high-level, loosely-typed, open source, multi-platform, extendable programming language.

Two such extensions created by Soicher, which ultimately led to my use of the software, are the DESIGN [39] and GRAPE [40] packages. Although I did not interface with GRAPE directly, its interface with McKay’s nauty package (for finding automorphism groups and isomorphism testing of graphs [34]) is wrapped by the DESIGN package.

The DESIGN package, amongst other things, is able to create and classify block designs. It is a laudable tool for both its robustness and generality; it is quite capable of handling generalised t -designs despite being created prior to Cameron’s initial paper on the topic.

The new DesignMC package [11], amongst other things, is a wrapper for the DESIGN package that drastically simplifies the creation and classification of generalised 2-designs with block size 3.


DesignMC is open source and may be obtained from <http://www.maths.qmul.ac.uk/~ald/DesignMC>. Its features include:

- Interface with the DESIGN package to generate proper and improper designs using the `ProduceSquare`, `ProduceFactorisation` and `ProduceTripleSystem` functions;
- Implementation of Jacobson and Matthews’ Markov chain for all generalised 2-designs with block size 3 with the `Hopper`, `OneStep` and `ManyStepsProper` functions;
- Generate and analyse a random sample of designs using the included C++ file, `Sample.cpp` (GAP is useful for us due to the DESIGN package, but we

revert to C++ for speed in the absence of requiring the DESIGN package's features);

- Mathematica integration for creating graphs useful in the analysis of the Markov chains;
- Export any design to JSON (using the new JSONGAP package[12]);

Rather than give a thorough overview of what the DesignMC package can do, I will mention how the package helped with the exploration of designs as and when they appear by using the following styled box:



To construct a Latin square with $r = 5$ rows, $c = 5$ columns and $s = 5$ symbols, we use the following command:

```
gap> r:=5;; c:=5;; s:=5;;  
gap> square:=ProduceSquare(rec(v:=[r,c,s]));
```

The blocks of the design are stored as a list of lists. For a Latin square, the points that represent rows are stored as $\{1, 2, \dots, n\}$. Columns and symbols are stored as $\{n + 1, n + 2, \dots, 2n\}$ and $\{2n + 1, 2n + 2, \dots, 3n\}$ respectively.

```
gap> square[1].blocks;
```

```
[ [ 1, 6, 11 ], [ 1, 7, 12 ], [ 1, 8, 13 ], [ 1, 9, 15 ], [ 1,  
10, 14 ], [ 2, 6, 12 ], [ 2, 7, 11 ], [ 2, 8, 15 ], [ 2, 9, 14  
, [ 2, 10, 13 ], [ 3, 6, 13 ], [ 3, 7, 15 ], [ 3, 8, 14 ], [ 3,  
9, 11 ], [ 3, 10, 12 ], [ 4, 6, 15 ], [ 4, 7, 14 ], [ 4, 8,  
12 ], [ 4, 9, 13 ], [ 4, 10, 11 ], [ 5, 6, 14 ], [ 5, 7, 13 ],  
[ 5, 8, 11 ], [ 5, 9, 12 ], [ 5, 10, 15 ] ]
```

Also for your reference, a user guide that explains how to use the DesignMC package is included in appendix A.

Also, a new mobile application for generating random Latin squares was created during this research. To appeal to a wider audience, the application contains information on basic theory, papers, books and open problems. The application is freely available for download from the Apple App Store by searching for “Latin Squares”. It will run on any iPhone, iPad or iPod touch running iOS 3.0 or above.

A tour of the application can be found in appendix B.

1.3 Original Content

The work presented in this thesis is my own, however there are numerous references to historical work throughout the text, which are clearly labelled as such. Below is a summary of the novel material in this thesis, and acknowledgement of co-authors where appropriate.

- **Chapter 3:** Lemmas 5 and 7 are generalised versions of statements used in Jacobson and Matthews’ paper on generating uniformly distributed random Latin squares [29].
- **Chapters 4, 5:** Unless otherwise stated, the contents of these chapters are new and have also been published in a peer-reviewed journal [14].
- **Chapter 6:** Contains new (also published) joint work with M. Grannell and T. Griggs [15].
- **Chapter 7:** Presents a new result, and a new technique, to a problem posed by Hilton [27].

During the course of this research, various pieces of software were created. The following three have been packaged and released under an open-source licence.

- **DesignMC** (see appendix A): a GAP [20] package for generating uniformly distributed random generalised 2-designs with block size 3.
- **Latin Squares App** (see appendix B): a universal iOS application aimed at a mathematically-interested, but not university-educated audience.
- **JSONGAP**: a JSON parser written for GAP [20]. This GAP parser has been used to export designs from GAP for publication on the web (see <http://www.maths.qmul.ac.uk/~ald/designs2.html>). The designs can also be imported back into GAP, or the Latin squares application (which can also export Latin squares in this JSON format). A useful feature of the JSONGAP software is that it is able to export GAP objects, such as Groups; these objects are preceded by a "GAP://" scheme and automatically converted back to an object when imported.

Chapter 2

Generalised t -designs

A *block design* is an ordered pair (V, \mathcal{B}) , where $V = \{1, \dots, v\}$, with $v > 0$, and \mathcal{B} is a finite, non-empty multiset of subsets of V . The elements of V are called *points* and the elements of \mathcal{B} are called *blocks*.

If all of the blocks of a block design (V, \mathcal{B}) have the same cardinality $k > 0$, and, for some non-negative integer $t \leq k$, each t -subset of V is contained in exactly $\lambda > 0$ blocks, then this block design is a *t -design*. Such a t -design is often described as a $t - (v, k, \lambda)$ design; notice that we do not directly offer information on how many blocks there are, or how many times a given point occurs amongst the blocks because this information is easily deduced.

Some of the most interesting questions in this area are also the most fundamental. Given t, v, k , and λ , does a design with those parameters exist? If so, is there a construction? How many are there? Is there an algorithm for selecting such a design uniformly at random?

As a case study, we investigate the parameters $t = 2, k = 3$ and $\lambda = 1$; the *Steiner triple systems on v points* (abbreviated to $\text{STS}(v)$). One of the earliest results in Design Theory is due to Kirkman, who proved that a $\text{STS}(v)$ exists

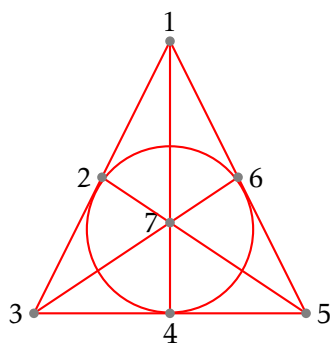


Figure 2.1: The Fano Plane. This is a graphical representation of a Steiner triple system on 7 points.

if and only if $v \equiv 1$ or $3 \pmod{6}$.

The famous *Fano Plane* (also known as the projective plane of order 2) is a graphical representation of a Steiner triple system on 7 points (figure 2.1). Each line of the Fano plane represents a block of the triple system. Every vertex that the line passes through is considered to be a point in that block. Except for one circle, all of the lines are usually straight. Up to isomorphism, there is only one STS(7).

For any $v \geq 21$, it is currently unknown how many STS(v) exist, and worse still, there is no known algorithm for choosing such a STS(v) uniformly at random in an acceptable time. An attempt to rectify this situation, not just for Steiner triple systems, but all $2 - (v, 3, \lambda)$ designs will be the topic of study for chapter 6.

Prompted by the similarity of Steiner triple systems to other combinatorial designs, Cameron developed the theory of generalised t -designs [6]. Informally, the difference between a classical and generalised t -design is a partition of the point set with the consequence that the blocks may specify how many points from each part they require, and multiple λ values dictate how many blocks contain each t -subset of the point set, depending on from which parts the points are drawn.

For example, consider the *Latin square of order n* , which is an $n \times n$ grid whose cells each contain one of the elements of the symbol set $S = \{1, 2, \dots, n\}$ such that each row and each column of the grid contains each symbol exactly once.

There are ways of describing a Latin square as a t -design, but they are not intuitive nor instructive for our purposes, so we omit them; the interested reader should see [2] for a longer discussion of the matter.

In the language of generalised t -designs, however, the description of a Latin square is very intuitive. The point set is partitioned into three types of points: rows, columns and symbols. Each block is of size three and contains exactly one point of each type. A block $\{r, c, s\}$ exists in the block set \mathcal{B} if and only if row r and column c of the Latin square contains symbol s . Any 2-subset of the point set that does not contain two points of the same type occurs amongst the block set exactly once. For example, two rows do not occur in any block together, but each row and column occur in exactly one.

For $n \geq 12$, it is not known how many Latin squares of order n exist. However, unlike the Steiner triple systems, generating Latin squares uniformly at random is possible thanks to Jacobson and Matthews' algorithm [29], which will be the focus of chapter 3. Fuelled by Cameron's conjecture below, this algorithm will be a core theme of this thesis with new proofs of various generalisations appearing in chapters 4, 5, and 6.

To formally define generalised t -designs, we use the variant of Cameron's original definition described by Soicher, who investigated generalised t -designs from their classical counterpart's perspective [41]. The key difference between the definitions is that Soicher's allows a block multiset, rather than a block set.

Given a partition $\mathbf{V} = (V_1, \dots, V_m)$ for some set $V = \cup_{i=1}^m V_i$, and some subset S of V , the \mathbf{V} -height of S is

$$[S]_{\mathbf{V}} = (|S \cap V_1|, \dots, |S \cap V_m|).$$

Let t be a non-negative integer and V a finite, non-empty set. A $t - (\mathbf{v}, \mathbf{k}, (\lambda_{\mathbf{t}}))$ design, or a *generalised t -design* with point set V , is an ordered pair $(\mathbf{V}, \mathcal{B})$, where $\mathbf{V} = (V_1, \dots, V_m)$ is an ordered partition of V with $|V_i| = v_i$, (V, \mathcal{B}) is a block design and the following properties hold:

- V has \mathbf{V} -height \mathbf{v} ;
- each block has the same \mathbf{V} -height, \mathbf{k} , with each entry in \mathbf{k} positive and the value of t is at most the sum of the entries of \mathbf{k} ;
- for each m -tuple \mathbf{t} of non-negative integers with the sum of the entries of \mathbf{t} equal to $t \leq k$, each t -subset T of V having $[T]_{\mathbf{v}} = \mathbf{t}$ is contained in the same (positive) number $\lambda_{\mathbf{t}}$ of blocks.

We now return to the two previous examples and describe them as generalised t -designs.

A STS(v) has no distinction between the points, so $\mathbf{V} = (V) = (\{1, 2, \dots, v\})$. Each block has size three, that is $\mathbf{k} = (3)$, and every pair of points occurs in exactly one block, so $t = 2$ and $\lambda_{(3)} = 1$. Therefore, STS(v) may be classified as a $2 - ((v), (3), 1)$ design.

A Latin square of order n has three point types (rows, columns and symbols) and hence the point set may be partitioned as $\mathbf{V} = (R, C, S)$ with $|R| = |C| = |S| = n$. As each block contains exactly one row, one column, and one symbol, we have $\mathbf{k} = (1, 1, 1)$, and the block set is

$$\mathcal{B} \subseteq R \times C \times S. \quad (2.1)$$

The values λ_{RC} , λ_{RS} and λ_{CS} dictate exactly how many blocks contain any row/column, row/symbol and column/symbol combination respectively. In the case of Latin squares, each of these values is equal to one. Therefore, a Latin square of order n may be described by a $2 - ((n, n, n), (1, 1, 1), (1, 1, 1))$ design.

After defining generalised t -designs and demonstrating which objects could be found for small parameter values, Cameron discussed the aforementioned algorithm by Jacobson and Matthews. In particular, he made the following conjecture:

Conjecture 1 (Cameron, [6]). *Jacobson and Matthews' algorithm for generating uniformly distributed Latin squares of order n may be used to generate any generalised 2-designs with block size 3 uniformly at random, where such a design exists.*

The conjecture covers three classes of designs:

- $\mathbf{k} = (1,1,1)$: Corresponds to a generalisation of Latin squares, (and orthogonal arrays), which in this thesis will simply be referred to as “squares”. Jacobson and Matthews have already handled the case when each of the λ values are equal to one. In chapter 4, we will prove this conjecture for squares of all admissible parameter values.
- $\mathbf{k} = (2,1)$. Corresponding to a generalisation of 1-factorisations of the complete graph. In chapter 5, we give a proof of the conjecture for some parameter values (that is, when $\lambda_{(2,0)} \geq \lambda_{(1,1)}$ and $\lambda_{(1,1)}$ is even) and explain why this case was harder to handle than that of squares.
- $\mathbf{k} = (3)$. The final case is the same as the classical triple systems. With only one type of point to work with, this case seems harder still as we will see in chapter 6. Although we will do not provide a proof that Jacobson and Matthews' method works for these objects, we will see evidence in favour of the conjecture.

We end this section and chapter with one final definition. As it will sometimes be appropriate to discuss all three of these objects simultaneously, we will use the umbrella term *designs* to refer to them collectively.

For further reading on this topic, Cameron's paper is the suitable place to start for a good introduction to the area with examples of designs for small

parameter values [6]. As well as the variant definition, Soicher's paper provides a number of results including strong restrictions on \mathbf{k} for generalised t -designs with block size k , constant λ_t , and $2 \leq t \leq k - 2$ [41]. Finally, Martin studied and gave constructions for $t - (\mathbf{v}, (k_1, k_2), (\lambda_t))$ designs ten years prior to Cameron [32]; he called these *mixed block designs*.

Chapter 3

Generating Latin Squares

Uniformly at Random

3.1 Latin Squares

Let L be an $n \times n$ grid with the property that each cell of L contains exactly one symbol from the set $S = \{1, 2, \dots, n\}$. We say L is a *Latin square of order n* , denoted $LS(n)$, if every symbol occurs in each row, and each column, exactly once. We will usually index the rows and columns with $\{r_1, r_2, \dots, r_n\}$ and $\{c_1, c_2, \dots, c_n\}$ respectively.

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

Figure 3.1: The cyclic $LS(5)$. To generate this square we write down the numbers from 1 to 5 in the first row. For each subsequent row, cycle the numbers around by one position.



The DesignMC package represents rows as $\{1, 2, \dots, n\}$, columns as $\{n + 1, n + 2, \dots, 2n\}$, and symbols as $\{2n + 1, 2n + 2, \dots, 3n\}$. For example, the first two rows in the grid shown in figure 3.1 may be represented by the DesignMC package as:

```
[ [ 1, 6, 11 ], [ 1, 7, 12 ], [ 1, 8, 13 ], [ 1, 9, 15 ], [ 1,
10, 14 ], [ 2, 6, 12 ], [ 2, 7, 11 ], [ 2, 8, 15 ], [ 2, 9, 14
], [ 2, 10, 13 ] ]
```

Given a positive integer $n \geq 2$, it is easy to see that a $LS(n)$ exists. For example, setting the symbol found in row i , column j to be $(i + j - 1) \pmod{n}$ constructs the *cyclic Latin square* [28], whose interesting properties will be discussed later.

3.1.1 Counting Latin Squares

One of the biggest open problems for Latin squares is determining exact values for the number of “different” Latin squares of a given order. In the mainstream literature, three tiers of “difference” are usually given.

Two squares L and L' are:

1. *Isotopic*: If there exists a permutation of the rows, columns and symbols that transforms L into L' . For example, if we have a function $f : \mathbb{Z} \rightarrow \mathbb{Z}$, $f(a) = (a \pmod{n}) + 1$ and we apply f to the row component of each block in the block set of some Latin square, the result is an isotopic Latin square.
2. *Conjugate*: If there exists a permutation of the roles of the rows, columns and symbols that transforms L into L' . For example, switching the roles of the rows and columns transposes the square.
3. *Main class isotopic*: If L is isotopic to a conjugate of L' .

We define two squares to be *isomorphic* if and only if they are main class isotopic, otherwise, they are *non-isomorphic*.

For $1 \leq n \leq 11$ the following table gives the exact number of non-isomorphic Latin squares of order n . At the time of writing, no exact values are known for $n \geq 12$.

n	Number of non-isomorphic LS(n)	References
1	1	
2	1	
3	1	
4	2	
5	2	Euler, 1782 [16]
6	12	Frolov, 1890 [19]
7	147	Sade, 1948 [37]
8	283657	Wells 1967, [47]
9	19270853541	S. Bammel, J. Rothstein, 1975 [4]
10	34817397894749939	B. McKay, E. Rogoyski, 1995 [33]
11	2036029552582883134196099	McKay, Wanless, 2005 [35]

Figure 3.2: A table showing the number of non-isomorphic Latin squares for small orders; as you can see from the table, the number of squares explodes quite quickly.

For a comprehensive overview of the information on the number of Latin squares of order up to 11 see [35].

3.1.2 Generating Latin Squares

So far we have only seen a construction for the cyclic square. How can we find other squares? Can we find a square uniformly at random?

To select a square uniformly at random, it would suffice to enumerate each square and then pick one uniformly at random. However, it would be quite a formidable task to enumerate all of the Latin squares of order 11, and presumably by the time the order passes the relatively low value of, say, 20, there are more squares than there are elementary particles in the visible universe!

Hill Climbing

Clearly the idea of enumerating every square is not feasible. One of the most common methods to generate a Latin square is *hill climbing*. Before explaining how the method works, a little more machinery is required.

Let R be an $r \times n$ grid ($r \leq n$) with the property that each cell of R contains exactly one element from the symbol set $S = \{1, 2, \dots, n\}$. We say R is a *Latin rectangle* if every symbol occurs in each row, and each column, at most once.

Let $S = \{S_1, S_2, \dots, S_n\}$ be a finite collection of finite sets. A *system of distinct representatives*, or SDR, of S is a set

$$x_1 \in S_1, x_2 \in S_2, \dots, x_n \in S_n$$

such that $x_i \neq x_j$ whenever $i \neq j$.

Theorem 2 (Hall's Marriage Theorem). *Let $S = \{S_1, S_2, \dots, S_n\}$ be a finite collection of finite sets. There exists a system of distinct representatives of S if and only if the following condition holds for any $T \subseteq S$:*

$$\left| \bigcup_{s \in T} s \right| \geq |T|$$

From this we get the following well known result:

Corollary 3. *Any $r \times n$ Latin rectangle ($r < n$) can be completed to an $(r+1) \times n$ Latin rectangle.*

To find a $LS(n)$ using the hillclimbing technique, first write down row 1 as $1, 2, \dots, n$ in order. Now look for suitable candidates for the next row – any *derangement* (that is, a permutation of $1, 2, \dots, n$ with no fixed points) will do for the second row. Now fill in the rest of the square row by row, by choosing from the set of suitable SDRs until the whole square is complete. We know from corollary 3 that we do not get stuck and therefore obtain a $LS(n)$.

The hill climbing method does generate random Latin squares relatively efficiently, but the sampling is not uniform, as the number of SDRs available is dependent on your choice of second row. For example, suppose we start hill climbing a LS(4) with first row 1,2,3,4. There are now 9 possible second row choices.

- Suppose we choose 2,1,4,3 for row two. Now there are four possible completions to a Latin square, they are:

A=	
----	--

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

B=	
----	--

1	2	3	4
2	1	4	3
4	3	1	2
3	4	2	1

C=	
----	--

1	2	3	4
2	1	4	3
3	4	2	1
4	3	1	2

D=	
----	--

1	2	3	4
2	1	4	3
4	3	2	1
3	4	1	2

- Suppose we choose 2,3,4,1 for row two.

Now there are only two possible completions to a Latin square, they are:

E=	
----	--

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

F=	
----	--

1	2	3	4
2	3	4	1
4	1	2	3
3	4	1	2

So when hill climbing, squares E and F are twice as likely to appear as any of A, B, C or D and therefore this method does not uniformly sample the space. As n gets large, it is conjectured that the ratio of number of completions tends to 1 [7].

Ruthless Hillclimbing

Another, more ruthless method, is a modification of hill climbing. This time, do not worry about finding SDRs, merely look at all possible permutations and select one uniformly at random to add as the next row. If, in doing so, you violate the rules of being a Latin square, restart the entire process. This method terminates with probability 1 and does achieve the uniform distribution. However, if $L(n)$ is the total number of $LS(n)$, the expected number of restarts is $n!^{n-1}/L(n) = e^{n^2(1+o(1))}$; an unacceptable price to pay for uniformity [29].

Cycle Swapping

A cycle swap is an iterative procedure that can be performed on a pair of rows, columns or symbols and may result in a different Latin square. We will discuss a row switch, but column and symbol switches are analogous. This important concept will be referenced in many places further through the text.

We use the convention that $\{r, c, s\}$ is a block of a Latin square if the symbol s is located in cell (r, c) . Further, we denote the set of all blocks by \mathcal{B} .

To begin a cycle switch on rows we first nominate two rows, r and r' , and construct the *row-pair graph* (also found in the literature as *cycle* or *neighbourhood* graphs), G , which has a vertex for every column and symbol of the square. Add a red edge between two vertices c and s if $\{r, c, s\} \in \mathcal{B}$. Similarly, add a blue edge between two vertices c' and s' if $\{r', c', s'\} \in \mathcal{B}$. This graph is therefore a union of disjoint cycles of even length. The construction for switching columns or symbols is analogous.



DesignMC

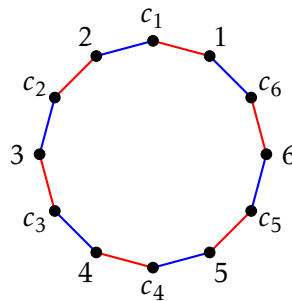
The DesignMC package may be used to display pair graphs using the `CreatePairGraph` function.

If G consisted of just one cycle and we were to interchange the colour of the edges, this would correspond to swapping the position of the two rows of the original square, leaving us with an isomorphic square. In general, if G consists of $k \geq 2$ cycles and we switch at most $k-1$ of them, then we will produce a square which is not isomorphic to the original.

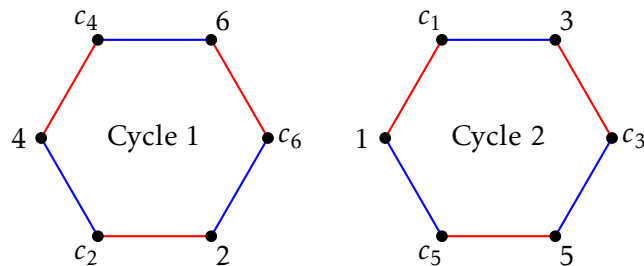
For example, consider the cyclic square of order 6

1	2	3	4	5	6
2	3	4	5	6	1
3	4	5	6	1	2
4	5	6	1	2	3
5	6	1	2	3	4
6	1	2	3	4	5

We shall construct the row-pair graph by nominating rows 1 and 2.



Interchanging the edge colours of this graph corresponds to swapping the position of the rows. If we had chosen rows 1 and 3, we would have resulted in the following graph:



If we only interchange the edges of “cycle 2”, this would correspond to the following square which is not isomorphic to the original.

3	2	5	4	1	6
2	3	4	5	6	1
1	4	3	6	5	2
4	5	6	1	2	3
5	6	1	2	3	4
6	1	2	3	4	5

The smallest possible cycle has length four and corresponds to an *intercalate*, that is, a 2×2 subsquare of a Latin square. In the literature, interchanging the symbols in an intercalate is called “intercalate switching”, “turning an intercalate” or “intercalate reversal”. Intercalate switching was employed in [36] by Norton as a means of discovering Latin squares of order 7. He began with a small population of squares and, where possible, performed intercalate switches to find as many other squares as he could; he found 146 of the 147 squares in this way.

Intercalate switching does not provide the means to discover all Latin squares of a given order starting from just one Latin square [38]. Furthermore, using larger cycle switches still cannot yield every square [29]. For a more thorough treatment of cycle switching in Latin squares, see [44]. In section 3.3.1 we will revisit the concept of these cycles for more general combinatorial objects.

Later we will see how Jacobson and Matthews addressed this issue by using a Markov chain, but before going into detail about how they proceeded, it will be necessary to take a short diversion through the underlying theory.

3.2 Markov Chains

Let X_i be the number of times a six has appeared in the first i rolls of a die. The sequence $X = (X_0, X_1, X_2, X_3, \dots)$ is Markov chain because it has each of the

following:

- **A state space:** Usually denoted as Ω , the state space contains all of the possible values X_i may take. Although the state space may be finite or infinite, continuous or discrete, we will always consider finite, discrete state spaces. In our example above, the state space $\Omega = \mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$.
- **A starting state:** The Markov chain moves from state to state using some algorithm autonomously. However, we must set our starting state X_0 . For the die example, $X_0 = 0$ because we have seen zero sixes after zero rolls.
- **It is *memoryless*:** We move from state to state according to some function $f : \Omega \rightarrow \Omega$. All previous states are irrelevant; only the current state is considered. If on the t^{th} roll we saw a 6, $f(X_t) = X_{t-1} + 1$, otherwise $f(X_t) = X_{t-1}$.
- **A concept of time:** This dictates how often we move from the current state. In this thesis, we shall always work in discrete-time.

Formally, a *Markov chain* is a sequence of random variables, called *states*, $(X_0, X_1, X_2, X_3, \dots)$ with the *Markov property*, which means that given the present state, the past and future states are independent. That is,

$$\Pr(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = \Pr(X_{n+1} = x \mid X_n = x_n).$$

In an effort to animate the rest of the theory in this section, and as an introduction to Jacobson and Matthews' method, we will now look at another basic Markov chain.

3.2.1 Example: Random Walk

Suppose G is a finite, connected graph with no loops or multiple edges with vertex set $V(G) = \{v_1, v_2, v_3, \dots, v_n\}$. We will construct a *random walk* on G , that is, a randomly created alternating sequence of vertices and edges starting and ending on a vertex.

Let the initial state $X_0 = v_1$ and, for some positive integer t , let X_{t+1} be a randomly chosen vertex from the *neighbour set* of X_t , defined by $N(X_t) = \{v \in V(G) : X_t \text{ and } v \text{ are connected by an edge}\}$.

After n iterations, $(X_0, X_1, X_2, X_3, \dots, X_n)$ could be any of the possible walks in G starting at x of length n . Note that the knowledge of where the walk has been yields no information to where the walk will move to at time $n + 1$; it only depends on the current vertex.

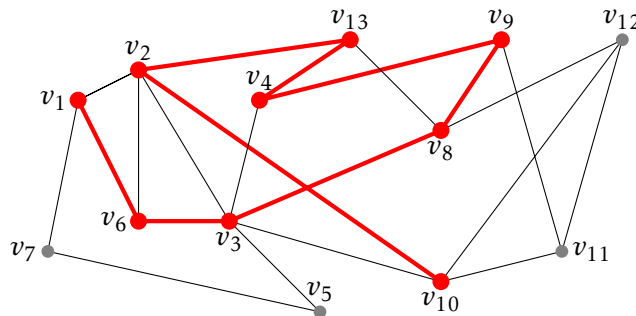


Figure 3.3: A random walk on a graph G . The walk began at v_1 and finished at v_{10} , as depicted here by the red edges; the grey edges were not used in the walk.

We shall now take a brief tour through some of the properties that Markov chains can exhibit. Given any pair of states i, j , if there exist a sequence of states that begins in state i and ends in state j , we say that the Markov chain is *irreducible*. Note that for a graph this means a random walk is irreducible if and only if the graph is connected. If we were hoping to use a random walk to select a vertex uniformly at random, the random walk would have to be irreducible otherwise some vertices have no chance of being chosen.

If a Markov chain has the property that the probability of moving from state i to state j , denoted P_{ij} , is equal to the probability of moving from state j to state i , then we say the Markov chain is *reversible*. Unless G is regular, the probability of moving from vertex v_i to v_j needn't be the same as moving from state v_j to v_i . Sampling uniformly at random is easily thwarted by an irregular graph (see figure 3.4). Also note that if the edges were directed, the situation can be even more complicated.

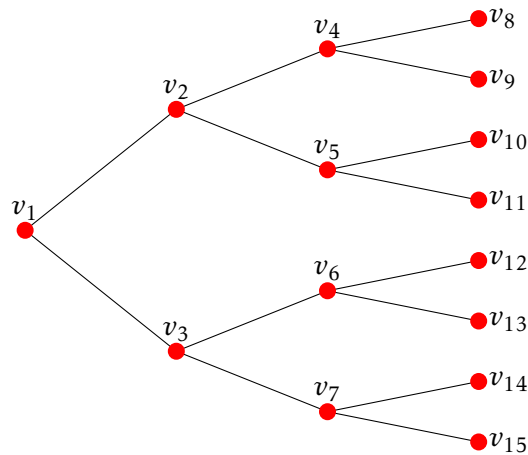


Figure 3.4: Although a random walk on this tree is irreducible, the non-reversibility prevents uniform sampling because, for example, v_1 is less likely to be discovered because when you are at any of v_2, \dots, v_7 , you are more likely to move to the right.

A state i has a *period* k if any revisit to state i must occur in multiples of k time steps. If $k = 1 \forall i$, then the Markov chain is *aperiodic*. The canonical example of a random walk that is not aperiodic is a bipartite graph; every state has period 2. Generally, any random walk on an undirected graph that contains an odd cycle (including loops) is aperiodic. In other words a Markov chain on a graph is aperiodic if and only if the graph is non-bipartite.

Moreover, if a Markov chain is aperiodic and reversible, it is *ergodic*. Ergodicity is usually summed up in the pithy statement “time averages equal space averages”. For our purposes, it is sufficient to understand that if a Markov

chain is ergodic, then after sufficient time has passed, the systems “forgets” where it started. In other words, no matter what your choice of X_0 , the sampling distribution of chain will converge; we call this distribution the *unique stationary distribution*.

A random walk on a finite, connected, undirected, non-bipartite, regular graph G is ergodic with the uniform stationary distribution. The amount of time required before the Markov chain exhibits this property is called the *mixing time*. Even though a chain converges to the uniform distribution, it might take too long. We will discuss mixing time in more detail in chapter 6. As a teaser, we admit that the mixing time for Jacobson and Matthews’ Markov chain is currently unknown.

The investment we have made in this section will hold us in good stead for what follows where we will study how Jacobson and Matthews used a Markov chain to generate uniformly distributed random Latin squares. We have only touched upon the very basics of the fascinating world of Markov chains. They are hugely powerful and I urge the interested reader to see [48] for a broader and deeper coverage.

3.3 Jacobson and Matthews’ Technique

In the previous section, we took a detour through some basic Markov chain theory and saw that a random walk on a finite, non-bipartite, regular, undirected, connected graph is ergodic with the uniform stationary distribution. Returning to the main focus of this thesis, we shall now see how, in 1991, Jacobson and Matthews used a random walk to generate (approximately) uniformly distributed random Latin squares.

Consider a graph whose vertex set contains a representative for every Latin square of order n exactly once. Let M be some operation on Latin squares. We add an edge between two vertices v_1, v_2 of our graph if $M(v_1) = v_2$ and

$M(v_2) = v_1$. If the edges that M creates form a connected, finite, non-bipartite, regular, and undirected graph, then we may take a random walk upon it to generate an approximately uniformly distributed random Latin square. Recall from section 3.1 how cycle switching can be used to move from one square to the another. The move consisted of forcing some symbol s' into some cell (r, c, s) , which then caused a chain reaction of changes to the square. When M is the operation of cycle switching, clearly the graph is finite as there are only finitely many $LS(n)$. Consider the cyclic square on p symbols where p is prime. Any cycle swap attempted will result in an isomorphic square, and therefore the graph has at least two components. This means that cycle switching is not a good candidate for generating uniformly distributed random Latin squares.

Jacobson and Matthews introduced a more general variant of the cycle switching operation that fulfils all of the conditions needed. Their first major insight is that their graph not only contains a vertex representing each Latin square, but it also contains some vertices for “improper” Latin squares. An *improper* $LS(n)$ is a $LS(n)$ with the added condition that some cell can contain a symbol -1 times. Suppose the symbol s occurs -1 times in row r , column c ; in block design notation, we denote this as $-(r, c, s)$. To comply with the rules of Latin squares, s must occur in row r and column c twice more so that in net it occurs exactly once in each. Further, as each cell must contain exactly one symbol, the cell found at the intersection of row r and column c must contain two further (proper) symbols. For example, the smallest improper Latin square (which is unique up to permuting rows, columns, symbols, or the roles of rows, columns and symbols) is:

2, 3, -1	1	1
1	3	2
1	2	3

Which, in block notation may be written as:

$$\mathcal{B} = \{-(r_1, c_1, 1), (r_1, c_1, 2), (r_1, c_1, 3), (r_1, c_2, 1), (r_1, c_3, 1), (r_2, c_1, 1), \\ (r_2, c_2, 3), (r_2, c_3, 2), (r_3, c_1, 1), (r_3, c_2, 2), (r_3, c_3, 3)\}$$

Analogously to Latin squares, two improper Latin squares I, I' are *isotopic* if there exists a permutation of the rows, columns and symbols that transforms I into I' . Further, I and I' are *conjugate* if there exists a permutation of the roles rows, columns and symbols that transforms one into the other. Finally, if I is isotopic to a conjugate of I' , then the two squares are *isomorphic*.

Their second major insight is how to move from one (proper or improper) square to another. For proper squares, we begin in a similar fashion to cycle switching. We randomly pick a cell, say (d, e, f) , and some symbol, say f' . We will add f' to our chosen cell and remove f . This now means that f' occurs in row d and column e twice. To fix this, find the row d' such that (d', e, f') exists and swap it for (d, e, f) . Similarly, do the same for the column e' such that (d, e', f') exists. This means that row d and column e now contain the correct number of occurrences of f and f' . However, row d' and column e' contain f twice and f' doesn't occur at all. The final part of the move occurs in the cell that completes the subsquare, that is, (d', e', x) for some symbol x . If $x = f$, we swap it for f' and we end with a Latin square. If $x \neq f$, then we still add an f' and introduce a $-f$ into that cell, resulting in an improper square.

The move beginning from an improper square is similar, except instead of picking a random cell at the start, you must use the improper cell and add in the symbol that occurs -1 times. By doing this, we ensure that we only ever have at most one improper cell at any one time. Note that there are eight possible conclusions to the move because there are two proper symbols in the improper cell, and two occurrences of the improper symbol in that row and column.

We can formalise this move in the following algorithm:

± 1 -move [Jacobson and Matthews, 1991]

1. If the current state is proper, pick any admissible cell (d, e, f) . If the current design is improper, let d , e and f be the row, column and negative symbol contained in the improper cell.
2. Find d', e', f' such that (d', e, f) , (d, e', f) , (d, e, f') exist.
3. Now we perform the following “trades”:
 - (a) Add the following blocks: (d, e, f) , (d, e', f') , (d', e, f') , (d', e', f) .
 - (b) Remove the following blocks: (d, e, f') , (d, e', f) , (d', e, f) . If you can, also remove (d', e', f') . If you cannot remove (d', e', f') (because it doesn't exist in the block set), then we develop an improper block $-(d', e', f')$ and are left with an improper design. Note that this means we can never have more than 1 improper block.

Moves will often be represented in *table notation*. This is a table with two columns and four rows; each row in the first column contains a block that should be added to the block set and each row in the second column contains a block that should be removed from the block set. We will omit the brackets from each block in the table for a cleaner presentation. For example: in table notation, step three in the ± 1 -move above would be represented like this:

+			-		
d	e	f	d'	e'	f'
d	e'	f'	d	e	f'
d'	e	f'	d	e'	f
d'	e'	f	d'	e	f



DesignMC

Given a square (or any other generalised 2-design with block size 3), the DesignMC package can move around the underlying graph of this Markov chain by using the `Hopper`, `OneStep`, `ManyStepsProper` and `ManyStepsImproper` functions.

An example of Jacobson and Matthews' technique

To illuminate the theory, we shall now see an example of how we can move from one square to another by using improper squares as stepping stones. There are (up to isomorphism) only two Latin squares of order five. Suppose that we begin with the cyclic square and would like to find the other. We shall now demonstrate how Jacobson and Matthews' technique can succeed where simple cycle switching cannot.

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

Step One The technique requires a starting point, the so-called X_0 . We shall begin with the cyclic square on 5 symbols. As this is a proper square, step 1 in the algorithm expects us to choose any row, column and symbol with which to form a block. We shall choose $(r_1, c_4, 5)$.

1	2	3	5	4
2	3	4	4	1, -4, 5
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

Step Two In row 1, column 4, we already have the symbol 4. We are going to remove it and add the symbol 5 in its place. Highlighted in orange are the places where 5 already occurred; we replace each 5 by a 4. Now, in both row r_1 and column c_4 the number of occurrences of symbol 4 is correct. However, we now have too many 4's and no 5's in column c_5 and row r_2 . To complete the move, we add a 5 and remove a 4 from the cell found at row r_2 , column c_5 , highlighted in blue.

1	2	3	5	4
2	3	4	1	5
3	4	5	1	2
4	5	1	2	3
5	1	2	3, -1, 4	1

Step Three As we have an improper square, we are forced to add $(r_2, c_5, 4)$ back. However, we may choose whether to remove a 1 or a 5 from this cell – we shall remove a 1. We must remove a 4 from both row r_2 and column c_5 , and we have two choices in each case. The choices that were made have been highlighted in orange. We complete the move by adding a 4 and subtracting a 1 from the cell found at row r_5 , column c_4 .

1	2	3	5	4
2	3	4	1	5
3	1	5	4	2
4	5	1	2	3
5	4	2	3	1

Step Four We can concisely describe the move made by carefully choosing a block that we have added and a block that we have removed. For example, we can return to properness by performing the unique move that corresponds to adding $(r_5, c_1, 2)$ and removing $(r_1, c_2, 1)$. Finally, we have used the ± 1 -move to find the non-cyclic square of order 5.

In the remainder of this section, we shall give an overview of how Jacobson and Matthews proved connectedness of the underlying graph, which ultimately climaxes in the following theorem:

Theorem 4 ([29], theorem 4). *Let X_0^* be an arbitrarily distributed order- n Latin square that starts a Markov chain of (proper and improper) squares: to each square, apply a move chosen uniformly at random from the permissible ± 1 -moves $[n^2(n-1)$ from a proper square, 8 from an improper square]. Let $X^* \equiv (X_1, X_2, X_3, \dots)$ be the subsequence of proper squares we encounter; then X^* is a Markov chain with a (unique) stationary distribution that is uniform over the set of order- n Latin squares. If $n \geq 3$, the chain is ergodic.*

3.3.1 Pair Graphs

In section 3.1 we learned how to construct a pair graph from two rows, columns or symbols of some Latin square L . We shall now expand this definition to include improper Latin squares. From now on, we will denote a *pair graph* as $D = D_L(a, b)$, where a, b are the two nominated rows, columns or symbols and L is a proper or improper square. As before, the vertex set contains one element for each point that is not the same type as a and b . For example, if a and b were

columns and we were creating the column-pair graph $D_L(a,b)$, then D would contain exactly one vertex for each row and symbol of our square, but no vertices representing columns. To form edges, we connect two vertices x,y with a red edge if (a,x,y) exists. Similarly, we connect two vertices x,y with a blue edge if (b,x,y) exists. We allow a to be one of the points contained in the improper block, but never b ; this is purely to aid our discussion.

For any proper square, the vertices of the pair graph are all incident with the same number of blue edges as red edges. For example, consider a row-pair graph $D_L(r,r')$. Each column-representing vertex has a red edge for each of the incidences that it has with row r – for proper Latin squares, this is always exactly 1. Similarly, the column only appears in exactly one block with r' , so this vertex has exactly 1 blue edge.

If L is an improper square, then we have some improper block $-\{j,k,l\}$. If $a \in \{j,k,l\}$, without loss of generality suppose $a = j$, then D will contain exactly two *special* vertices k,l – they are special because the number of red edges incident with them is exactly one more than the number of blue edges incident with them. This is because j and k must occur together precisely once, and as the improper block contributes -1 to this value, there must be an extra incidence to compensate (similar for j and l). Also, there is no red edge connecting the special vertices. As the definition of the pair graph forbids b from being a member of the negative block, we may unambiguously refer to “the red component of the pair graph containing the special vertices”.

In what follows, we are going to work exclusively with row-pair graphs. This distinction is to ease discussion and the reader should be aware that permuting the words “row”, “column” and “symbol” would yield the same results.

Jacobson and Matthews classified the five different types of row-pair graph. We shall quickly recall them here. To aid us, we will use the following improper Latin square of order 7 to illustrate each type.

1	5	4	2	6	7	3
2, -3, 4	1	3	6	5	3	7
5	3	2	4	7	1	6
3	6	7	5	1	2	4
3	2	5	7	4	6	1
7	4	6	1	3	5	2
6	7	1	3	2	4	5

Type 1: This graph is a union of even cycles, each of length at least 4. As well as having the property that the edges alternate, we also have the property that the vertices alternate between representing columns and symbols. These are actually the same graphs as we constructed in section 3.1. If our square has $-(r, c, s)$, then we may form a type 1 graph by forming the row-pair graph with any pair of proper rows that do not contain a column conflict. For example, in figure 3.5 we show $D_L(1, 3)$.

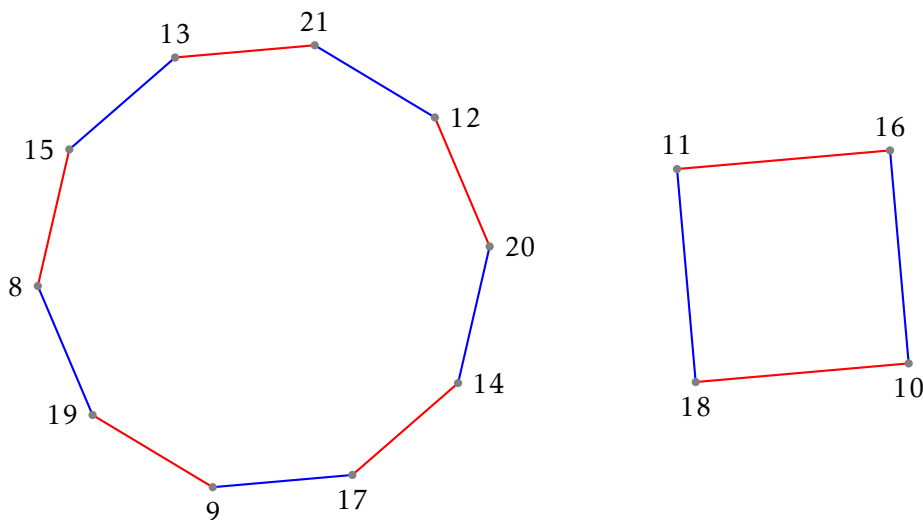


Figure 3.5: A type 1 row-pair graph constructed from rows that do not contain the improper cell.

Type 2: Still not using the improper row, a type 2 graph is similar to a type 1 graph except that we have a column conflict in the rows we choose. In general,

suppose we have $-(r, c, s)$, then there must exist rows r', r'' such that (r', c, s) and (r'', c, s) exist. If we form the row-pair graph $D_L(r', r'')$, then we will obtain a type 2 graph. The identifying factors of a type 2 graph are that every vertex has degree 2 and there exists a 2-cycle. In figure 3.6, we use rows 4 and 5 of L where the symbol 3 appears in column 1 of both rows.

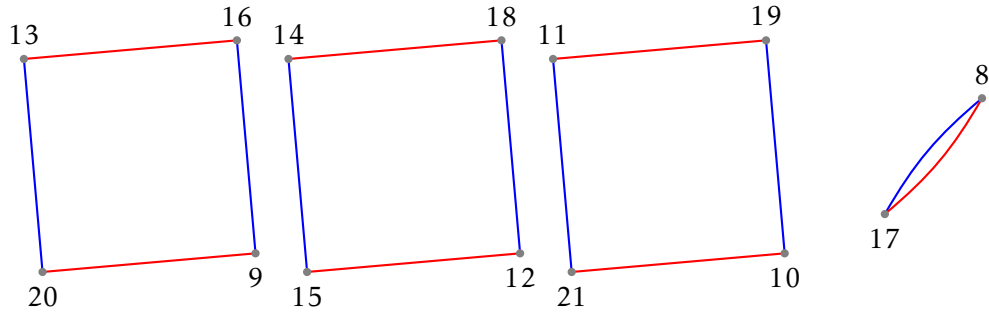


Figure 3.6: A type 2 row-pair graph constructed from $D_L(4,5)$.

Type 3A: Suppose we have $-(r, c, s)$, then a type 3A graph is constructed by using rows r and r' , such that (r', c, s) exists. Type 3A graphs may be thought of as a collection of Type 1 graphs with an additional component that contains both special vertices, which are connected by a blue edge. Continuing use of the above square as an example, we may create a type 3A row-pair graph with rows 2 and 4, or 2 and 5 see 3.7. Remember that as a matter of notation, we are only going to let the improper row label red edges. Also, we represent the improper edge with a dashed line.

Type 3B/C: Given that we have $-(r, c, s)$, both of these graphs are formed by using rows r and r' such that (r', c, s) does not exist. In the graph $D_L(r, r')$, locate one of the special vertices and trace out an alternating path by moving away along a blue edge. When you arrive at a special vertex, stop. Depending on the structure of the square, either you will find the other special vertex, in which case the graph is of type 3B. Alternatively, you will return to where you started before without finding the other special vertex, in which case we classify

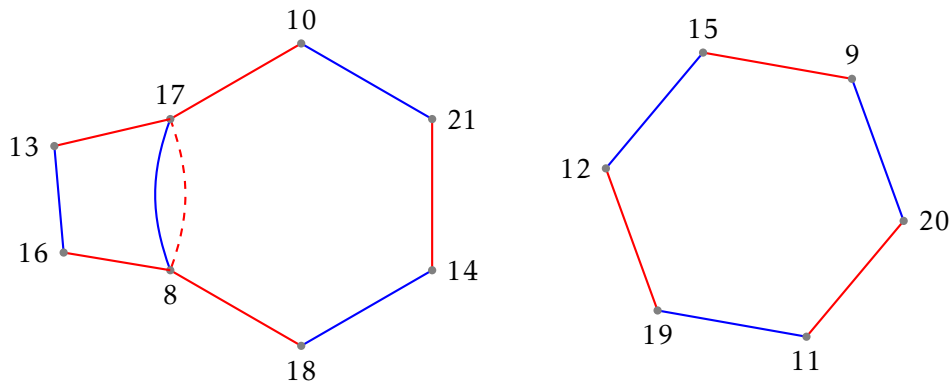


Figure 3.7: A type 3A row-pair graph involving the improper row (row 2) and the proper row 4, that is, $D_L(2, 4)$.

the graph as type 3C. Type 3B graphs are similar to type 3A, except that instead of having an edge connecting the special vertices, there is an alternating path of length at least 3 (see figure 3.8). Type 3C graphs are characterised by having two disjoint alternating cycles of even length which each contain a special vertex. There is an alternating path connecting these cycles starting and ending at the special vertices. There may optionally be some other type 1 graphs (see figure 3.9).

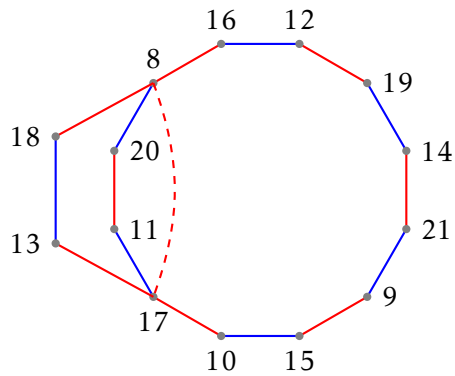


Figure 3.8: A type 3B row-pair graph involving the improper cell $D_L(2, 7)$.

Pair graphs are a very useful tool for proving connectedness of Jacobson and Matthews' Markov chain, as well as generalisations of it that will follow later.

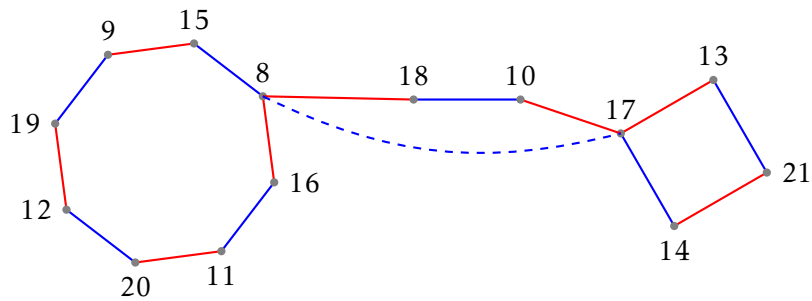


Figure 3.9: A type 3C row-pair graph involving the improper cell $D_L(1,2)$.



As mentioned earlier, the DesignMC package can be used to display pair graphs (with the aid of Mathematica) using the `CreatePairGraph` function.

The following two lemmas were used without formal proof in the original paper. Not only will we detail the proofs here, but we'll generalise the statements because we shall get more use from them later.

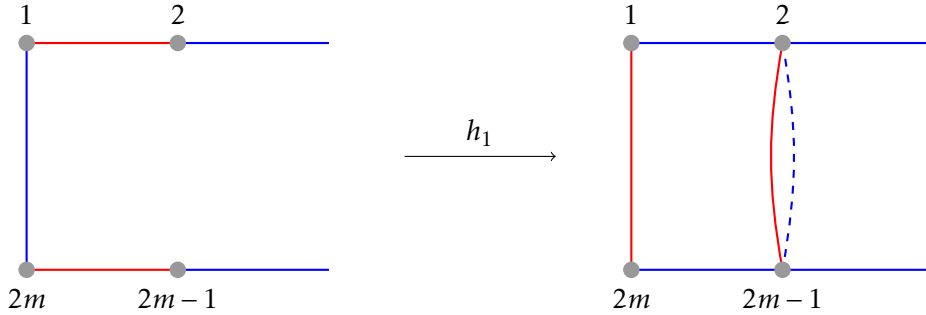
Lemma 5 (Closed Alternating Trail Switching). *Using Jacobson and Matthews' ± 1 -move, we may interchange the edge-colours of a closed alternating trail, T , in some pair graph without altering any edges outside of this trail.*

Proof. We create a modified version of our alternating trail T by splitting all of the vertices with degree $d > 2$ into $d/2$ child vertices in such a way that we form an alternating cycle; we can clearly do this by following the alternating trail and splitting off new vertices as we need to. Let the vertices be labelled with $1, 2, \dots, 2m$ and call this new graph H . To interchange the edge colours of H , we perform m moves, where move i is defined to be:

$$\begin{array}{c}
\begin{array}{c|c}
+ & - \\
\hline
i & 2m-i & c_i & i+1 & 2m-i & c_{i+1} \\
h_i = & i & 2m-i & c_{i+1} & i & 2m-i+1 & c_{i+1} \\
& i+1 & 2m-i+1 & c_{i+1} & i & 2m-i & c_i \\
& i+1 & 2m-i & c_i & i+1 & 2m-i+1 & c_i
\end{array}
\end{array}$$

where

$$c_k = \begin{cases} \text{red} & \text{if } k \text{ is odd} \\ \text{blue} & \text{if } k \text{ is even} \end{cases}$$

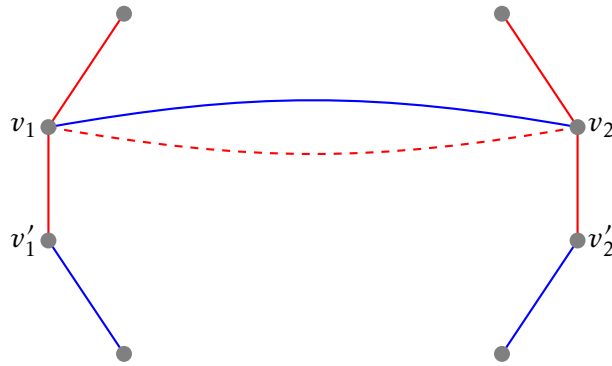


If we now replace each vertex label by its parent's label in each h_i , we create a set of Markov chain moves that will interchange the colours of our original alternating trail T . Note that although you may now be making moves on a graph with loops or multiple edges, each move works on the same edges as before and is apathetic that vertices may now intersect. \square

Corollary 6. *The edge labels of a type 1 graph may be interchanged.*

Lemma 7 (Return to Properness). *Suppose L is an improper square with $-(a, v_1, v_2)$. Let $D = D_L(a, b)$ be a pair graph. Given a closed alternating trail (on red and blue edges) including a blue edge connecting the two special vertices v_1, v_2 in the pair graph, D_L , one may return to properness using the ± 1 -move without altering any edges outside of this trail.*

Proof. We focus attention on the interesting part of the pair graph, which looks like this (the dashed red line between v_1, v_2 represents the improper block):



Begin with a move to add a red edge between v_1 and v_2 . Now we must remove a red edge incident with v_1 (choose $\{v_1, v'_1\}$) and v_2 (choose $\{v_2, v'_2\}$). Also, we must remove a blue edge and add a red edge between $\{v'_1, v'_2\}$. If such a blue edge existed, then we are now proper. However, if there was no blue edge between $\{v'_1, v'_2\}$, then we have been left with an improper design with a shorter closed alternating trail between two special vertices v'_1, v'_2 . Repeat the move to shorten the trail until you have a trail with only four edges in it, the next time you perform the move you will return to properness. \square

Corollary 8. *Any type 3A pair graph may be converted to a type 1 graph.*

In keeping with the original paper, the process described in the previous proof may be referred to as “sliding the chord”.

3.3.2 Proving Connectedness

Of all the conditions required for this Markov chain to be ergodic with uniform stationary distribution, connectedness requires the most attention. In fact, in the upcoming generalisations, only connectedness needs to be proved as all of the other features are covered by the original work.

Jacobson and Matthews presented their connectedness theorem with the aid of two lemmas. The first of these lemmas shows that it is possible to make small changes in a row causing damage in at most two other rows. The proof consists of three cases where the first two are much smaller than the third. For our

purposes, it will be instructive to give the proof of the first two cases separately from the third. For future generalisations, it will only be necessary to generalise case 3. If the reader is interested in viewing the unadulterated proof, see [29], lemma 2.

The second lemma shows that we can use the first to transform some square into any given square by working one row at a time.

Lemma 9. *Suppose that for some proper row t we have (t, c, s) and (t, c', s') , with the additional condition that, if the square is improper, we have $-(r, c, s)$ (for some $r \neq t$). Further, suppose we have (r, c', s) . Then there is a sequence of ± 1 -moves that leaves a working square having (t, c, s') and (t, c', s) but, apart from making this swap in row t , changes incidences in only row r ; additionally, if the new working square is improper, it has $-(r, c, s')$.*

Proof. **Case 1:** If the square is proper, we can prove the result by performing the following move:

+		-	
t	c	r	c'
t	c'	t	c
r	c	t	c'
r	c'	r	c

As you can see we now have the blocks (t, c, s') and (t, c', s) whilst minimising damage to only row r .

Case 2: If the square is improper, then we may prove the result by performing two moves. Note that because the square has an improper cell, we are forced to add it.

+			-	-		
r	c	s	t	x	s'	
r	x	s'	r	c	s'	
t	c	s'	r	x	s	
t	x	s	t	c	s	

This move has gained us ‘half’ of our requirements in that we now have (t, c, s') . Unfortunately, we have damaged column x , whatever that may be. We shall fix this damage, and gain the other required cell in the following move (note that whether or not we are proper or not is irrelevant as we are going to add the potential negative block back anyway).

+			-	-		
t	x	s'	r	c'	s	
t	c'	s	t	x	s	
r	x	s	t	c'	s'	
r	c'	s'	r	x	s'	

As you can see, this not only returns column x to its initial state, but it also provides us with the second required block. □

The proof of the previous lemma was quite straightforward, but does not cover circumstances where the square is improper and (r, c', s) does not exist. The next lemma addresses this issue.

Lemma 10. *Suppose that for some proper row t we have (t, c, s) and (t, c', s') . Also, the square is improper with $-(r, c, s)$ (for some $r \neq t$). Further, suppose we have (r', c', s) , for some $r \neq r'$. Then there is a sequence of ± 1 -moves that leaves a working square having (t, c, s') and (t, c', s) but, apart from making this swap in row t , changes incidences in only rows r and r' ; additionally, if the new working square is improper, it has $-(r, c, s')$ or $-(r', c, s')$.*

Proof. The goal of this proof is transform the square so that lemma 9 (case 2) applies. To do this, we need to either exchange (r, c, s) for (r', c, s) or exchange (r', c', s) for (r, c', s) – but not both.

To begin, we must study the discrepancy graph $D_L(r, r')$. This graph is either type 3B or type 3C and therefore there must exist an alternating path between c and s starting and ending on a red edge.

Suppose this alternating path has the form $c, r, s^*, \dots, c^*, r, s$, that is, (r, c^*, s) and (r, c, s^*) both exist. Also note that there exists some row $u \notin \{r, r', t\}$ such that (u, c, s) exists (this is because c and s occur together in a block twice due to them both appearing in the improper block. One of these occurrences is in a block with t , the other cannot be with r as we have $-(r, c, s)$ and the other cannot be with r' because we know that (r', c', s) exists and r' and s cannot occur in another block together). Perform the following move, called “move u ”:

+			-		
r	c	s	u	c^*	s^*
r	c^*	s^*	r	c	s^*
u	c	s^*	r	c^*	s
u	c^*	s	u	c	s

Depending on whether the graph was type 3B or type 3C, the row-pair graph will be transformed into either two or three components respectively (see figure 3.10 for an example where the graph is type 3B; type 3C is analogous).

This altered graph now contains an even length, alternating cycle, denoted H . If the square is in a proper state, we may use lemma 5 to interchange the edge labels of this cycle and reverse move u which leaves us with $-(r', c, s)$ and (r', c', s) , and therefore we may complete the proof by returning to lemma 9, invoking case 2.

If, after performing move u the square is improper, then before we are allowed to interchange the colours of the cycle, we must return to properness.

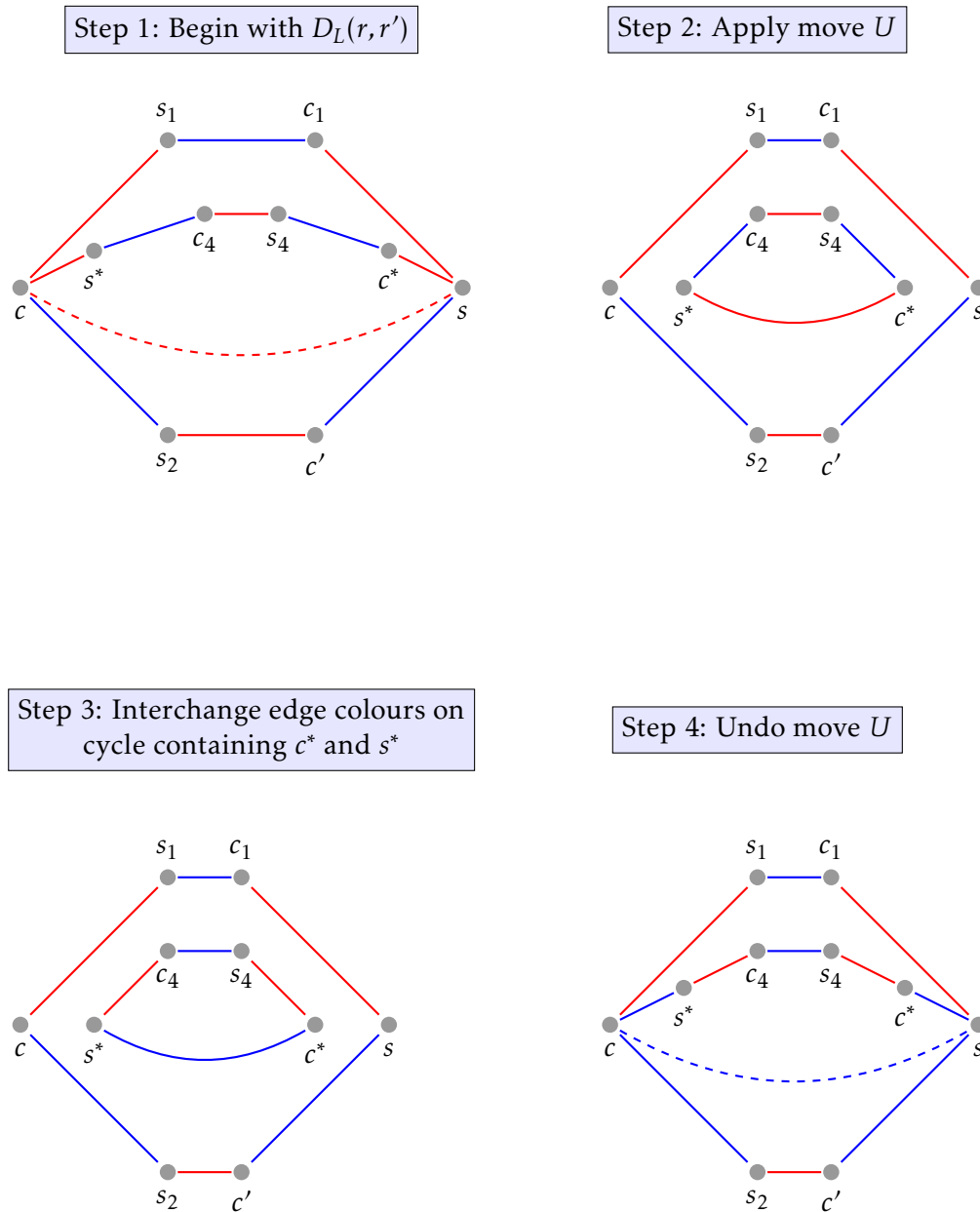


Figure 3.10: If the row-pair graph $D_L(r, r')$ is of type 3B, then it will have a similar form to this graph. Its defining characteristic is that it is 2-connected. In step 2 we perform “move U ” to isolate the edges that we wish to interchange. This is important because we do not want to disrupt (r', c', s) . In step 3 we assume we have managed to get back to a proper square without disrupting any cell in rows r or r' . We interchange the edge labels on exactly one of the components and then in step 4 reverse “move U ”.

If the alternating path that we found had length three, that is (r', c^*, s^*) exists, then after performing move u , the resulting graph is type 2 and we have a 2-cycle. We can immediately reverse move u , but instead of choosing row r , choose row r' , that is, perform the following move:

+			-		
u	c^*	s^*	r'	c	s
r'	c	s^*	r'	c^*	s^*
r'	c^*	s	u	c	s^*
u	c	s	u	c^*	s

Now, as already mentioned, we continue as in lemma 9, case 2.


If the alternating path had size greater than 3, we find $v \notin \{r, r', u\}$ such that (v, c^*, s^*) exists. We can always find such a v because if (u, c^*, s^*) existed, we would not be in this case. Similarly, for (r', c^*, s^*) . Also, as r occurs with c^* in (r, c^*, s) , it could not also occur with c^* and s^* .

Construct the type 3A row-pair graph with v and u . We use lemma 7 and return to properness without damaging rows r and r' . Interchange the edge colours on the cycle H before immediately undoing the damage we caused to rows v and u . Finally, reverse move u to return to case 2 of lemma 9. □

In the next few chapters we shall be generalising the previous proof to show how the ± 1 -move can be used to find other combinatorial structures. For now, let's continue to see how lemmas 9 and 10 are used to move from any given square to any other given square.

Given some proper square W that we wish to transform into some other proper square T , focus on some row r and define a *discrepancy cycle* to be the graph whose vertices are labelled with each symbol and each column. There is a T -coloured edge between a symbol vertex s and column vertex c if (r, c, s) exists in T (similar for W -coloured edges). The discrepancy cycles are a union of disjoint

even length cycles. Any 2-cycle is called trivial because they indicate that there is no work to be done.



The DesignMC package's `CreatePairGraph` function can be used to display these discrepancy graphs.

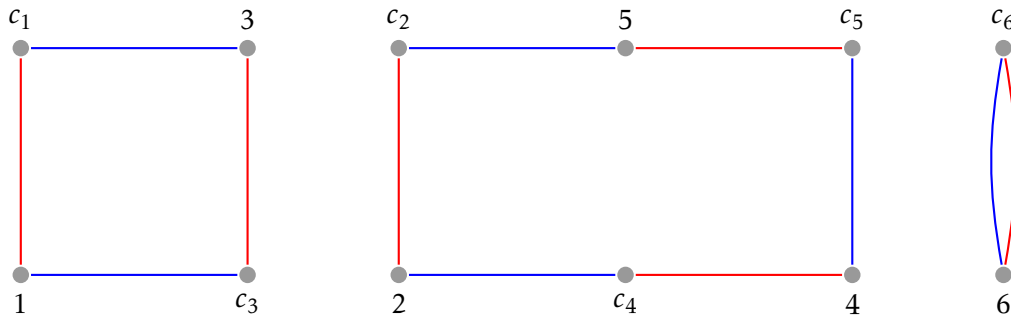
For example, suppose that in square W we have the following entries in row r :

1 2 3 4 5 6

and in square T row r looks like this:

3 5 1 2 4 6

then the discrepancy graph would look like this:



The next lemma states that we can use lemmas 9 and 10 to break a discrepancy graph down into 2-cycles, and thus converting any given square into any other given square.

Lemma 11 ([29], lemma 3). *Given proper working and target squares, select a non-trivial discrepancy cycle in (target) row t . Let R be (the indices of) the minimal set of other working-square rows that must be altered in order to correct this cycle. (That is,*

each T -edge $\{c, s\}$ in the discrepancy cycle indicates a column-symbol incidence that must move to row t from another (working-square) row r' ; R comprises all such r' .) Then there is a sequence of ± 1 -moves that corrects the discrepancies along the cycle, producing a proper square, without changing other incidences in row t or incidences in rows that do not belong to R .

Proof. Omitted. See [29], lemma 3 □

Armed with these new tools, we are finally able to prove that the underlying graph of the ± 1 -move is connected.

Theorem 12 ([29], theorem 1). *Given two (proper or improper) order- n squares, there exists a sequence of ± 1 -moves that transforms one square into the other. An upper bound on the length of the shortest such sequence is $2(n - 1)^3$ ($n \geq 2$).*

Proof. Omitted. See [29], theorem 1 □

Chapter 4

Generalised Latin squares

4.1 Definition

So far we have looked exclusively at the theory of Latin squares. In the introduction to this thesis we said that our primary aim is to address a conjecture that marries other combinatorial structures with the ± 1 -move. The reader may wonder, therefore, why so much emphasis has been placed on Latin squares. The reason for this is that the path through the theory of the other designs is very similar. Where differences occur, the route through Latin squares is often clearer. Hopefully being exposed to Latin squares first makes the upcoming theory easier to follow.

Let's begin the journey into these other combinatorial structures by considering a first generalisation of Latin squares. The definition of a Latin square required an $n \times n$ grid to contain each of the symbols from $\{1, 2, \dots, n\}$ exactly once in each row and column. What happens if, instead of once, we want each symbol to occur exactly twice, with each cell containing exactly two symbols? It is trivial to see that such structures exist because repeating the symbol contained in each cell of a $LS(n)$ satisfies this new definition. Similarly,

1 1	2 2
2 2	1 1

1 2	1 2
1 2	1 2

Figure 4.1: These are the only (up to isomorphism) $LS(2, 2)$.

1 1	3 3	2 2
3 3	2 2	1 1
2 2	1 1	3 3

1 1	3 3	2 2
2 3	1 2	1 3
2 3	1 2	1 3

1 1	2 3	2 3
2 3	1 3	1 2
2 3	1 2	1 3


1 3	1 2	2 3
1 2	2 3	1 3
2 3	1 3	1 2

Figure 4.2: These are the only (up to isomorphism) $LS(3, 2)$.

replacing “twice” with “ $\lambda \in \mathbb{N}$ number of times” is equally trivial to prove.

Formally, this generalised Latin square on n symbols, which contains each of the n symbols in each row and column exactly λ times, is denoted by $LS(n, \lambda)$.

For example, up to isomorphism, there are 2 different $LS(2, 2)$ (figure 4.1) and 4 different $LS(3, 2)$ (figure 4.2).



We obtain these squares using the DesignMC package's

`EnumerateSquares` function.

Up to isomorphism, there are 44 $LS(4, 2)$ and 48568 $LS(5, 2)$ but the computation to discover anything more, such as the number of $LS(6, 2)$ (up to isomorphism), is beyond the computing power of a modern home desktop machine.

This demonstrates that for an even smaller symbol set than Latin squares, we soon reach the combinatorial explosion. Given n and λ , how could we select a square uniformly at random for $\lambda \geq 2$, and n at least, say, 8? Enumeration, hill climbing and cycle switching all fail for the same reasons as before. Jacobson

and Matthews' ± 1 -move was not built to withstand anything but $LS(n, 1)$ – could it perhaps be extended to work with these new designs?

Even more generally, suppose that the number of columns, number of rows and number of symbols may all be different values. This leads us to the most general definition of the Latin square that we shall consider:

4.1.1 Generalised Squares

Let $R = \{\rho_1, \dots, \rho_r\}$, $C = \{\gamma_1, \dots, \gamma_c\}$ and $S = \{\sigma_1, \dots, \sigma_s\}$, and let $\lambda = (\lambda_{RC}, \lambda_{RS}, \lambda_{CS})$ be a triple of positive integers. We call the elements of R, C and S *rows*, *columns* and *symbols* respectively. A *generalised Latin square* (or “square”, for brevity), is an $r \times c$ grid, in which each cell contains λ_{RC} symbols in such a way that every row contains each symbol exactly λ_{RS} times and each column contains each symbol exactly λ_{CS} times. We shall denote these squares as $LS((r, c, s), (\lambda_{RC}, \lambda_{RS}, \lambda_{CS}))$. If $r = c = s$ then we may represent the triple by s . Similarly, if $\lambda_{RC} = \lambda_{RS} = \lambda_{CS}$, then we say the square has *constant* λ and represent the triple by a single value. For example, an $LS(s, 1)$ is the well-studied, and previously defined, Latin square of order s .

Those familiar with the notation of Cameron's generalised t -designs, which we discussed in chapter 2, will recognise the generalised Latin squares as 2- $((r, c, s), (1, 1, 1), (\lambda_{(1,1,0)}, \lambda_{(1,0,1)}, \lambda_{(0,1,1)}))$ designs.

An *improper square* contains some symbol x exactly -1 times in row u column v , denoted $-(u, v, x)$. We refer to this block as the *improper block* or *negative block*. A square that contains $-(u, v, x)$ must have the symbol x occurring in row u exactly λ_{RS} times. This means that there should be $\lambda_{RS} + 1$ proper occurrences of x so that in net there are λ_{RS} in total. Similarly, the proper symbol x should occur $\lambda_{CS} + 1$ times in column v and there should be $\lambda_{RC} + 1$ proper symbols in the improper cell.

To exemplify the use of these objects, we turn to the field of experiment

1 2	3 4	5 6
3 4	5 6	1 2
5 6	1 2	3 4

Figure 4.3: a $LS((3, 3, 6), (\lambda_{RC} = 2, \lambda_{RS} = 1, \lambda_{CS} = 1))$ and also a $(3 \times 3)/2$ semi-Latin square. There are only two (up to isomorphism) such designs with these parameters.

design. An $(n \times n)/k$ semi-Latin square is an $n \times n$ square on nk symbols such that each cell contains n symbols and each symbol occurs in each row and each column exactly once. The square shown in figure 4.3 is a $LS((3, 3, 6), (\lambda_{RC} = 2, \lambda_{RS} = 1, \lambda_{CS} = 1))$ and also a $(3 \times 3)/2$ semi-Latin square.



The `ProduceSquare` function from the DesignMC package can find such designs. For example, if we wanted to find both of the $LS((3, 3, 9), (\lambda_{RC} = 3, \lambda_{RS} = 1, \lambda_{CS} = 1))$ designs, we can use

```
gap> ProduceSquare(rec(v:=[3, 3, 9], lambdas:=[3, 1, 1],
isoLevel:=2));
```

The `isoLevel` parameter, defined in the DESIGN package, can be set to either 0, 1 or 2. Setting `isoLevel:=0` will return exactly 1 design (if any exist). Setting the `isoLevel:=1` guarantees to find a representative from every isomorphism class (but perhaps multiple representatives from a class will appear). Setting `isoLevel:=2` will return exactly one representative from each isomorphism class.

For more information on the theory, uses, constructions and optimality of semi-Latin squares, see [1].

Below is a table displaying the number of squares (both proper and improper) for small values. Information about squares with non-constant λ are

found in the shaded rows of the table.



DesignMC

The DesignMC's `EnumerateSquares` function was used to calculate each of the values in the table shown below.

Proper designs

r	c	s	λ_{RC}	λ_{RS}	λ_{CS}	Total squares
2	2	2	2	2	2	2
3	3	3	2	2	2	4
4	4	4	2	2	2	44
5	5	5	2	2	2	48568
3	3	6	2	1	1	2
3	3	9	3	1	1	2
3	3	3	3	3	3	9
4	4	4	3	3	3	2424

Improper designs

r	c	s	λ_{RC}	λ_{RS}	λ_{CS}	Total squares
3	3	3	2	2	2	2
4	4	4	2	2	2	142
3	3	3	3	3	3	9

Lemma 13. *The size of the row, column and symbol sets are all equal, that is, $r = c = s$, if and only if the design has constant λ , that is, $\lambda_{RC} = \lambda_{RS} = \lambda_{CS}$.*

Proof. Let the number of symbols in row/column j be denoted by $\#(j)$. For any row ρ or column γ we have

$$\#(\rho) = c\lambda_{RC} \text{ and } \#(\gamma) = r\lambda_{RC} \tag{4.1}$$

On the other hand,

$$\#(\rho) = s\lambda_{RS} \text{ and } \#(\gamma) = s\lambda_{CS} \quad (4.2)$$

If $r = c = s$, then by (4.1) we have $\#(\rho) = \#(\gamma)$, and thus by (4.1) and (4.2) we get $r\lambda_{RC} = s\lambda_{RS} = s\lambda_{CS}$, which implies $\lambda_{RC} = \lambda_{RS} = \lambda_{CS}$.

Conversely, if $\lambda_{RC} = \lambda_{RS} = \lambda_{CS}$ then by (4.2) we have $\#(\rho) = \#(\gamma)$, and thus by (4.1) and (4.2) we get $r\lambda_{RC} = c\lambda_{RC} = s\lambda_{CS}$, which implies $r = c = s$. \square

Lemma 14. *An $LS(s, m)$ exists for all $s \geq 2, m \geq 1$.*

Proof. To obtain an $LS(s, m)$, repeat m times the blocks of an $LS(s, 1)$. \square

4.2 Pair Graphs

In section 3.3 we saw how pair graphs behaved for Latin squares. Now we are working with generalised Latin squares, and as a result they have different properties. Let L be a proper generalised Latin square on n symbols, and a, b be rows of L . Recall that the row-pair graph $D = D_L(a, b)$ is a graph with vertex set $V(D) = C \cup S$. We join two vertices x, y with a red edge if (a, x, y) is a block of L . Similarly, we join x and y with a blue edge if (b, x, y) is a block of L .

Pair graphs for proper Latin squares were a disjoint union of even length cycles such that the edge labels alternated as well as the type of vertex (that is, whether the vertex represented a symbol or a column). The graph for generalised Latin squares is slightly more complicated. Firstly, as every cell contains λ_{RC} symbols, the red and blue degree of a column-representing vertex is exactly λ_{RC} , making the degree $2\lambda_{RC}$ overall. Similarly, as every symbol occurs in each row λ_{RS} times, the symbol-representing vertices have degree $2\lambda_{RS}$. Note that row-pair graphs for these generalised designs need not be regular and are certainly not cycles.

To demonstrate the complexity of the pair graph of a generalised square, we can see an example of a square, L , defined to be an

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

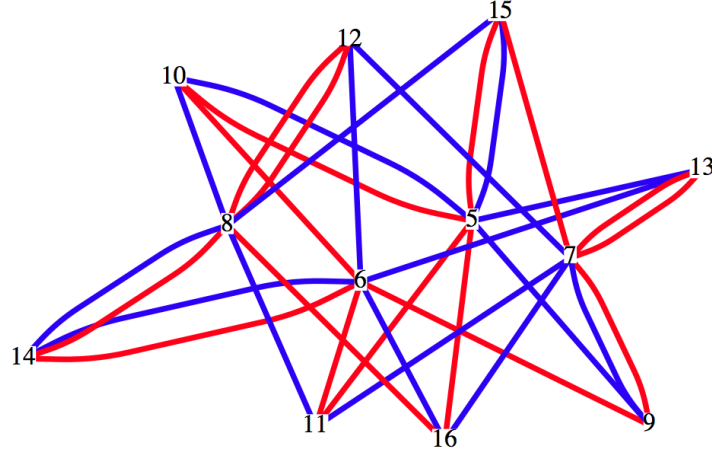


Figure 4.4: a generalised Latin square $L = LS((4, 4, 8), (\lambda_{RC} = 4, \lambda_{RS} = 2, \lambda_{CS} = 2))$ and the row-pair graph $D_L(1, 4)$.

$$LS((4, 4, 8), (\lambda_{RC} = 4, \lambda_{RS} = 2, \lambda_{CS} = 2))$$

and the row-pair graph $D_L(1, 4)$ in figure 4.4.

Although we have only discussed row-pair graphs, and will continue in this way, the reader should observe that the analysis for column- and symbol-pair graphs is analogous.

4.2.1 Square Row-Pair Graph Analysis

Throughout this section, let L be an improper square with the improper block $-(a, c, s)$ and $D = D_L(a, b)$ be any row-pair graph, for some row $b (\neq a)$.

The component of the row-pair graph D containing the special vertices c, s is called the *core* of D , denoted $\text{core}(D)$; if D is connected, then $\text{core}(D) = D$. If we delete some blue edge incident with c , say $\{c, x\}$, from D , we are left with a *near-core* of D , denoted $\text{near-core}(D)$. For every $\text{core}(D)$, there are λ_{RC} $\text{near-core}(D)$ s.

Note that the deleted blue edge is always incident with the column vertex c .

We are interested in a particular type of alternating trail between two vertices $q, r \in V(D)$, starting and ending on a red edge. For brevity, we call this type of alternating a trail an *RBR*. An RBR that starts at q and ends at r is denoted as an $RBR(q, r)$.



To find an RBR between two vertices in a pair graph, use the DesignMC package's `FindAlternatingTrail` function using the `isPathEvenLength` attribute to indicate that the path should have odd length. You can specify a list of vertices that the path must include, as well as a list of forbidden vertices.

Lemma 15. *An $RBR(c, s)$ exists in any connected near-core(D).*

Proof. For now, we set $G = \text{core}(D)$.

Firstly, note that an $RBR(c, c)$ cannot exist in the bipartite graph G , this is because such a trail would necessarily contain an even number of vertices, but an odd number of edges (as the number of red edges is one more than the number of blue edges). However, an odd length cycle must have an odd number of vertices, and thus this trail cannot exist. Similarly, we cannot create an alternating trail that starts on a red edge at c and ends at s along a blue edge.

We are going to create the desired trail by deleting edges from the graph G and keeping track of edges we have deleted. To begin, pick and delete a red edge at c , say, $\{c, w_1\}$. Now w_1 is incident with exactly one less red edge than blue edges and c now has even valency. As w_1 and s are the only two vertices with odd degree, they lie in the same connected component. Pick and delete some blue edge at w_1 , say, $\{w_1, w_2\}$. Now w_2 and s are the only vertices with odd degree, so they must lie in the same connected component. Continue in this fashion until the first instance of either reaching s via a red edge (in which case

the deleted edges form $\text{RBR}(c,s)$), or returning to c via a blue edge. If you are in the latter case, you have a new graph G_1 which you constructed from D by removing an edge-alternating trail starting with a red edge at c and ending with a blue edge at c . As c now has odd valency again, it is connected to s ; you may now repeat the process again. As there are more red edges than blue edges at c , eventually you will begin an edge-alternating trail that cannot return to c , hence you must reach s along a red edge.

Note that this forms an $\text{RBR}(c,s)$ that is guaranteed not to use a blue edge incident with c , so any near-core(D) also contains an $\text{RBR}(c,s)$. \square

The analysis of row-pair graphs for improper generalised Latin squares is not as satisfying as it was for the traditional improper $LS(s,1)$. This need not hinder our progress and in fact, we can still say quite a lot about them. For example, we know they are *semi-Eulerian* (as they only have two vertices of odd degree). If the graph contains more than 1 component, then each component (except the core(D)) is Eulerian. Also, as a corollary of lemma 15, we know that we can find an edge-alternating Eulerian trail from c to s , starting and ending on a red edge. Furthermore, due to the general nature in which we stated and proved lemma 5, we can use the ± 1 -move to interchange the edge colours of any closed alternating trail.

We are now ready to address the question: can Jacobson and Matthews' technique be extended to generate uniformly distributed random generalised Latin squares?

4.3 Generating Squares Uniformly at Random

We have already discussed that a random walk on a finite, connected, non-bipartite, regular, undirected graph is ergodic with a uniform stationary distribution. The theory supporting Jacobson and Matthews' ± 1 -move (which we

described in section 3.3) handles everything except the finite and connectedness conditions. Clearly the graph we are working on is finite because there only a finite number of squares (both isomorphic and non-isomorphic) with given parameters. The only unknown is connectedness, which we shall deal with now. As in section 3.3, we split the lemma in to two parts. The first part contains the relatively easily proved first two cases. The third case, which requires some knowledge of pair graphs is reserved for the lemma that follows afterwards.

Lemma 16. *Suppose that we have a square in which the target row t is proper with (t, c, s) and (t, c', s') , with the additional condition that, if the square is improper, we have $-(r, c, s)$ (for some $r \neq t$). Also, we have (r, c', s) . There is a sequence of ± 1 -moves that leaves a working square having (t, c, s') and (t, c', s) but, apart from making this swap in row t , changes incidences in only row r ; additionally, if the new working square is improper, it has $-(r, c, s')$.*

Before we start the proof, which is of a similar flavour to the proof of 9, we are going to assume that $\lambda_{CS} \geq \lambda_{RC}, \lambda_{RS}$. This assumption makes the proof a little neater, but should not be seen as deviating from generality as “rows”, “columns” and “symbols” are just arbitrary labels that we can permute.

Proof. Case 1: If the square is proper, we can prove the result by performing the following move:

+		-	
t	c	r	c'
t	c'	t	c
r	c	t	c'
r	c'	r	c

As you can see we now have the blocks (t, c, s') and (t, c', s) whilst minimising damage to only row r , which was unwanted anyway.

Case 2: If the square is improper, then we may prove the result by performing two moves. Note that because the square has an improper cell, we are forced to add it.

+			-		
r	c	s	t	x	s'
r	x	s'	r	c	s'
t	c	s'	r	x	s
t	x	s	t	c	s

This move has gained us ‘half’ of our requirements in that we now have (t, c, s') . Unfortunately, we have damaged column x , whatever that may be. We shall fix this damage, and gain the other required cell in the following move (note that whether or not we are proper or not is irrelevant as we are going to add the potential negative block back anyway).

+			-		
t	x	s'	r	c'	s
t	c'	s	t	x	s
r	x	s	t	c'	s'
r	c'	s'	r	x	s'

As you can see, this not only returns column x to its initial state, but it also provides us with the second required block. □

Lemma 17. *Suppose that we have a square in which the target row t is proper with (t, c, s) and (t, c', s') , with $-(r, c, s)$ (for some $r \neq t$) (r', c', s) , for some $r' \neq r$. Then there is a sequence of ± 1 -moves that leaves a working square having (t, c, s') and (t, c', s) but, apart from making this swap in row t , changes incidences in only rows r and r' ; additionally, if the new working square is improper, it has either $-(r, c, s')$ or $-(r', c, s')$.*

Proof. In the original proof, the technique used involved finding an $\text{RBR}(c, s)$ of the form

$$c, red, s^*, blue, \dots, blue, c^*, red, s$$

By lemma 15, we know that such a path exists. Call the shortest such path P and then perform the following move, called “move u ”:

+			-		
r	c	s	u	c^*	s^*
r	c^*	s^*	r	c	s^*
u	c	s^*	r	c^*	s
u	c^*	s	u	c	s

If this new design is improper, we will use lemma 5 to interchange the edge colours of the cycle we just created and return to case 2.

If it is not, we must first return to properness before we can proceed.

If we are improper and P has length 3, then we must have (r', c^*, s^*) , in which case we return to case 2 by performing the following move:

+			-		
u	c^*	s^*	r'	c	s
r'	c	s^*	r'	c^*	s^*
r'	c^*	s	u	c	s^*
u	c	s	u	c^*	s

If we are improper and P has length greater than 3, then we must find some row-representing vertex $v \notin \{r, r', u\}$ such that (v, c^*, s^*) exists. To show that such a v exists, we consider each of the cases separately. Clearly (u, c^*, s^*) does not exist because we have $-(u, c^*, s^*)$. Also, if we had (r', c^*, s^*) , then P would have had length 3, so that cannot be true. Finally, observe that c^* and s^* may occur together $\lambda_{CS} + 1$ as they are both in the improper triple. However, r, s^* may only occur λ_{RS} and r, c^* may only occur λ_{RC} . By assumption, $\lambda_{CS} + 1 > \lambda_{RC}, \lambda_{RS}$.

Therefore there is at least one triple containing c^* and s^* not containing r and hence v exists.

Examine the row-pair graph generated by u and v . We know that (v, c^*, s^*) exists, and by lemma 15, we know there exists an $\text{RBR}(c^*, s^*)$. Hence, there exists a closed, alternating trail that we can use with lemma 7 to return to properness without damaging any other rows except u and v . Now we have a proper square we interchange the cycle that we made earlier in rows r and r' before undoing the damage we just made to rows u and v . Finally, we reverse move u to return us to case two, completing the proof. \square

Using the previous lemma as well as lemma 11 we get the following theorem.

Theorem 18 (D. 2012 [14]). *Given two (proper or improper) $LS((r, c, s), (\lambda_{RC}, \lambda_{RS}, \lambda_{CS}))$, there exists a sequence of ± 1 -moves that transforms one square into the other for any admissible parameter values.*

Having shown connectedness of this graph, we can now generate generalised Latin squares uniformly at random.

The next obvious question is to ask about the efficiency of the method. Like the original case, only heuristic information about this is known, and it is presented in our conclusion in chapter 8.

Chapter 5

Generalised Factorisations

5.1 Definition

Having completely solved the square case, we move on to consider the second type of design in Cameron's conjecture: factorisations.

A *1-factor* (or perfect matching) of a graph G is a set of pairwise disjoint edges of G that are collectively incident with every vertex of G (note that this requires G to have an even number of vertices). We often illustrate a 1-factor of a graph by colouring the associated edges. By doing this, we can describe a 1-factor in block design notation as a set of $\frac{n}{2}$ blocks of the form (v_1, v_2, κ) where v_1, v_2 are vertex labels of G and κ is the colour given to the 1-factor.

A *1-factorisation* of a graph G is a set of disjoint 1-factors whose union contains every edge. To differentiate between the 1-factors, we give each a unique colour.

The *complete graph* is a simple, undirected, graph in which every pair of vertices is joined by an edge. If the graph has n vertices, the degree of each vertex is $n - 1$, and there are $\binom{n}{2}$ edges. We denote this graph by K_n .

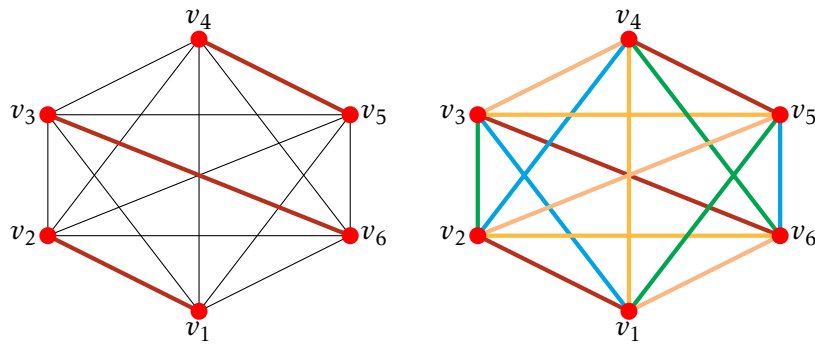


Figure 5.1: (left) A 1-factor of K_6 . The 1-factor (or perfect matching) is indicated by the red edges. (right) A 1-factorisation of K_6 where each 1-factor is represented by a different edge colouring.

Theorem 19 (Harary, [26]). *Any complete graph with an even number of vertices admits a 1-factorisation.*

5.1.1 Counting 1-factorisations

When Are Two 1-Factorisations Different?

We say two (proper or improper) factorisations are isotopic if there exists a permutation of the vertices and colours that transforms one into the other. The points of a Latin square had three roles (rows, columns and symbols) whereas the points the factorisations only have two roles (vertices and colours). Unlike Latin squares, the roles of the points of factorisations are not interchangeable. That is, two factorisations are non-isomorphic if and only if they are not isotopic.

Exact Results

As with Latin squares, picking from an enumerated list is infeasible because the combinatorial explosion hits very early (figure 5.2). For example, it is currently unknown how many 1-factorisations of K_{16} there are.

For a survey on the known theory of 1-factorisations of K_n , see [9]. We are now going to consider generalisations of this design in the same way that we

n	Number of non-isomorphic 1-factorisations of K_n	References
4	1	
6	1	
8	6	
10	396	
12	526915620	J. Dinitz, D. Garnick and B. McKay, 1994 [10]
14	1132835421602062300	P. Kaski and P. Östergård, 2009 [31]

Figure 5.2: A table showing the number of non-isomorphic 1-factorisations for small orders.

handled Latin squares.

5.1.2 Generalised Factorisations

A λ -factor of a graph G is a regular subgraph of degree λ with the property that every vertex is incident with an edge of the subgraph. A λ -factorisation of a graph G is a set of disjoint λ -factors whose union contains every edge. The λ -complete graph is an undirected, regular graph in which every pair of vertices is joined by λ edges. We are interested in discovering if it is possible to generate these designs uniformly at random, but as with Latin squares, Cameron's generalised t -design notation can take us further to get:

5.1.3 Generalised λ_{NC} -factorisation of $\lambda_{NN}K_n$

Let $N = \{v_1, \dots, v_n\}$ and $C = \{\kappa_1, \dots, \kappa_c\}$ and let $\lambda = (\lambda_{NN}, \lambda_{NC})$ be a pair of positive integers. We call the elements of N and C nodes and colours respectively.

A generalised λ_{NC} -factorisation of $\lambda_{NN}K_n$ (or "factorisation", for brevity) is a graph on n nodes with λ_{NN} edges between every pair of nodes. The edges are coloured with elements of C in such a way that every node is incident with exactly λ_{NC} edges of each colour. We denote these factorisations by $F((n, c), (\lambda_{NN}, \lambda_{NC}))$. If $\lambda_{NN} = \lambda_{NC}$, then we say the factorisation has constant

λ and represent the pair by a single value. For example, $F((2n, 2n - 1), 1)$ is the familiar 1-factorisation of K_{2n} .

For any factorisation F , we define the *point set* as $\mathcal{P} = \mathcal{P}(F) = N \cup C$ and if there is an edge between v_1 and v_2 coloured κ , then we say that the *block* $\{v_1, v_2, \kappa\}$ exists (note that we may have multiple copies of a block). The set of all blocks of F is called the *block set*, $\mathcal{B} = \mathcal{B}(F)$. As with squares, factorisations may also contain exactly one improper block.

Lemma 20. *If an $F((n, c), (\lambda_{NN}, \lambda_{NC}))$ exists, then $n\lambda_{NC} \equiv 0 \pmod{2}$ and*

$$c = \frac{(n-1)\lambda_{NN}}{\lambda_{NC}}.$$

Proof. To show $n\lambda_{NC} \equiv 0 \pmod{2}$, count the number of edges coloured κ , denoted $\#e_\kappa$. Using the handshaking lemma, $\#e_\kappa = \frac{1}{2}(n\lambda_{NC})$. This quantity must be an integer, so the numerator must be even and the equality follows.

To show the second equality, we count the number of edges in the graph (denoted $\#e$) in two different ways. Firstly, pick two vertices and look at how many edges there are between them to get $\#e = \frac{1}{2}n(n-1)\lambda_{NN}$. Secondly, pick a vertex and a colour and then look at how many edges of that colour are incident with that vertex, giving $\#e = \frac{1}{2}nc\lambda_{NC}$. The result follows. \square

Lemma 21. *An $F((n, n-1), m)$ exists for all $n \geq 2, m \geq 1$ such that $nm \equiv 0 \pmod{2}$.*

Proof. **For n even:** To obtain an $F((n, n-1), m)$, repeat m times the blocks of an $F((n, n-1), 1)$.

For n odd: By lemma 20, we know that m must be even. To obtain an $F((n, n-1), m)$, repeat m times the blocks of an $F((n, n-1), 2)$. \square

The ± 1 -move for Factorisations

The algorithm for the ± 1 -move as stated in section 3.3 was written in general enough terms to still be correct for factorisations. The only subtlety is that now

an “admissible” cell is one that contains two vertices and a colour.

Jacobson and Matthews’ method does require a factorisation from which it begins randomly walking.



DesignMC

Using the DesignMC package, we can attempt to find a design with given parameters. For example, suppose we would like to find a random $LF((5,8),(4,2))$ design (if any exist). We can use the DesignMC package to look for such a design with the following commands:

```
gap> input:=rec(v:=[5,8], lambdas:=[4,2]);;
```

```
gap> design:=ProduceFactorisation(input);;
```

Once we have found one (there are actually 63 such designs), we can use the following command to iterate the ± 1 -move until we find the n^{th} proper design.

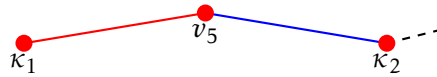
```
gap> ManyStepsProper(design[1], n);;
```

Of course, we have not yet investigated whether it is worthwhile traversing the graph in this way – if the graph is disconnected, then we have no hope of finding a uniform sample. In the next section we shall inspect the behaviour of the pair graph for factorisations.

5.2 Pair Graphs

Throughout this section, let F be an improper factorisation with the improper block $-\{a, v_1, v_2\}$. When we dealt with generalised Latin squares, we observed that we could create a row- column- or symbol-pair graph. We stuck with row-pair graphs for consistency, but it was remarked that this was for no real reason and, if columns or symbols were preferred, the reader may choose to use them instead. However, the situation with factorisations is more delicate. For

example, if we form a pair graph using two vertices, for example v_3, v_4 , and follow an edge-alternating path in this graph, it might be something like:



Notice that the vertices alternate between colour and node. Recall in the proof of lemma 10 we needed to return to a proper square without damaging any rows other than r and r' . We were able to achieve this because we could find a row-pair graph $D_L(u, v)$ (in which the only rows that appear are u and v – the vertices are labelled with columns and symbols) and “slide the chord” back to properness. If we attempted such an act with a node-pair graph, we would damage arbitrary nodes! If we chose to make a pair graph from two colours, then this is not the case. As two colours cannot occur in the same block and all of the edges are labelled with colours, the vertices must all represent nodes. Sliding the chord through such a graph would damage only the two colours that were used to create the graph. It is important in what follows that our pair graph, $D = D_F(a, b)$, be a colour-pair graph where a, b are colours and $b \neq a$.

Chorded VS. Bridged

Unfortunately, this is not a perfect solution either. Let’s take another look at the row-pair graph $D_L(r, r')$ for some improper square L with $-(r, c, s)$ and rows r, r' . The only place edges make any appearance is on the edge labels. The vertices are all labelled with symbols or columns. Further, an edge cannot connect two vertices of the same type. This has important implications for the graph as a whole. In particular, we will never find an $\text{RBR}(c, c)$. This is easy to see – an RBR that starts and ends at c must have an odd number of edges (because the number of red edges is one more than the number of blue edges) but an even number of vertices (because we alternate between column and symbol representing vertices), which is impossible.

If removing some blue edge from a special vertex prohibits the existence of an RBR between the special vertices of a pair graph then we say the graph is *bridged*. Otherwise, we say the graph is *chorded*. All pair graphs for squares are chorded.



Given an improper generalised 2-design with block size 3, the DesignMC package can quickly determine whether an associated pair graph is chorded by using the `IsChordedDG` function.

Returning to the world of factorisations, a node-pair graph $D_F(n, n')$, for some improper factorisation F with $-(n, m, c)$ for nodes n, n', m and colour c , has alternating vertices, and therefore has the property that a $\text{RBR}(m, m)$ cannot exist. This means that there exists an RBR starting from m and finishing at c . Hence, all node-pair graphs are chorded because removing a blue edge from m cannot disconnect the graph.

Chorded graphs have nice properties. For example, using lemma 7 we can transform an improper design to a proper design if we have a chorded pair graph. This is not true of bridged pair graphs. Given a bridged pair graph, sliding the chord can accomplish interchanging edge colours, but cannot return a graph to properness. We can picture what happens by supposing we have two disjoint odd-length cycles each containing one of the special vertices. Now imagine a bridge connecting the two special vertices. Sliding the chord will rotate the cycles, and change the colour of the bridge.

Unfortunately, colour-pair graphs may be bridged. The inability to return to properness derails our method of proving connectedness. However, all is not lost because we may thwart this quandary by considering what happens for different values of λ_{NC} . As we shall see, in lemma 23, if λ_{NC} is even, all colour-pair graphs

are chorded.

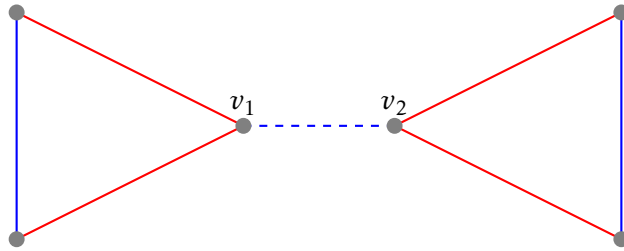
Lemma 22. *The special vertices of D lie in the same component.*

Proof. The special vertices v_1 and v_2 are the only vertices with odd degree, therefore they must lie in the same connected component. \square

Lemma 23. *If $\lambda_{NC} \equiv 0 \pmod{2}$, then any near-core(D) is connected*

Proof. Suppose this is not the case, that is, the edge $\{v_1, x\}$ was a bridge of the core(D). Originally, x and v_1 had an even number of blue edges incident with them (because $\lambda_{NC} \equiv 0 \pmod{2}$). Having deleted $\{v_1, x\}$, they now each have odd degree. In fact, they are the only vertices in their respective components with an odd blue degree, which is impossible. Therefore, $\{v_1, x\}$ cannot be a bridge and as the core(D) was connected, so is any near-core(D). \square

The following picture shows the core of some improper pair graph. The dotted blue edge represents the edge we delete to form a near-core; this proves that the $\lambda \equiv 0 \pmod{2}$ condition in lemma 23 is required.



Lemma 24. *If a near-core(D) is connected, it contains an RBR(v_1, v_2).*

Proof. For brevity, let $E = \text{near-core}(D)$ and recall that we constructed the near-core by deleting some blue edge $\{v_1, x\}$ from the core(D).

Case A: $x \neq v_2$ Starting at v_1 along a red edge, try to construct an RBR(v_1, v_2). As you construct the trail, direct the edges in the direction of travel. One of three things may happen:

Case A1: You complete the RBR(v_1, v_2)

Case A2: You get stuck at x

If you get stuck at x it is because you arrived on a red edge having previously used all of the other available edges. If we had not deleted the blue edge $\{v_1, x\}$, then you could have continued your trail which would have ultimately been successful. With this in mind, temporarily reinstate the deleted edge and complete the $\text{RBR}(v_1, v_2)$; your trail has the form:

$$v_1, \text{red}, \dots, \text{red}, x, \text{blue}, v_1, \text{red}, \dots, \text{red}, v_2$$

To construct a valid $\text{RBR}(v_1, v_2)$ of the near-core(D), simply forget everything in your trail up to, and including, the use of the forbidden blue edge.

Case A3: You get stuck at v_1

Let T be the directed trail that you created; it is an $\text{RBR}(v_1, v_1)$. Note that traversing T in reverse will also yield an $\text{RBR}(v_1, v_1)$. Further, we can arrive at any vertex $y \in T \setminus \{v_1\}$ on either coloured edge by reversing the direction of T .

If $v_2 \in T$, then reversing the direction of the trail must form an $\text{RBR}(v_1, v_2)$, because the original trail had the form:

$$v_1, \text{red}, \dots, \text{blue}, v_2, \text{red}, \dots, \text{red}, v_1$$

and reversing it yields

$$v_1, \text{red}, \dots, \text{red}, v_2, \text{blue}, \dots, \text{red}, v_1$$

which contains an $\text{RBR}(v_1, v_2)$.

We now suppose that $v_2 \notin T$. Consider the subgraph, H , formed by removing the edges of T from E . Note that there is no reason, a priori, to assume that H is connected, so will consider the component H_1 which contains the only two vertices of odd degree (x and v_2). There exists some vertex w in H_1 , with positive degree, that also lies in T . If this were not true, then T would contain every

vertex, (which it does not as $v_2 \notin T$) or H_1 would be disconnected (which we assumed it was not).

As the colours present at each vertex are distributed as fairly as possible H_1 is semi-Eulerian; we can form an $\text{RBR}(x, v_2)$, directing the edges as we go. Now, starting with a red edge at v_2 , move through the trail backwards until you reach w . Suppose you find w along a red (blue) edge, we know that w has had at least one edge of each colour removed (that is, T did not terminate at w) so (possibly by reversing the direction of T), we find a trail from v_1 along a red edge, ending on a blue (red) edge at w ; join this trail with the trail you created from v_2 to w to complete the $\text{RBR}(v_1, v_2)$.

Case B: $x = v_2$

Case B1: You complete the $\text{RBR}(v_1, v_2)$

Case B2: You get stuck at v_1

The reason that this case differs from case A3 is that having deleted a blue edge incident with both v_1 and v_2 , all of the vertices have even degree. Also, the colours do not split as fairly as possible in this graph because v_1 and v_2 have exactly two more red edges than blue edges so this graph is not (semi-)Eulerian.

However, we can still find an $\text{RBR}(v_1, v_2)$ in the following way. Having attempted to create this trail and getting stuck at v_1 , you form an $\text{RBR}(v_1, v_1)$, call this trail T_1 . If T_1 contains v_2 , we can complete the desired trail easily by reversing T_1 as before. So suppose that $v_2 \notin T_1$, then there exists some vertex $w_1 \in T_1$ with positive degree in the subgraph E' formed by removing the edges of T_1 from E . We now turn our attention to forming an alternating trail starting from v_2 on a red edge, to w_1 in the graph E' ; call it T_2 . If we find such a trail, then we set the direction of T_1 in such a way that will allow us to hop from T_1 to T_2 at w_1 , completing an $\text{RBR}(v_1, v_2)$. However, it could be the case that you get stuck at v_2 before finding w_1 creating an $\text{RBR}(v_2, v_2)$. Now consider the subgraph formed by removing the edges of T_2 from E' – call this E'' ; every vertex has even

degree, with the blue degree and red degree equal for all vertices – this graph is Eulerian. Let $w_2 \in T_2$ be a vertex with positive degree in E'' .

If E'' is connected, then form an Eulerian trail; this will result in forming a directed trail between w_1 and w_2 ; denote this trail W . Now orient T_1 and T_2 so that we may move from v_1 to w_1 along T_1 , w_1 to w_2 along W and finally w_2 to v_2 along T_2 , completing the $\text{RBR}(v_1, v_2)$

Before considering if E'' is disconnected, note that if E' is disconnected, each component must intersect T_1 . Consider the component containing v_2 from which we removed the edges of T_2 to form E'' . Now, if E'' is disconnected, T_2 must intersect each part of it. This means that there is at least one component through which both T_1 and T_2 pass; this component, S , is Eulerian so we may form an alternating trail W , from $w_1 \in T_1 \cap S$ to $w_2 \in T_2 \cap S$. Now orient T_1 and T_2 so that we may move from v_1 to w_1 along T_1 , w_1 to w_2 along W and finally w_2 to v_2 along T_2 , completing the $\text{RBR}(v_1, v_2)$, and the proof. \square

5.3 Generating Factorisations Uniformly at Random

In the following we enforce that $\alpha, \beta, \gamma, \delta$ and σ represent colours and require $\lambda_{NN} \geq \lambda_{NC}$ and $\lambda_{NC} \equiv 0 \pmod{2}$. In the proof, we will use the notation $\lambda_{\gamma h}$ to mean “the λ value which determines how many times points of the same type as γ may occur with points of the same type as h ”.

The next lemma (which is a generalisation of Lemma 2 in [29]) shows that if we have two blocks such as $\{\gamma, v_1, v_2\}$ and $\{\gamma, v_3, v_4\}$, we can perform an operation to exchange them for $\{\gamma, v_1, v_4\}$ and $\{\gamma, v_2, v_3\}$ limiting all other damage to blocks containing two other points.

The first two cases of the proof are essentially the same as those contained in both lemma 9 and lemma 16 so we omit the details here.

Lemma 25. *Let J be either any square, or any factorisation satisfying the above conditions. Suppose that the proper blocks $\{\gamma, v_1, v_2\}$ and $\{\gamma, v_3, v_4\}$ exist in $\mathcal{B}(J)$. For*

some $\beta \neq \gamma$ there exists $\{\beta, v_1, v_4\}$ and if we are improper, then we have $-\{\alpha, v_1, v_2\}$.

Then there exists a sequence of Markov chain moves which results in $\{\gamma, v_1, v_4\}$ and $\{\gamma, v_2, v_3\}$, but apart from this swap in γ , the only other triples affected are coloured β or α .

Proof. **Case 1:** J is proper.

(omitted)

Case 2: J is improper and $\alpha = \beta$

(omitted)

Case 3: J is improper and $\alpha \neq \beta$

In this situation we have the blocks $\{\gamma, v_1, v_2\}, \{\gamma, v_3, v_4\}, -\{\alpha, v_1, v_2\}$, and $\{\beta, v_1, v_4\}$. We proceed by exchanging $-\{\alpha, v_1, v_2\}$ for $-\{\beta, v_1, v_2\}$, which allows us to return to the comfort of case 2.

To begin, consider the pair graph $D = D_F(\alpha, \beta)$. We want to construct an $\text{RBR}(v_1, v_2)$ in $\text{near-core}(D)$ created by deleting the blue edge $\{v_1, v_4\}$ from $\text{core}(D)$. We know that the near-core is connected (by lemma 23) and therefore lemma 24 ensures the existence of the desired trail. Let the $\text{RBR}(v_1, v_2)$ we created be denoted by T ; it has the form:

$$T = v_1, \alpha, g, \beta, \dots, \beta, h, \alpha, v_2$$

If we perform the following move, for some $\sigma \in \{\alpha, \beta\}$ such that $\{\sigma, v_1, v_2\}$ exists, we make an alternating, closed, trail Y in the $D_F(\alpha, \beta)$ graph, not containing $\{\beta, v_1, v_4\}$.

	+			-	
α	v_1	v_2	σ	h	g
α	h	g	α	v_1	g
σ	v_1	g	α	h	v_2
σ	h	v_2	σ	v_1	v_2

If we can interchange the edge labels on Y (using lemma 5) and then reverse the previous move, we will result in $\{\beta, v_1, v_4\}$ and $-\{\beta, v_1, v_2\}$, and therefore we are in case 2. However, we cannot necessarily proceed with the interchange because we may have an improper triple $-\{\sigma, h, g\}$ that forces our hand with regard to the next move we must make. If $\{\beta, h, g\}$ exists, then simply perform the following move (*) to return to case 2:

+			-		
σ	h	g	β	v_1	v_2
σ	v_1	v_2	σ	h	v_2
β	h	v_2	σ	v_1	g
β	v_1	g	β	h	g

If it does not exist and we are improper, then we first move back to properness in the following way. Find some $\delta \notin \{\alpha, \beta, \sigma\}$ such that $\{\delta, h, g\}$ exists. (We can do this: obviously $\{\beta, h, g\}$ does not exist, otherwise we would have been returned to case 2 already. Also, $\{\sigma, h, g\}$ does not exist, otherwise we would not have $-\{\sigma, h, g\}$ and would already be proper and could make the interchange of Y easily. So the only case to consider is that every block containing both g and h contains α . We consider each of the two designs separately to show that this is not possible. The number of times g and h can occur together is $\lambda_{NN} + 1$ and the number of times α and g can occur together is λ_{NC} times, which is at most λ_{NN} , by assumption. Hence there must be some triple containing g and h not containing α .)

We would like to return to properness by “sliding the chord” through the pair graph $D_I(\delta, \sigma)$. To do this, find an RBR(g, h) in the near-core(D) created by deleting $\{\delta, g, h\}$ (using either lemma 15 or 24). Using this alternating trail together with $\{\delta, g, h\}$ forms a closed, alternating trail in the original pair graph which we use (along with lemma 7) to return to properness; call this “move X ” and note that it does not affect any block incident with α or β . As we now have a

proper design, use lemma 5 to interchange the colours on Y .

Finally, reverse move X and perform move $(*)$ to return to case 2. \square

By using the same reasoning found in [29], we get the following result to prove that the underlying graph of the Markov chain is connected.

Theorem 26. *Given two (proper or improper) factorisations with the same parameters and $\lambda_{NN} \geq \lambda_{NC}$ and $\lambda_{NC} \equiv 0 \pmod{2}$, there exists a sequence of Markov chain moves that transforms one into the other.*

Finally, we are able to address part of the second third of Cameron's Conjecture with the following:

Theorem 27 (D. 2012 [14]). *The Jacobson and Matthews Markov chain is able to generate admissible generalised factorisations with $\lambda_{NN} \geq \lambda_{NC}$ and $\lambda_{NC} \equiv 0 \pmod{2}$ uniformly at random.*

5.4 Experimental Results

Unfortunately, we have not been able to prove the general result for 1-factorisations of K_n . However, using the DesignMC package, we have verified that the ± 1 -move is able to generate these designs uniformly at random for $n \in \{4, 6, 8, 10\}$. Below is a complete list of parameter values that have been tested. A question mark appearing in the "Connected?" column indicates that the test was inconclusive due large number of designs.

Proper designs

n	c	λ_{NN}	λ_{NC}	Total factorisations	Connected?
4	3	1	1	1	✓
6	5	1	1	1	✓
8	7	1	1	6	✓
10	9	1	1	396	✓
12	11	1	1	526915620	?
14	13	1	1	1132835421602062300	?
4	3	3	3	5	✓

Improper designs

n	c	λ_{NN}	λ_{NC}	Total factorisations	Connected?
6	5	1	1	1	✓
8	7	1	1	22	✓

Chapter 6

Generalised Triple Systems

6.1 Definition

The final type of design that we shall work with is a generalisation of the Steiner triple systems. A *Steiner triple system of order v* (abbreviated to $\text{STS}(v)$) consists of a point set \mathcal{P} of size v , and a block set \mathcal{B} . The elements of \mathcal{B} , called blocks, are 3-subsets (hence “triple”) of \mathcal{P} with the property that any distinct pair of points occurs in exactly 1 block. If we want to change the property that every pair of points occurs in exactly one block to something more general like “every pair of points occurs in exactly λ blocks” then we just use the more generic term *λ -triple system of order v* (abbreviated to $\text{TS}(v, \lambda)$). It is worth noting that unlike squares or factorisations that we saw in earlier chapters, triple systems have just one type of point. This lack of discrimination amongst the points will become more important later.

As a titbit for readers interested in t -design terminology, we note that it is appropriate to use classical t -design language to describe a $\text{TS}(v, \lambda)$ because the generalised t -design version is actually the same design. That is, the generalised 2 - $((v), (3), (\lambda_t))$ is the classical 2 - $(v, 3, \lambda)$ design.

The earliest recorded work on triple systems was contributed by Kirkman in 1847. Working with Steiner triple systems (the case where $\lambda = 1$), he proved the following:

Theorem 28. *A $TS(v, 1)$ exists if and only if $v \equiv 1, 3 \pmod{6}$.*

More generally, the following is also well known:

Theorem 29. *A $TS(v, \lambda)$ design exists only if:*

- $v \equiv 1, 3 \pmod{6}$ (for any value of λ);
- $v \equiv 0, 4 \pmod{6}$ and λ is even;
- $v \equiv 2 \pmod{6}$ and $\lambda \equiv 0 \pmod{6}$;
- $v \equiv 5 \pmod{6}$ and $\lambda \equiv 0 \pmod{3}$;

These designs are attributed to the more famous Jacob Steiner who unwittingly rediscovered them in 1853 [42]. Kirkman was not completely unacknowledged and he does have a very special class of Steiner triple systems named after him. A *Kirkman triple system* is a Steiner triple system with the added property that the blocks may be partitioned so that the union of the blocks in each part contains each point exactly once. We call a design with this property *resolvable*. This may be thought of as the analogue of a transversal decomposition in a Latin square.

Two triple systems are *isomorphic* if and only if there exists some permutation that transforms one into the other. The smallest Steiner triple system (with a non-empty point set) has just 1 block containing 3 points. The next smallest is also unique (up to isomorphism) and has 7 points and 7 blocks. The $TS(7, 1)$ is often referred to as the Fano plane, which is a graphical representation of this system (see the discussion in chapter 2).

The final definition we need is that of an *improper* triple system, which is triple system with a benevolent disrespect of the rules. An improper triple

system may contain some block $b \in \mathcal{B}$ that occurs -1 times; we call this block the *negative block* and put a minus sign in front of it to differentiate it from the other “positive” blocks. Suppose that we have a TS with $-(1, 2, 3)$. As every pair of points must occur amongst the block set exactly λ times, there must be $\lambda + 1$ other blocks that contain the points $(1, 2)$, $(1, 3)$ and $(2, 3)$.

6.1.1 Counting Triple Systems

As with the other designs we have studied in this thesis, counting the number of triple systems is not an easy task. As the table below indicates, the combinatorial explosion occurs very early on. With just one type of point, two systems are *isomorphic*, that is, “different”, if and only if there exists some permutation that transforms the point set of one into the other.



DesignMC

The DESIGN and DesignMC packages allow us to calculate the number of proper or improper systems (up to isomorphism) for small values with the `ProduceTripleSystem` function. For example, if we want to know how many proper STS(15) there are (up to isomorphism), we would type

```
gap> designs:=ProduceTripleSystem(rec(v:=[15], lambdas:=[1],
isoLevel:=2));
gap> Size(designs);
```

80

The `ProduceTripleSystem` accepts an `improper:=true` parameter to be passed in, but some of the DESIGN package functions are currently unable to handle any designs that are not *binary*, that is, designs that allow a point to occur more than once in a block. Therefore, users should not use this feature for $\lambda > 1$ until a future release of the DESIGN package.

The reader may like to verify that there is no improper Steiner triple system on less than 9 points. The smallest improper system, (which is unique up to isomorphism) has 9 points.

Here is a table that shows the number of $TS(v, \lambda)$ (up to isomorphism) for $v \geq 7$. Apart from $T(8,6)$, $T(19,1)$, these numbers were calculated using the DesignMC and DESIGN packages.

Proper designs

v	λ	Total triple systems
7	1	1
7	2	4
7	3	10
7	4	35
7	5	109
7	6	418
8	6	3077244
9	1	1
9	2	36
9	3	22521
10	2	960
13	1	2
15	1	80
19	1	11084874829

Improper designs

v	λ	Total triple systems
9	1	1
13	1	50
15	1	21004

6.2 Pair Graphs

In section 5.2 we introduced the concept of bridged and chorded discrepancy graphs. Consider a pair graph for an improper triple system. Firstly, we notice that in creating a pair graph, we do not have to worry about which “type” of points we use; triple systems only have one type. This simplification, which at first glance appears to make things easier, is devastating. For example, having just one type of point means that the core of any pair graph might be bridged, making an analogue of lemmas 7 and 15 impossible. This alone does not necessarily prevent us from proving that we can generate triple system uniformly at random using Jacobson and Matthews’ ± 1 -move. For if we manage to find a path between every pair of triple systems without encountering, or circumventing, pair graphs that not bridged we could proceed.

In part of the proof of lemma 10, it was necessary to return to properness without damaging certain points. The goal was to interchange the edge labels of a cycle before undoing the damage caused by returning to properness. The net result is exactly the same as what we started with, except for the cycle whose edges colours we interchanged. We were able to do this by “protecting” the row labels as we returned to properness. For example, if we want to protect two rows r, r' , then we simple create some other row-pair graph with p, p' which never contains any other row labels. This would be impossible to enforce with triple systems because we have no method of protecting points. This means that even if we were able to find the necessary RBRs and return to properness because we circumvented the bridged graph problem, we still could not proceed with the

proof unless we never needed to return to properness without arbitrary damage to the system (this could actually happen if we always found an RBR of length 3).

All of this suggests that a significantly different method of proof would be required to show that the ± 1 -move could succeed. In the next section, we will see evidence in favour of the existence of such a proof.

6.3 Evidence Supporting the ± 1 -move

Although we do not know for sure whether or not the ± 1 -move connects the space of Steiner triple systems, there is mounting evidence to suggest that it probably does. In this section we shall present this evidence.

Isomorphic Systems Are Connected

Two triple systems S_1, S_2 are isomorphic if and only if there exists some permutation π that, when applied to the point set of S_1 results in S_2 .

Theorem 30 ([15]). *If S_1, S_2 are isomorphic $TS(v, 1)$, then there exists a sequence of ± 1 -moves that converts S_1 into S_2 .*

Proof. Consider the permutation π that transforms S_1 into S_2 . Any permutation may be expressed as a product of transpositions and therefore $\pi = (a, b)(c, d) \cdots (y, z)$.

Consider the first transposition (a, b) and form the pair graph $D_{S_1}(a, b)$. If we were able to interchange all of the edge labels of this graph, then we would have effectively applied the permutation (a, b) to S_1 . Further, we know that we can perform this action because the pair graph is Eulerian, and therefore we can use lemma 5.

Continue to apply each transposition until you have applied the whole permutation and have obtained S_2 . □

Note that in the proof we didn't actually require a $TS(v, 1)$; any generalised 2-design with block size 3 would have worked due to the generality of lemma 5.

Interchanging the edge-labels of a single cycle using lemma 5 is known as *cycle trading*. In the previous proof we traded every cycle in the pair graph. If, instead, we only trade some of the cycles then in general we find non-isomorphic systems. Note that if $\lambda = 1$, then each cycle is its own component, but if $\lambda > 1$, then we can alter a cycle in some component, leaving other cycles fixed.

Connectivity Results for Small Parameter Values

As isomorphic systems are now known to be connected, we can easily verify that for $v \leq 13$ the ± 1 -move connects the space of $TS(v, 1)$. Further, thanks to the joint efforts of [21] and [24], we can also report that by just using cycle trades, the graph of all 80 $TS(15, 1)$ are connected by ± 1 -moves. In practise, it would be preferable to simply enumerate all 80 systems and pick one at random and then apply a random permutation to it. However, it is important encouragement to know that these systems are so easily connected. Finally, thanks to the remarkable efforts of [30], we also know that the $TS(19, 1)$ are also connected just by using cycle trades! There are over 11 billion different systems and we are now at the limit of what can be enumerated. The proof used a computer search, so there is little hope of improving this result for $v \geq 21$, even with a few more years of computing power.

Unfortunately, all Steiner triple systems of order v cannot be connected by cycle trades alone due to the existence of so-called perfect systems. A Steiner triple system is *perfect* if all its pair graphs consist of only 1 cycle. Perfect triples are rare and in fact, only a finite number of perfect systems are known [23], [18]. Clearly any cycle trading with such a system leads to an isomorphic system. The smallest non-trivial perfect system exists when $v = 25$. In [15], it is shown that this system can easily be transformed into a non-perfect system by using ± 1 -

moves.

Trades

Let T_1 and T_2 be sets of n blocks of some triple system. Furthermore, suppose that if x, y are points incident in λ blocks of T_1 , then x, y are also incident in λ blocks of T_2 and vice versa. A *trade* $T = \{T_1, T_2\}$ is an operation that we may apply to a block set to remove all of the blocks contained in T_1 and insert all of the blocks contained in T_2 , leaving another triple system. The set of points that a trade covers is known as the *foundation* of the trade, and the number of blocks a trade affects is called the *volume* of the trade. The smallest configuration that can be traded is known as the *Pasch configuration*, or *quadrilateral*, which has foundation 6 and volume 6 (figure 6.1). If we set $T_1 = \{(a, b, c), (x, y, c), (x, b, z), (a, y, z)\}$ and $T_2 = \{(x, y, z), (x, b, c), (a, y, c), (a, b, z)\}$, then $T = \{T_1, T_2\}$ is a trade.

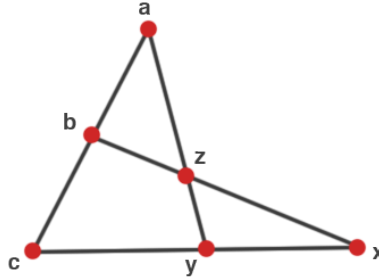


Figure 6.1: A Pasch configuration. Each line represents a block. If a vertex is incident with a line, then that point is in the block.

It has been shown that by only Pasch switching from a given STS, one cannot discover every other STS on the same number of points because there exist systems which do not contain a Pasch configuration; these are known as *anti-Pasch systems* [22]. However, there are many other tradable configurations that could be used instead. Forbes has a list of over 100 trades with volume at most 10 [17] which Grannel and Griggs proved all but one could be affected by the ± 1 -move! Clearly this result puts pressure on the graph to be connected. The

trade that could not be affected (number 68 in Forbes' list) was indicative of a broader problem.

A *partial triple system* consists of a set of v points which lie in 3-subsets, called blocks, such that for each pair of points there exists at most 1 block containing them. One may easily create partial triple systems by deleting blocks from a triple system, however not every partial triple system has this form. That is, some partial triple systems cannot be completed to a triple system. Given a block (a, b, c) , the *orbit* on the point set \mathbb{Z}_v is the set $\langle a, b, c \rangle_v = \{(a + i, b + i, c + i) : i = 0 \dots v - 1\}$ (where addition is modulo v). An orbit is *suitable* if a, b, c are distinct and no pair of points is repeated amongst the distinct blocks of the orbit.

Lemma 31. ([15], lemma 3.1) *Suppose that $T_1 = \langle 0, a, b \rangle_v$ is a suitable orbit of v distinct blocks and that $T_2 = \langle 0, b - a, b \rangle_v$, so that $\{T_1, T_2\}$ is a trade pair. Suppose also that none of the following relationships hold in \mathbb{Z}_v :*

$$3a = 0, 3b = 0, b = -2a, b = 3a, a = -2b, a = 3b, 2b = 3a, 2a = 3b, 3a = 3b \quad (6.1)$$

Then there does not exist a sequence of ± 1 -moves that transforms the partial triple system T_1 into T_2 .

This lemma is by no means a deal breaker – given a triple system whose only differences are that one contains $\langle 0, a, b \rangle_v$, and another that contains $T_1 = \langle 0, b - a, b \rangle_v$, we still might be able to find a sequence of ± 1 -moves that connects them, it will just require temporary damage to blocks outside of the configuration. In any case, we shall offer further methods to attack this issue in the next chapter.

A Failed Attempt

It may be instructive to hear of a failed attempt that the author has investigated. Because we cannot distinguish between the points, an attempt was made to see if it was possible to convert one system into another by fixing the smallest “bad” point first. That is, given two Steiner triple system S_1, S_2 on the points $\{1, 2, \dots, v\}$, we can always apply some permutation to enforce that the blocks of S_1 and S_2 containing a 1 coincide. We can use the isomorphism result to enact this manoeuvre. Now look at the lowest point k such that the blocks of S_1 and S_2 containing this point do not coincide. Suppose there is some block $b = (h, j, k)$ which is in S_1 but not S_2 . If we add b to S_2 and result in a proper triple system, then we can continue by adding another block. If we result in an improper system, then we want to return to properness as soon as possible. The first issue can occur if λ is odd because we can get bridged pair graphs that cannot be returned to properness. For this reason, let us enforce that λ is even. Now all pair graphs may be returned to properness, but with a complete lack of respect for other points. For example, suppose we have fixed all the points less than, say, 10. When returning to properness, we might damage any number of these points and it seems difficult to coerce the movement to be non-damaging. Increasing λ to higher values seems likely to work because (for a “typical” system) we create many more closed alternating trails that we could use that might not contain a fixed point.

Chapter 7

Decomposing Latin Rectangles

A *diagonal* of a $LS(n)$ is a set of n cells, no two of which share a column or a row. A *transversal* is a diagonal with the added property that the union of the cells contain no symbol twice. The cyclic squares of even order never contain a transversal [45], whereas the number of transversals in cyclic squares of odd order have been shown to be at least exponential in n [8].

Transversals are the source of many open problems, the most famous of which is the following revised conjecture of Ryser.

Conjecture 32 (Ryser). *Every Latin square of odd order contains a transversal.*

Ryser originally conjectured that the number of transversals in a $LS(n)$ was congruent to $n \pmod{2}$. For the even case, this was shown to be true by Balasubramanian [3]. However, for the odd case there are many counter examples, and therefore the conjecture was weakened.

As some squares do not contain any transversals, one may instead consider a *partial transversal* of size $s \leq n$, which is a collection of s cells, no two in the same row or column, and no two containing the same symbol. However, even proving that a partial transversal of size $n - 1$ exists in all Latin squares is a difficult

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

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

Figure 7.1: On the left, we see a LS(9) with a diagonal highlighted. It is not a transversal because some symbols in it occur more than once. On the right, we see another diagonal that doesn't contain any symbol twice, therefore it is also a transversal.

task; the following conjecture, which is true for cyclic Latin squares, has been unresolved in general for over 30 years ([28], page 103).

Conjecture 33 (Brualdi). *Every LS(n) has a partial transversal of size $n - 1$.*

On the other hand, some LS(n) have the special property that not only do they have a transversal, they have n mutually disjoint transversals, which we call a *transversal decomposition*.

Given a transversal decomposition, suppose we colour each cell with a unique colour depending on the transversal in which it appears. By construction, each symbol occurs with each colour exactly once. As cells in a transversal cannot occur in the same row or column, the reader may notice that the colours themselves form a second Latin square (see figure 7.2); we call these two Latin squares *mutually orthogonal* as every symbol occurs with every colour exactly once.

After failing to produce two mutually orthogonal Latin squares (MOLS) of order 6, Euler famously conjectured in 1782 that for $n \equiv 2 \pmod{4}$, there cannot exist two MOLS. Although he was proven correct in 1901 for $n = 6$ [43], the conjecture is was ultimately shown to be false for $n \geq 10$ by Bose, Shrikhande and Parker [5].

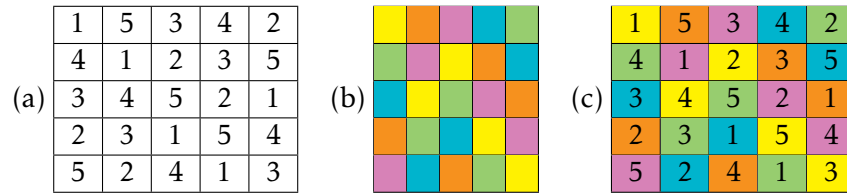


Figure 7.2: (a) and (b) are mutually orthogonal Latin squares (MOLS) of order 5; in square (b) we have represented the symbols as colours, rather than the conventional $\{1, 2, \dots, n\}$. Square (c) is a superposition of (a) and (b) exhibiting a transversal decomposition.

7.1 Decomposing Latin Rectangles

Let R be a $k \times n$ grid with the property that each cell of R contains exactly one element from the symbol set $S = \{1, 2, \dots, n\}$. R is a *Latin rectangle* if every symbol occurs in each row exactly once, and at most once in each column.

At the 13th British Combinatorics Conference, Hilton posed the following interesting problem that marries the topics of Latin rectangles and partial transversal decomposition.

Question 34 (Hilton [27]). *Let R be an $n \times 2n$ Latin rectangle on $2n$ symbols. Is it true that R can be expressed as the union of $2n$ partial transversals of size n ?*

A positive resolution of this problem would also resolve the following question of Wanless.

Question 35 (Wanless [46]). *Let L be an $LS(2n)$. Is it true that L may be decomposed into $4n$ partial transversals of length n .*

For large n , Häggkvist and A. Johansson have shown that for any $\varepsilon > 0$, every $(n - \varepsilon n) \times n$ Latin rectangle can indeed be decomposed into partial transversals of length n [25].

In the rest of this section, we shall prove the following theorem:

Theorem 36. *If R is an $\frac{n}{3} \times n$ Latin rectangle on n symbols with $n < 15$, then R may be decomposed into n partial transversals of length $\frac{n}{3}$.*

Although the result is relatively modest, the method of proof is new and may perhaps be extended to resolve further cases. The first tool we need is the *poach move*. A cell that is currently not associated with (or covered by) any transversal, may poach a transversal association from some other cell in the same row that is covered.

Lemma 37. *Let R be an $k \times n$ Latin rectangle on n symbols with an incomplete decomposition into partial transversals of length k . Any uncovered cell in row r may poach a transversal association from at least $n - 2k + 2$ cells row r .*

Proof. Suppose $u = (r, c, s)$ is the uncovered cell that we would like to cover.

The cell u cannot poach from a cell covered by a transversal that occurs in column c ; if it did, that transversal would occur in column c twice.

Similarly, u cannot poach from a cell that is covered by a transversal that already contains the symbol s ; if it did, s would occur twice in this transversal.

Therefore, there are at least $n - (k - 1) - (k - 1) = n - 2k + 2$ cells in row r from which u can poach. □

Corollary 38. *Any cell may be poached by at least $n - 2k + 2$ other cells in the same row.*

Proof. Suppose $u = (r, c, s)$ is the covered cell that we would like to uncover.

The cell u cannot be poached by a cell whose column contains the transversal with which u is associated; if it did, the transversal would occur twice in that column.

Similarly, u cannot be poached by a cell that is covered by its transversal already; if it did, that symbol would occur twice in the transversal.

Therefore, there are at least $n - (k - 1) - (k - 1) = n - 2k + 2$ cells in row r to which can poach from u . □

Using poaching, we are able to quickly prove the following result:

Theorem 39. *Let R be a $k \times n$ Latin rectangle on n symbols with some mutually disjoint partial transversals identified. We can redistribute the transversal associations in R to find an uncovered diagonal if $k < \frac{n+2}{3}$.*

Proof. Let A be a set containing one (arbitrarily selected) uncovered cell from each row. Assuming that there are no uncovered diagonals, there exists cells $a_1 = (r, c, s), a_2 = (r', c', s') \in A$ such that $c = c'$ or $s = s'$, that is, a pair of cells share a column or symbol.

We identify some cell in row r that does not share a column with any cell in A (note that this is always possible as there are n choices in total, at most $k - 1$ columns could be unsuitable, leaving at least $n - k + 1$ suitable cells). We know that a_1 may poach from $n - 2k + 2$ cells (by lemma 37). If $(n - k + 1) + (n - 2k + 2) > n$, that is, $k < \frac{n+2}{3}$, then we are guaranteed to find a suitable cell to uncover in this row.

Repeat the procedure as required in each row until a diagonal is exposed. \square

By considering the case when $k = \frac{n}{4}$, we shall now verify the result of Häggkvist and A. Johansson with the poach move as a warm up for proving theorem 36.

Theorem 40. *If R is an $\frac{n}{4} \times n$ Latin rectangle on n symbols, then R may be decomposed into n partial transversals of length $\frac{n}{4}$.*

Proof. Begin by greedily finding as many partial transversal as you can. Suppose you found $0 \leq i \leq n$ partial transversals of length $\frac{n}{4}$. Let A be a set containing one (arbitrarily selected) uncovered cell from each row. As there were no partial transversals left, there exists cells $a_1 = (r, c, s), a_2 = (r', c', s') \in A$ such that $c = c'$ or $s = s'$, that is, two cells that share a column or symbol.

We identify some cell in row r that does not share a column or symbol with any cell in A (note that this is always possible as there are n choices in total, at most $\frac{n}{4} - 1$ columns could be unsuitable as could at most $\frac{n}{4} - 1$ symbols, leaving

at least $\frac{n}{2} + 2$ suitable cells). Let ρ be one of these suitable cells. Now, if a_1 can directly poach from ρ , we are done.

If not, then find the cell α such that α can poach from ρ , and a_1 can poach from α – note that α exists because there are $\frac{n}{2} + 2$ cells that may poach from ρ (by lemma 37) and a_1 can poach from $\frac{n}{2} + 2$ cells (by lemma 37).

If A now contains a partial transversal, we are done, otherwise, repeat this process with another pair of cells for each row until A does contain a transversal. Once A contains a transversal, we cover those cells and if we still have not decomposed the rectangle, iterate the whole process again with a new selection of uncovered cells for A . □

The proof of theorem 36 is only slightly more complicated, and just requires a little more insight given in the following lemma.

Lemma 41. *Let R be an $\frac{n}{3} \times n$ Latin rectangle on n symbols with some mutually disjoint partial transversals identified. Also suppose that a_1 and a_2 are uncovered and share the same symbol. Let U be a set of size $\frac{n}{3} + 2$ of covered cells in the same row as a_1 . If a_1 cannot poach from any cell in U , then:*

1. *There exist at least 3 cells in U whose transversals do not occur in the same column as a_1 ;*
2. *There exist at least 4 cells in U whose transversals do not contain the same symbol as a_1 .*

Proof. At most $\frac{n}{3} - 1$ of the cells in U may be members of transversals that pass through the column containing a_1 ; as $|U| = \frac{n}{3} + 2$, at least 3 cells may not.

At most $\frac{n}{3} - 2$ of the cells in U may be members of transversals that containing the same symbol as a_1 (remember that a_2 is also uncovered and shares the same symbol as a_1); as $|U| = \frac{n}{3} + 2$, at least 4 cells may not. □

We are now ready to prove theorem 36.

Proof of theorem 36. Begin by greedily finding as many partial transversal as you can. Suppose you found $0 \leq i \leq n$ partial transversals of length $\frac{n}{3}$. We use theorem 39 to find a diagonal; let A be this diagonal. As there were no partial transversals left, there exists cells $a_1 \in A$ in row r , and $a_2 \in A$ in row r' that share the same symbol.

We identify the set U consisting of all cells in row r that do not share a column or symbol with any cell in A – note that U is non-empty as there are n choices in total, at most $\frac{n}{3} - 1$ columns could be unsuitable as could at most $\frac{n}{3} - 1$ symbols, leaving at least $\frac{n}{3} + 2$ suitable cells. As $n < 15$, we have $\frac{n}{3} + 2 < 7$, and therefore by lemma 41, there must exist some cell u in U whose transversal neither passes through the same column, nor contains the same symbol as a_1 . Therefore a_1 may poach the transversal association from u .

Repeat for each row as needed until A is a transversal. □

Corollary 42. *Any $3n \times 3n$ Latin square may be decomposed into $9n$ partial transversals of length n .*

We conclude this topic with the following conjecture:

Conjecture 43. *Using the poach move repeatedly, it is possible to find a partial transversal decomposition of any $\frac{n}{3} \times n$ Latin rectangle.*

The stumbling block with proving the conjecture is that for $n \geq 15$, it is possible that in row r , there is no sequence of poach moves that can uncover a suitable cell. However, given that you do not already have a transversal, and we do have a diagonal, there must be another uncovered cell in row r' that shares the same symbol - perhaps this row will have more success. If not, we can move to another cell (still maintaining a diagonal) – the cell that we have moved to must contain a symbol that is already in our diagonal, can this cell uncover a suitable symbol? Given the wealth of available options, it seems overwhelmingly likely that this conjecture is true.

Chapter 8

Conclusion

The primary goal of this thesis was to solve the conjecture posed by Cameron in [6]. We have seen numerous generalisations of Jacobson and Matthews' method and, perhaps, taken it to the limit.

One big topic that we have not covered is that of mixing time. The *mixing time* of a Markov chain is the number of iterations that must be completed before the distribution may be satisfactorily close to the stationary distribution. The current mixing time of the the Jacobson and Matthews Markov chain is currently unknown, despite many (unpublished) attempts to rectify the problem. There are two main methods of attacking a problem like this. The first is to find a *canonical path* between two vertices of the underlying graph and showing that no bottleneck occurs. The second method, known as *coupling* involves taking a walk on the graph from two arbitrarily chosen vertices. The walk is conducted using an algorithm to dictate the next move so that once the two paths collide, they stay together forever. The time it takes for the two paths to collide suggests information about the mixing time. The difficulty in this approach is finding an algorithm that pressurises the two systems to coincide.

We summarise what we have discovered in the following table:

Design	Jacobson and Matthews' technique works?
$LS((r, c, s), (\lambda_{RC}, \lambda_{RS}, \lambda_{CS}))$	Yes, for all admissible parameters
$LF((n, c), (\lambda_{NN}, \lambda_{NC}))$	Yes, if $\lambda_{NN} \geq \lambda_{NC}, \lambda_{NC}$ even. Also, true when $\lambda_{NC} = \lambda_{NN} = 1$ with $n < 12$. Otherwise, unknown.
$TS(v, \lambda)$	Yes, for $\lambda = 1$ with $v \leq 19$. Otherwise, unknown.

From the table we may infer that triple systems are “harder” to reconcile than factorisations, which in turn are “harder” than the resolution of squares. As we moved up in difficulty, we lost a type of point, and having the ability to “protect” points as we proved connectedness was of vital importance in the core of Jacobson and Matthews’ technique. It is my belief that to resolve the final cases, we need to discover a technique that takes advantage of having only 1 type of point. As we saw in the previous chapter, cycle trades are a promising concept, but alone they are not up to the task. Perhaps the right answer involves a mixture of cycle trades with a simple, predictable, sequence of ± 1 -moves.

There is another option available to us, which is to investigate what happens if we allow more than 1 negative triple to occur. In the previous chapter we saw that trade 68 in Forbes’ list could not be mimicked by ± 1 -moves. This is not the case if we allow two negative triples [15]. Allowing more negative triples allows us to simplify the connectedness proof because we do not need to return to a proper system before interchanging the edges of a cycle. For factorisations, this also allows us to ignore the condition that $\lambda_{NC} \geq \lambda_{NN}$ because that was only important in returning to properness. For the triple systems, we still have the issue of bridged graphs to overcome, but this is not an issue if λ is even. The question remains: is this Markov chain ergodic? At what price do these results come? The exact answer is unknown, but it certainly does increase the size of the

graph, meaning the amount of time we should traverse the graph will increase, perhaps damaging the mixing time.

Appendix A

DesignMC User Guide

DesignMC is a GAP [20] package for generating uniformly distributed random generalised 2-designs with block size 3.

A.1 Background

From the early stages of this research, it became apparent that it would benefit from having software to perform experiments. For example, we were able to prove connectivity of the Markov chain (for some small values) by simulating a random walk and logging each of the non-isomorphic systems that we discovered.

This software has been open-sourced so users may continue this research, or simply use it to generate generalised 2-designs of block size 3. This appendix contains the user guide for the software.

A.1.1 Licence

GNU General Public License v3

A.1.2 Requirements

GAP v4.5.5 [20]

Required GAP Packages:

- DESIGN Package [39]
- JSONGAP Package [12]

A.1.3 Installation

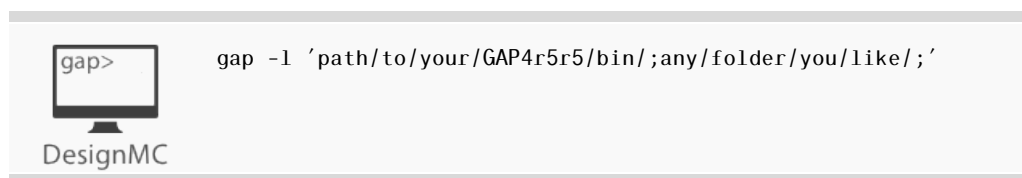
To initialise the DesignMC Package, put the root folder in the pkg directory of your GAP root and in GAP type:



```
gap> LoadPackage("DesignMC");
```

The screenshot shows a terminal window with a computer monitor icon and the text "DesignMC" below it. The command `gap> LoadPackage("DesignMC");` is entered in the terminal.

Alternatively, you can download the source to any/folder/you/like/DesignMC and then run GAP with



```
gap -l 'path/to/your/GAP4r5r5/bin/;any/folder/you/like/;'
```

The screenshot shows a terminal window with a computer monitor icon and the text "DesignMC" below it. The command `gap -l 'path/to/your/GAP4r5r5/bin/;any/folder/you/like/;'` is entered in the terminal.

A.1.4 Function Reference

Generating Generalised 2-designs

These functions are wrappers for Soicher's DESIGN Package. They handle the boiler plate code that is required to generate the particular designs in which we are interested.

QuickLatinSquare

Required Parameters:


- `n` *Positive integer*

Returns: *Record*

Description: Returns the cyclic Latin square of order `n`.

Usage:

```
gap> square:=QuickLatinSquare(4);
```



DesignMC

ProduceSquare

Required Parameters:

- `input` *Record*
- `input.v` *List A tuple of positive integers*

Optional Parameters:

- `input.lambdas` *List A tuple of positive integers for the lambda values RC, RS and CS (in that order).*
- `isoLevel` *0, 1, 2: See DESIGN Documentation.*
- `requiredAutSubgroup` *Group: See DESIGN Documentation.*
- `isoGroup` *Group See DESIGN Documentation.*
- `show_output` *Boolean Set to `true` for verbose mode.*
- `improper` *Boolean Set to `true` if you only want improper designs.*

Returns: *List*

Description: Returns a square with the specified parameters. This function wraps the DESIGN Package.

Usage:

```
gap> input:=rec(v:=[4,4,4], lambdas:=[2,2,2], isoLevel:=0,
improper:=true);
gap> ProduceSquare(input);
```

ProduceLamdaFactorisation

Required Parameters:

- `input` *Record*
- `input.v` *List* A tuple of positive integers

Optional Parameters:


- `input.lambdas` *List* A tuple of positive integers for the lambda values NC, NN (in that order).
- `isoLevel` *0, 1, 2:* See DESIGN Documentation.
- `requiredAutSubgroup` *Group:* See DESIGN Documentation.
- `isoGroup` *Group* See DESIGN Documentation.
- `show_output` *Boolean* Set to `true` for verbose mode.
- `improper` *Boolean* Set to `true` if you only want improper designs.

Returns: *List*

Description: Returns a lambda factorisation with the specified parameters. This function wraps the DESIGN Package.

Usage:

```
gap> ProduceLamdaFactorisation(rec(v:=[6,5]));
```



ProduceTripleSystem

Required Parameters:

- `input` *Record*
- `input.v` *List* A tuple of positive integers

Optional Parameters:


- `input.lambdas` *List* A tuple of positive integers for the lambda values RC, RS and CS (in that order).
- `isoLevel` *0, 1, 2*: See DESIGN Documentation.
- `requiredAutSubgroup` *Group*: See DESIGN Documentation.
- `isoGroup` *Group* See DESIGN Documentation.
- `show_output` *Boolean* Set to `true` for verbose mode.
- `improper` *Boolean* Set to `true` if you only want improper designs.

Returns: *List*

Description: Returns a triple system with the specified parameters. This function wraps the DESIGN Package.

Usage:

```
gap> ProduceTripleSystem(rec(v:=[7]));
```



Make2Design

Required Parameters:

- `input` *Record*
- `input.v` *List* A tuple of positive integers
- `input.k` *List* A tuple of positive integers

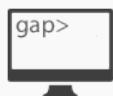
Optional Parameters:

- `input.lambdas` *List* A tuple of positive integers for the lambda values RC, RS and CS (in that order).
- `isoLevel` *0, 1, 2*: See DESIGN Documentation.
- `requiredAutSubgroup` *Group*: See DESIGN Documentation.
- `isoGroup` *Group* See DESIGN Documentation.
- `show_output` *Boolean* Set to `true` for verbose mode.
- `improper` *Boolean* Set to `true` if you only want improper designs.

Returns: *List*

Description: Returns a 2-design with the specified parameters. This function wraps each of `ProduceSquare`, `ProduceFactorisation` and `ProduceTripleSystem`. It uses `input.k` to determine which function to call.

Usage:



```
gap> Make2Design(rec(v:=[3,3,3], k:=[1,1,1]));
```

DesignMC

MakeSquare

Required Parameters:

- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer


Returns: *Record*

Description: Returns a square of order `n` with for lambda = `lambdaInt` .

This function wraps `ProduceSquare` .

Usage:

```
gap> MakeSquare(3,2);
```



MakeImproperSquare

Required Parameters:

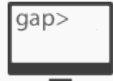
- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer

Returns: *Record*

Description: Returns an improper square of order `n` with for lambda =

`lambdaInt` . This function wraps `ProduceSquare` .

Usage:



```
gap> MakeImproperSquare(7, 1);
```

DesignMC

MakeLambdaFactorisation

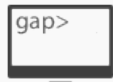
Required Parameters:

- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer

Returns: *Record*

Description: Returns a lambda factorisation on `n` vertices and `n - 1` colours, with for lambda = `lambdaInt`. This function simply wraps `ProduceFactorisation`.

Usage:



```
gap> MakeLambdaFactorisation(6, 1);
```

DesignMC

MakeImproperLambdaFactorisation

Required Parameters:


- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer

Returns: *Record*

Description: Returns an improper lambda factorisation on n vertices and $n - 1$ colours, with for lambda = `lambdaInt` . This function simply wraps `ProduceFactorisation` .

Usage:

```
gap> MakeImproperLambdaFactorisation(6,1);
```



DesignMC

MakeTripleSystem

Required Parameters:


- n *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer

Returns: *Record*

Description: Returns a triple system on n points, with for lambda = `lambdaInt` . This function wraps `ProduceTripleSystem` .

Usage:

```
gap> MakeTripleSystem(7,1);
```



DesignMC

MakeImproperTripleSystem

Required Parameters:


- n *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer

Returns: *Record*

Description: Returns an improper triple system on `n` points, with for lambda = `lambdaInt` . This function wraps `ProduceTripleSystem` .

Usage:

```
gap> MakeTripleSystem(9,1);
```



Enumerating Generalised 2-designs

EnumerateSquares

Required Parameters:


- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation

Returns: *List*

Description: Returns a list of squares on `n` points, with for lambda = `lambdaInt` . Setting the `isoLevel` to 2 will find the exact number of designs (up to isomorphism) matching your parameters. This function simply wraps `Make2Design` .

Usage:

```
gap> EnumerateSquares(5,1,2);
```



EnumerateImproperSquares

Required Parameters:


- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation

Returns: *List*

Description: Returns a list of improper squares on `n` points, with for lambda = `lambdaInt`. Setting the `isoLevel` to 2 will find the exact number of designs (up to isomorphism) matching your parameters. This function wraps `Make2Design`.

Usage:

```
gap> EnumerateImproperSquares(5,1,2);
```



EnumerateLambdaFactorisations

Required Parameters:

- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation


Returns: *List*

Description: Returns a list of lambda factorisations on `n` vertices and `n - 1` colours, with for lambda = `lambdaInt`. Setting the `isoLevel` to

2 will find the exact number of designs (up to isomorphism) matching your parameters. This function wraps `Make2Design`.

Usage:

```
gap> EnumerateLambdaFactorisations(6,2,2);
```



DesignMC

EnumerateImproperLambdaFactorisations

Required Parameters:


- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation

Returns: *List*

Description: Returns a list of improper lambda factorisations on `n` vertices and `n - 1` colours, with for lambda = `lambdaInt`. Setting the `isoLevel` to 2 will find the exact number of designs (up to isomorphism) matching your parameters. This function wraps `Make2Design`.

Usage:

```
gap> EnumerateImproperLambdaFactorisations(6,2,2);
```



DesignMC

EnumerateTripleSystems

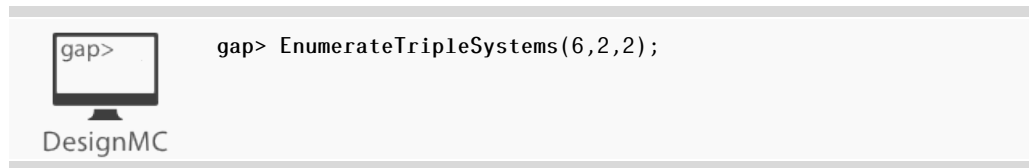
Required Parameters:

- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation

Returns: *List*

Description: Returns a list of triple systems on `n` points, with for lambda = `lambdaInt` . Setting the `isoLevel` to 2 will find the exact number of designs (up to isomorphism) matching your parameters. This function wraps `Make2Design` .

Usage:



```
gap> EnumerateTripleSystems(6,2,2);
DesignMC
```

EnumerateImproperTripleSystems

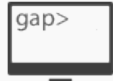
Required Parameters:

- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation

Returns: *List*

Description: Returns a list of improper triple systems on `n` points, with for lambda = `lambdaInt` . Setting the `isoLevel` to 2 will find the exact number of designs (up to isomorphism) matching your parameters. This function wraps `Make2Design` .

Usage:



```
gap> EnumerateTripleSystems(9,1,2);
```

DesignMC

Moving Around the Markov Chain

GeneratePivot

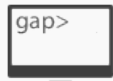
Required Parameters:

- `n` *Integer* A positive integer
- `lambdaInt` *Integer* A positive integer
- `isoLevel` *0, 1, 2* See DESIGN Documentation

Returns: *List*

Description: Returns a list of improper triple systems on `n` points, with for $\lambda =$ `lambdaInt`. Setting the `isoLevel` to 2 will find the exact number of designs (up to isomorphism) matching your parameters. This function wraps `Make2Design`.

Usage:



```
gap> square:=MakeSquare(5,1);
```

```
gap> GeneratePivot(square);
```

DesignMC

RemovableBlocks

Required Parameters:

- `D` *BlockDesign* The block design that you are going to modify.

- `pivot` *List* A block that you would like to add into the design `D` .

Returns: *List*

Description: Returns a list of blocks of `D` . One of these blocks will be the improper block, if any exists, after adding `pivot` .

Usage:

```

gap> square:=MakeSquare(5,1);
gap> pivot:=GeneratePivot(square);
gap> RemovableBlocks(square, pivot);

```

Hopper

Required Parameters:

- `D` *BlockDesign* A block design.
- `add` *List* A block that you would like to add into the design `D` .
- `remove` *List* A block that you would like to remove from the design `D` (must be an element of `RemovableBlocks(D, add)`). Alternatively, pass the empty list and a suitable block will be removed at random.

Returns: *BlockDesign*

Description: Performs one iteration in the Markov chain starting from `D` . The block `add` will be added in, and the block `remove` will be removed. If the resulting block design is improper, then the improper block is `remove` .

Usage:

```

gap> square:=MakeSquare(5,1);
gap> pivot:=GeneratePivot(square);
gap> newSquare:=Hopper(square, pivot, []);

```

OneStep

Required Parameters:

- **D** *BlockDesign* A block design.

Returns: *BlockDesign*

Description: Performs a set of iterations in the Markov chain starting from **D** and returning the first proper design encountered, whilst moving around at random.

Usage:



DesignMC

```
gap> square:=MakeSquare(5,1);  
gap> newSquare:=OneStep(square);
```

ManyStepsProper

Required Parameters:

- **D** *BlockDesign* A block design.
- **i** *Integer* Number of proper designs to ignore.

Returns: *BlockDesign*

Description: Performs a set of iterations in the Markov chain starting from **D** and returning the i^{th} proper design encountered, whilst moving around at random.

Usage:



DesignMC

```
gap> square:=MakeSquare(5,1);  
gap> newSquare:=ManyStepsProper(square, 10);
```

ManyStepsImproper

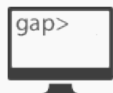
Required Parameters:

- **D** *BlockDesign* A block design.
- **i** *Integer* Number of designs to ignore.

Returns: *BlockDesign*

Description: Performs a set of iterations in the Markov chain starting from **D** and returning the i^{th} improper design encountered, whilst moving around at random.

Usage:



DesignMC

```
gap> square:=MakeSquare(5,1);  
gap> newSquare:=ManyStepsImproper(square, 10);
```

RandomWalkOnMarkovChain

Required Parameters:


- **D** *BlockDesign* A block design.
- **improper** *Boolean* Should look for improper designs.

Description: Performs a set of iterations in the Markov chain starting from **D**, making a log of the number of designs (both isomorphic, and non-isomorphic)

that it finds. If `improper = true` then it only looks for improper designs. This function is useful when deciding if the Markov chain is connected.

Usage:

```
gap> square:=MakeSquare(5,1);
gap> newSquare:=RandomWalkOnMarkovChain(square, false);
```



Pair Graphs

CreatePairGraph

Required Parameters:

- `D1` *BlockDesign* A block design.
- `p1_red` *Integer* A point to assign to the red edges.
- `D2` *BlockDesign* A block design.
- `p2_blue` *Integer* A point to assign to the blue edges.

Description: Creates a pair graph using all the blocks in `D1` containing `p1_red`, and all the blocks in `D2` containing `p1_blue`.

The graph is created using Mathematica, and saved to “ /Desktop/file”. The graph is constructed by starting choosing any block in `D1` that contains `p1_red`, for example, `[1, 2, p1_red]`; form an edge `[1,2]` coloured `p1_red`. Next, look in `D2` for a block that contains both `1` and `p2_blue`, for example, `[1, a, p2_blue]`; now form an edge `[1,a]` coloured `p2_blue`. Continue in this way until you return to your starting point and start the process again for the next component of the graph. Once you have no more components to make, the graph is complete.

Usage:

```
gap> square:=MakeSquare(5,2);;
gap> CreatePairGraph(square, 1, square, 2);;
```

DesignMC

FindAlternatingTrail

Required Parameters:

- `D` *BlockDesign* An improper block design.
- `starting_vertex` *Integer* The vertex that the path should start from.
- `finishing_vertex` *Integer* The vertex that the path should finish on.
- `isPathEvenLength` *0,1* Indicate whether the path should have even length.
- `edgeColour1` *List* A label to assign to the red edges.
- `edgeColour2` *List* A label to assign to the blue edges.
- `include_edge_lists` *List* Blocks that must be included in the path.
- `forbidden_edge_list` *List* Blocks that must not be included in the path.

Description: Finds an alternating path with the specified parameters.

Returns: *List* (vertices describing the path).

Usage:

```
gap> improperSquare:=MakeImproperSquare(5,2);;
gap> FindAlternatingTrail(improperSquare, 1, 5,0,12,15,[],[]);;
```

DesignMC

FindAlternatingTrailWithoutGivenBlueEdge


Required Parameters:

- `D` *BlockDesign* An improper lambda factorisation.

Description: Wraps `FindAlternatingTrail`. Finds an alternating trail in the pair graph of an improper lambda factorisation that does not include a blue edge adjacent to one of the special vertices of the pair graph.

Usage:

```
gap> improperFact:=MakeImproperLambdaFactorisation(6,1);
gap> FindAlternatingTrailWithoutGivenBlueEdge(improperFact);
```



IsChordedDG

Required Parameters:


- `D` *BlockDesign* A block design.
- `pointsList` *List* The two points that will be used to make the pair graph.

Description: Use this to detect if a pair graph is chorded or bridged.

Returns: *Boolean*

Usage:

```
gap> IsChordedDG(improperFactorisation, [9,10]);
```



Appendix B

Latin Squares App

The final piece of software that I created on this topic is a universal iOS application called “Latin Squares”. This software is aimed at a mathematically-interested, but not university-educated audience. It is freely available on the App Store and has been downloaded thousands of times since it appeared in May 2011.

In this section we shall tour the features of the application and show screenshots of each section.

If you would like to see the app yourself, and have an iOS device running iOS 3.0 or above, you may download it by searching the iOS App Store for “Latin Squares” or see [13].

The application is open-source on Github, and may be viewed at: <http://www.github.com/andydrizen/LatinSquares>

B.1 Detailed Tour

Landing Screen

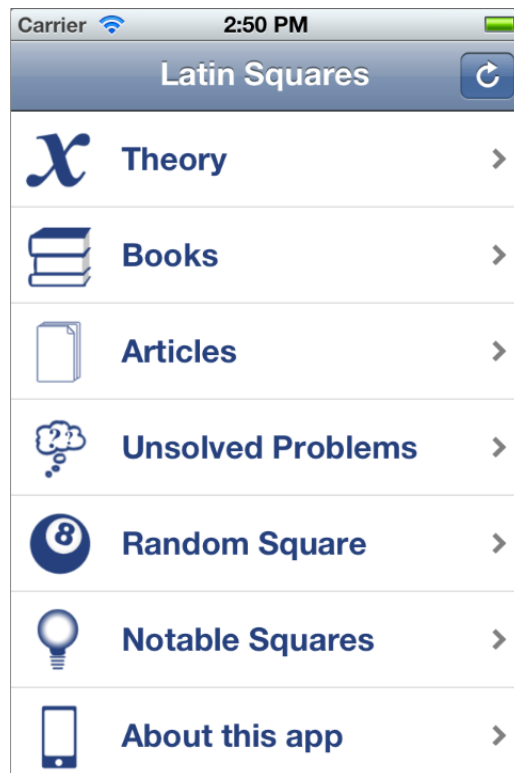
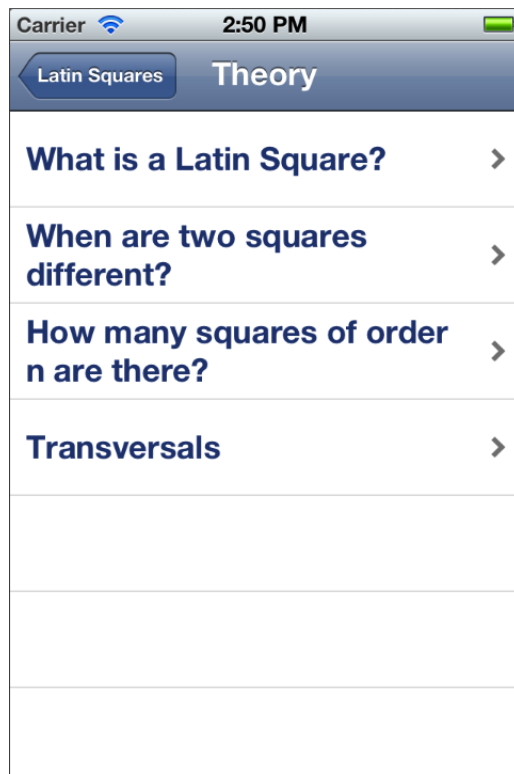


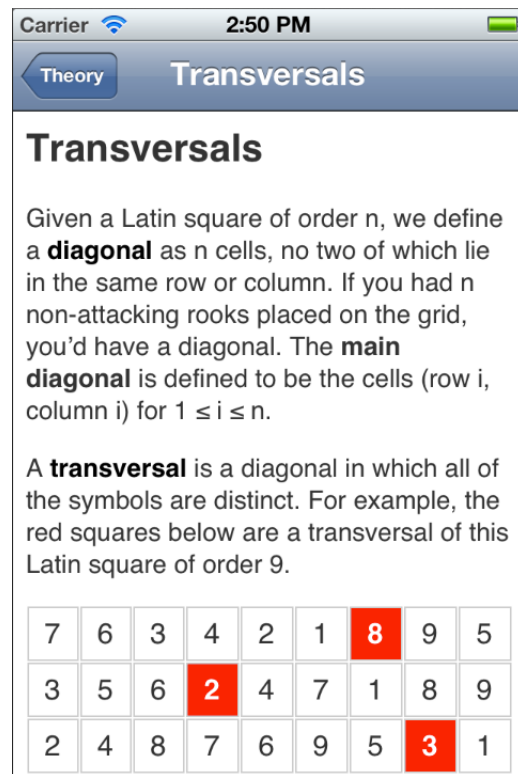
Figure B.1: The landing screen of the Latin Squares App.

Theory



Theory

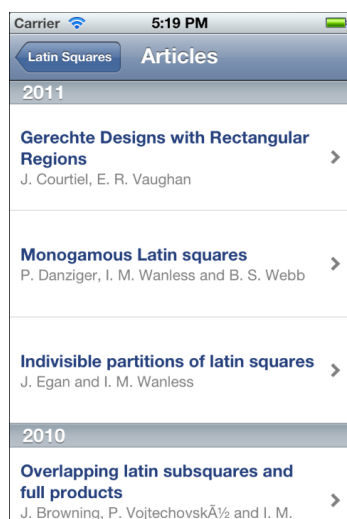
This section contains the basic theory of Latin squares. Currently there are only four articles here that cover the very basics of the field. More articles can be added and automatically added to this list. Each article is marked up using HTML and some pre-defined CSS.



Transversals

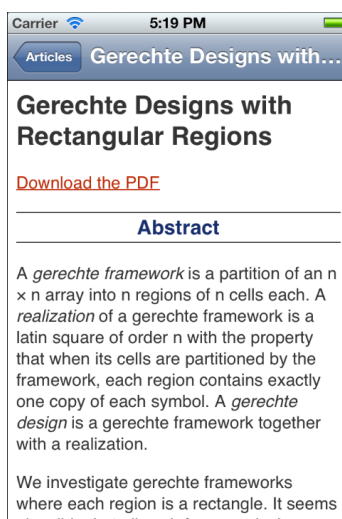
This article on transversals demonstrates the type of articles that the app can display. Each article is marked up using HTML and some predefined CSS. As these are just webpages, the articles may also include any interactive items that iOS will display (for example, Javascript (not Flash or Java)).

Articles



Articles List

The app contains an offline list of recent articles regarding Latin squares. This list is updated manually, and may be updated remotely. Every time the app is launched or manually updated, this list is updated.



Article Detail

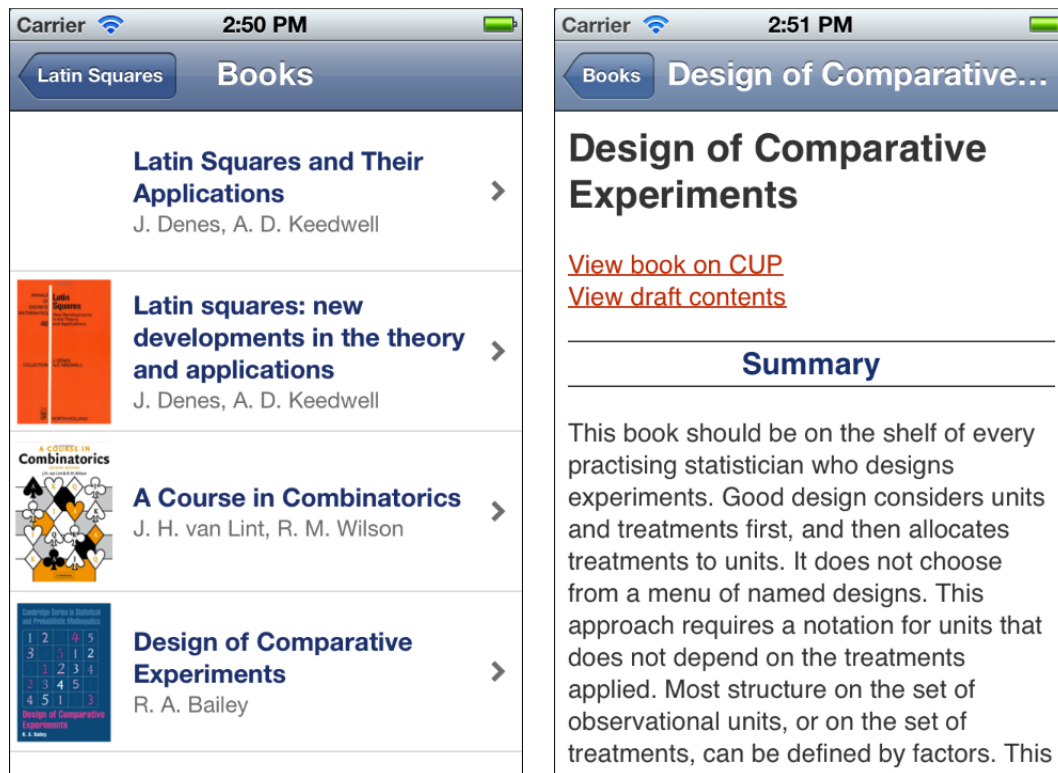
For each article, we display the title, abstract, and a link to the DOI or PDF (if available).



Article PDF

Tapping on the “download pdf” link takes you straight to the article (if available) or alternatively, to the journal page for the article.

Books



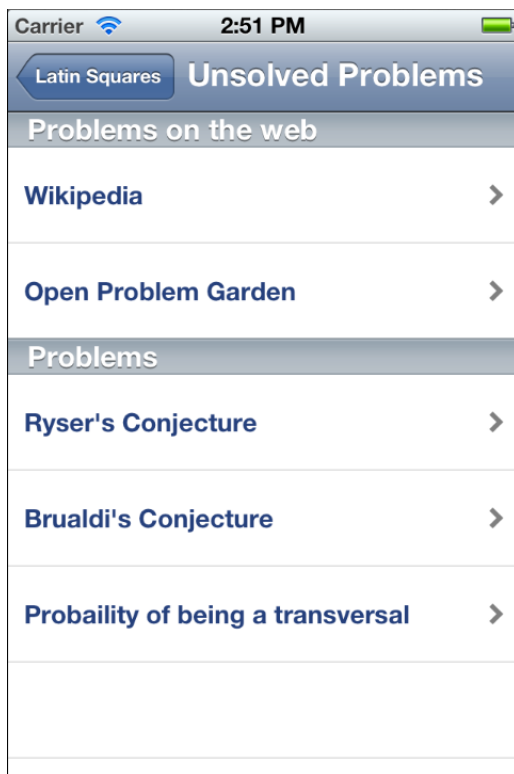
Books List

The app contains an offline list of the best books regarding Latin squares. This list is updated manually, and may be updated remotely. Every time the app is launched or manually updated, this list is updated.

Book Detail

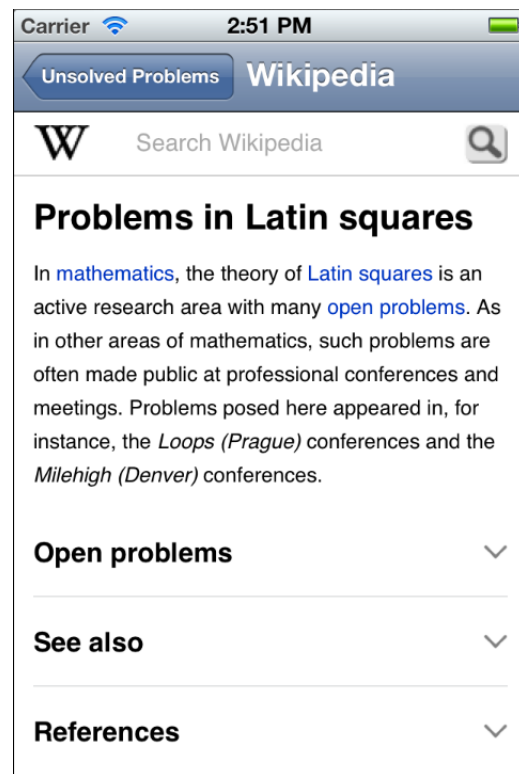
For each book, we display the title, summary, and a link to the publisher's page or PDF (if available).

Open Problems



Problems List

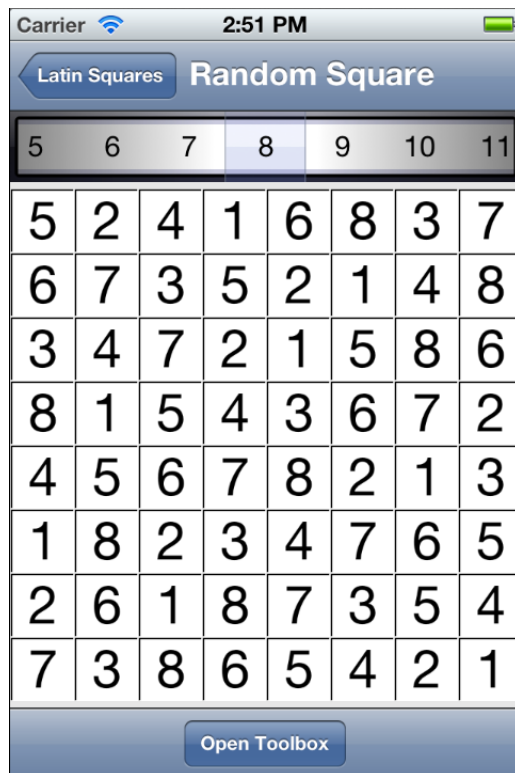
We keep a list of the problems that I found most interesting. Also, there are links to the open problem garden and Wikipedia.



Problems on Wikipedia

Tapping on the Wikipedia item shows the list of open problems on on Wikipedia, without leaving the app.

Random Squares



Carrier 2:51 PM

Latin Squares Random Square

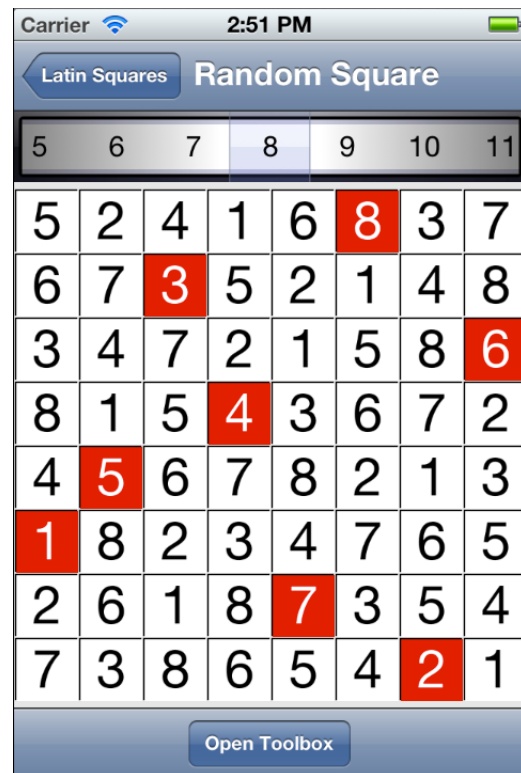
5 6 7 8 9 10 11

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

Open Toolbox

Random Squares

The user is able to generate a uniformly distributed random Latin square using Jacobson and Matthews' Markov chain. The square is presented in the more aesthetically pleasing grid format.



Carrier 2:51 PM

Latin Squares Random Square

5 6 7 8 9 10 11

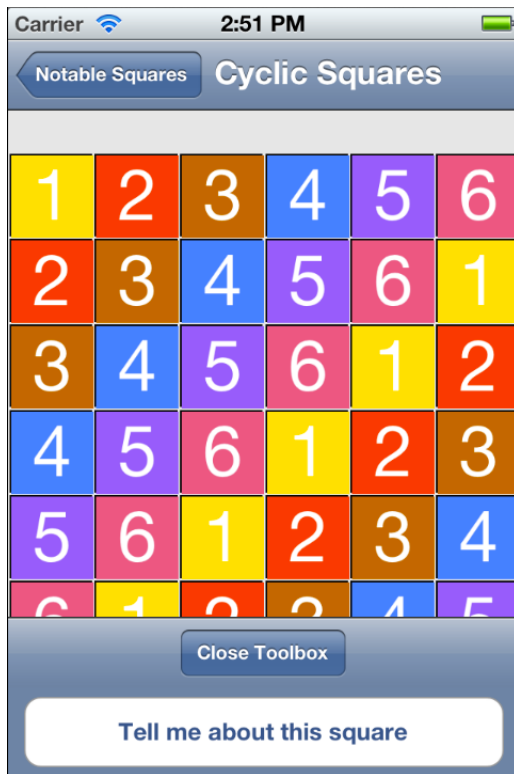
5	2	4	1	6	8	3	7
6	7	3	5	2	1	4	8
3	4	7	2	1	5	8	6
8	1	5	4	3	6	7	2
4	5	6	7	8	2	1	3
1	8	2	3	4	7	6	5
2	6	1	8	7	3	5	4
7	3	8	6	5	4	2	1

Open Toolbox

Toolbox

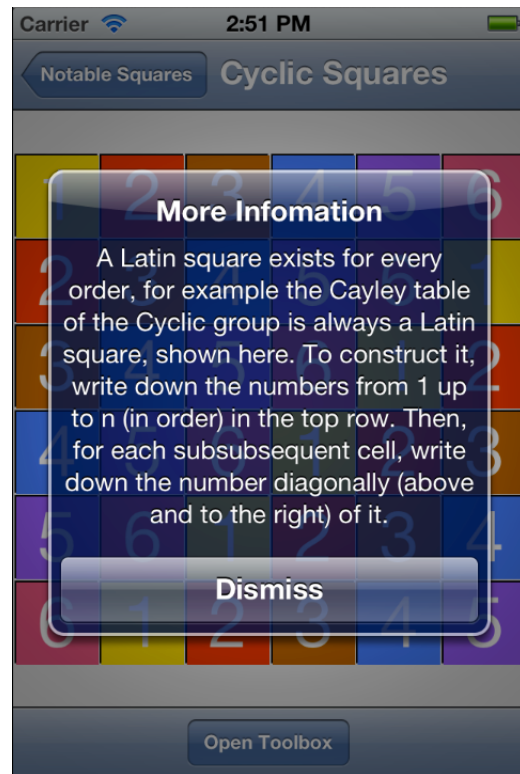
The toolbox allows the user to find transversals, generate new squares and set a colour for cell highlighting. Further, squares can be saved, emailed (as JSON) and loaded later (either into the app or into the DesignMC package).

Notable Squares



Notable Squares

Notable squares can be listed in this section. Any squares that have been exported from the DesignMC package (in JSON format) are easily added to this section.



Extra Info

A brief description of the square may also be given. In this case, we describe the Cyclic Latin square.

Bibliography

- [1] R. A. Bailey. Semi-latin squares. <http://www.maths.qmul.ac.uk/~rab/sls.html>. [Online; accessed 01-July-2012].
- [2] R. A. Bailey and P. J. Cameron. Latin squares: Equivalentents and equivalence. *The Encyclopaedia of Design Theory*, 2003. [Online; accessed 01-July-2012].
- [3] K. Balasubramanian. On transversals in latin squares. *Linear Algebra Appl.*, 131:125 – 129, 1990.
- [4] S. Bammel and J. Rothstein. The number of 9×9 latin squares, discrete math.11. *J. Comb*, (93), 1975.
- [5] R. C. Bose, S. S. Shrikhande, and E. T. Parker. Further results on the construction of mutually orthogonal latin squares and the falsity of Euler’s conjecture. *Canad. J. Math.*, 12:189 – 203, 1960.
- [6] P. J. Cameron. A generalisation of t-designs. *Discrete Math.*, 309(14):4835 – 4842, 2009.
- [7] N. J. Cavenagh, C. Greenhill, and I. M. Wanless. The cycle structure of two rows in a random latin square. *Random Structures and Algorithms*, 33(3):286–309, 2008.
- [8] N. J. Cavenagh and I. M. Wanless. On the number of transversals in cayley tables of cyclic groups. *Discrete Appl. Math.*, 158(2):136 – 146, 2010.

- [9] C. J. Colbourn and J. H. Dinitz, editors. *Handbook of Combinatorial Designs*. CRC Press, 2006.
- [10] J. Dinitz, D. Garnick, and B. McKay. There are 526,915,620 nonisomorphic one-factorizations of K_{12} . *Journal of Combinatorial Designs*, 2(4):273–285, 1994.
- [11] A. L. Drizen. The DesignMC package for GAP, version 1.0. <http://www.maths.qmul.ac.uk/~ald/DesignMC>, 2011. [Online; accessed 01-July-2012].
- [12] A. L. Drizen. The JSONGAP package for GAP, version 1.0. <https://github.com/andydrizen/JSONGAP>, 2011. [Online; accessed 01-July-2012].
- [13] A. L. Drizen. Latin Squares - A Universal iOS App. <http://itunes.apple.com/gb/app/latin-squares/id441478641?mt=8>, 2011. [Online; accessed 01-July-2012].
- [14] A. L. Drizen. Generating uniformly distributed random 2-designs with block size 3. *Journal of Combinatorial Designs*, 20(8):368–380, 2012.
- [15] A. L. Drizen, M. J. Grannell, and T. S. Griggs. Pasch trades with a negative block. *Discrete Math.*, 311(21):2411 – 2416, 2011.
- [16] L. Euler. Recherches sur une nouvelle espece des quarres magiques. *Verh.Genootsch. der Wet.Vlissingen*, (9):1782.
- [17] A.D. Forbes. *Configurations and colouring problems in block designs*. PhD thesis, The Open University, 2006.
- [18] A.D. Forbes, M.J. Grannell, and T.S. Griggs. On 6-sparse steiner triple systems. *J. Combin. Theory Ser. A*, 114:235–252, 2007.
- [19] M. Frolov. Sur les permutations carrés. *J. de Math. spéc.*, IV, pages 25–30, 1890.

- [20] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.12*, 2008. [Online; accessed 01-July-2012].
- [21] P.B. Gibbons. Computing techniques for the construction and analysis of block designs. *Department of Computer Science, University of Toronto, Technical Report, 92/76*, 1976.
- [22] M. J. Grannell, T. S. Griggs, and C. A. Whitehead. The resolution of the anti-pasch conjecture. *J. Combin. Des.*, 8(4):300 – 309, 2000.
- [23] M.J. Grannell, T.S. Griggs, and J.P. Murphy. Some new perfect steiner triple systems. *J. Combin. Des.*, 7:327–330, 1999.
- [24] M.J. Grannell, T.S. Griggs, and J.P. Murphy. Switching cycles in steiner triple systems. *Util. Math*, 56:3–21, 1999.
- [25] R. Häggkvist and A. Johansson. Orthogonal latin rectangles. *Combinatorics, Probability and Computing*, 17:519 – 536, 2008.
- [26] F. Harary. *Graph Theory*, chapter 9. Addison-Wesley, 1969.
- [27] A. J. W. Hilton. Problem bcc 13.20. [Online; accessed 01-July-2012], 1991.
- [28] A. D. Keedwell J. Dénes. *Latin Squares and their Applications*. Academic Press, New York-London, 1974.
- [29] M. T. Jacobson and P. Matthews. Generating uniformly distributed random latin squares. *J. Combin. Des.*, 4:405 – 437, 1996.
- [30] P. Kaski, V. Mäkinen, and P. R. J. Östergård. The cycle switching graph of the steiner triple systems of order 19 is connected. *Graphs Combin.*, 27:539 – 546, July 2011.
- [31] P. Kaski and P. Östergård. There are 1,132,835,421,602,062,347 nonisomorphic one-factorizations of K_{14} . *Journal of Combinatorial Designs*, 17(2):147–159, 2009.

- [32] W. J. Martin. Mixed block designs. *J. Combin. Des.*, 6:151–163, 1998.
- [33] B. McKay and E. Rogoyski. Latin squares of order 10. *Electron. J. Combin.*, 2, 1995.
- [34] B. D. McKay. nauty user’s guide (version 1.5), technical report tr-cs-90-02. <http://cs.anu.edu.au/people/bdm/nauty/>. [Online; accessed 01-July-2012].
- [35] B.D. McKay and I.M. Wanless. On the number of latin squares. *Ann. Comb.*, 9:335 – 344, 2005.
- [36] H. W. Norton. The 7×7 squares. *Annals of Human Genetics*, 9(3):269 – 307, 1939.
- [37] A. Sade. Enumération des carrés latins. application au 7 ème ordre. conjectures pour les ordres supérieurs, privately published, 1948.
- [38] A. Sade. An omission in norton’s list of 7×7 squares. *Ann. Math. Statist.*, 22(2):306 – 307, 1951.
- [39] L. H. Soicher. The DESIGN package for GAP, version 1.6. http://designtheory.org/software/gap_design, 2011. [Online; accessed 01-July-2012].
- [40] L. H. Soicher. The GRAPE package for GAP, version 4.6.1. <http://www.maths.qmul.ac.uk/~leonard/grape>, 2012. [Online; accessed 01-July-2012].
- [41] L.H. Soicher. On generalised t -designs and their parameters. *Discrete Math.*, 311:1136–1141, 2011.
- [42] J. Steiner. Combinatorische aufgabe. *Journal für die Reine und Angewandte Mathematik*, 45:181 – 182, 1853.

- [43] G. Tarry. Le problème de 36 officiers. *Compte Rendu de l'Association Française pour l'Avancement de Science Naturel (Secrétariat de l'Association)*, 2:170 – 203, 1901.
- [44] I. M. Wanless. Cycle switches in latin squares. *Graphs Combin.*, 20:545 – 570, 2004. 10.1007/s00373-004-0567-7.
- [45] I. M. Wanless. Diagonally cyclic latin squares. *Eur. J. Combin.*, 25(3):393 – 413, 2004.
- [46] I. M. Wanless. Discussion of hitlon's bcc13.20 problem. Personal communication, 2011.
- [47] M. Wells. The number of latin squares of order eight. *J. Combin. Theory*, (3):98–99, 1967.
- [48] R. D. Yates and D. J. Goodman. *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*, chapter 11.4. John Wiley and Sons inc., 1999.

Index

- $(n \times n)/k$ semi-Latin square, 50
- λ -complete graph, 63
- λ -factor, 63
- λ -factorisation, 63
- λ -triple system of order v , 77
- V-height, 11
- t -design, 9
- $t - (v, k, \lambda)$ design, 9
- 1-factor, 61
- 1-factorisation, 61
- anti-Pasch systems, 84
- aperiodic, 25
- binary, 79
- block, 64
- block design, 9
- block set, 64
- blocks, 9
- bridged, 67
- canonical path, 95
- chorded, 67
- colours, 63
- columns, 49
- complete graph, 61
- conjugate, 28
- Conjugate:, 16
- constant λ , 49, 63
- core, 53
- coupling, 95
- cycle, 20
- cycle trading, 83
- cyclic Latin square, 16
- derangement, 18
- designs, 13
- diagonal, 87
- discrepancy cycle, 44
- ergodic, 25
- Fano Plane, 10
- foundation, 84
- generalised λ_{NC} -factorisation of $\lambda_{NN}K_n$,
63
- generalised t -design, 12
- generalised Latin square, 49
- hill climbing, 18

improper, 27, 78
 improper block, 49
 improper square, 49
 intercalate, 22
 irreducible, 24
 isomorphic, 17, 28, 78, 79
 isotopic, 28
 Isotopic:, 16

 Kirkman triple system, 78

 Latin rectangle, 18, 89
 Latin square of order n , 11, 15

 Main class isotopic:, 16
 Markov chain, 23
 Markov property, 23
 memoryless, 23
 mixed block designs, 14
 mixing time, 26, 95
 mutually orthogonal, 88

 near-core, 53
 negative block, 49, 79
 neighbour set, 24
 neighbourhood, 20
 nodes, 63
 non-isomorphic, 17
 orbit, 85

 pair graph, 32

 partial transversal, 87
 partial triple system, 85
 Pasch configuration, 84
 perfect, 83
 period, 25
 poach move, 90
 point set, 64
 points, 9

 quadrilateral, 84

 RBR, 54
 resolvable, 78
 reversible, 25
 row-pair graph, 20
 rows, 49

 semi-Eulerian, 55
 special, 33
 states, 23
 Steiner triple system of order v , 77
 Steiner triple systems on v points, 9
 suitable, 85
 symbols, 49
 system of distinct representatives, 18

 table notation, 29
 trade, 84
 transversal, 87
 transversal decomposition, 88
 unique stationary distribution, 26

volume, 84