The Search for a Triple of Mutually Orthogonal Latin Squares of Order Ten: Looking Through Pairs of Dimension Thirty-Five and Less

by

Erin Delisle
B.Sc., University of Victoria, 2005

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

©Erin Delisle, 2010
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.

The Search for a Triple of Mutually Orthogonal Latin Squares of Order Ten:
Looking Through Pairs of Dimension Thirty-Five and Less

by

Erin Delisle
B.Sc., University of Victoria, 2005

Supervisory Committee

Dr. Wendy Myrvold, Supervisor
(Department of Computer Science)

Dr. Frank Ruskey, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Wendy Myrvold, Supervisor
(Department of Computer Science)

Dr. Frank Ruskey, Departmental Member
(Department of Computer Science)

ABSTRACT

A computer generation of all pairs of mutually orthogonal Latin squares of order ten and dimension 35 or less is undertaken. All such pairs are successfully generated up to main class equivalence. No pairs of mutually orthogonal Latin squares of order ten exist for dimension 33. Six dimension 34 pairs, which are counterexamples to a conjecture by Moorehouse, are found. Eighty-five pairs can be formed with dimension 35. None of the pairs can be extended to a triple. If a triple of mutually orthogonal Latin squares exists for order ten, the pairs of Latin squares in the triple must be of dimension 36 or 37.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A long outstanding question in design theory is: does there exist a triple of mutually orthogonal Latin squares of order ten? Many attempts to answer this question have failed to provide a solution. Recent results [14] have provided new areas to investigate for such a triple. This work uses these results to search for triples of mutually orthogonal Latin squares of order ten.

1.1 Definitions

The definitions listed here are critical to the understanding of this work. Further definitions, specific to the methods used to solve the problem, can be found throughout Chapter 2.

A *Latin square of order* n is a $n \times n$ array of n symbols such that each symbol appears once in each row and column [11, 12]. The following is Latin square of order three:

$$\begin{array}{ccc} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{array}$$

The set $\{0, \ldots, n-1\}$ will be used as the symbols for a Latin square of order n. A *Latin subsquare of order k* is a selection of k rows and k columns of a Latin square for which only k different symbols appear. A Latin square with entries only partially entered is said to adhere to the *Latin square property* if no symbol appears more than once in any row or column. A Latin square is *reduced* if both the first row and column are sorted in increasing order. Any Latin square can be put in reduced form by permuting its rows and columns. The following is a reduced version of the first Latin square:

$$\begin{array}{ccc} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{array}$$

Two Latin squares of the same order are said to be *orthogonal* if when superimposed, each ordered pair of symbols is unique. The following orthogonal pair of Latin squares is composed from the previous two Latin squares.

$$\begin{array}{ccc} 00 & 21 & 12 \\ 11 & 02 & 20 \\ 22 & 10 & 01 \end{array}$$

A set of Latin squares is said to be *mutually orthogonal* if all Latin squares in the sets are pairwise orthogonal. Mutually orthogonal Latin squares are referred to as MOLS.

A *transversal* of a Latin square of order n is a set of n cells that cover each row, column, and symbol exactly once. The following is a transversal of the first Latin square:

$$\begin{array}{ccc} 0 & - & - \\ - & - & 2 \\ - & 1 & - \end{array}$$

A pair of transversals is *disjoint* if they share no common cell. A set of n disjoint transversals for a Latin square can be used to generate another Latin square orthogonal to the original by assigning a symbol to each transversal. Similarly, a set of cells with the same symbol in an orthogonal mate correspond to a transversal in the original. Due to this, a Latin square of order n has an orthogonal mate if and only if it has n disjoint transversals.

A set of MOLS is said to have a *transversal* if there are n cells that include each row, column, and a symbol from each Latin square exactly once each. In the same manner as a single Latin square, a set of r MOLS of order n can be extended to a set of $r+1$ MOLS if and only if the original set of MOLS has n disjoint transversals.

The *orthogonal array representation* of a Latin square of order n consists of n^2 rows and three columns representing rows, columns, and symbols of the original Latin square [15]. The row entries of the orthogonal array each specify a cell and the symbol at that cell. The following is the orthogonal array representation of the first Latin square:

Row	*Column*	*Symbol*
0	0	0
0	1	2
0	2	1
1	0	1
1	1	0
1	2	2
2	0	2
2	1	1
2	2	0

A pair of MOLS can be represented using an orthogonal array representation with four columns: row, column, symbol of the first square, and symbol of the second square. These columns will be labelled R, C, S, and T. A set of k MOLS can similarly be represented with $k + 2$ columns in the orthogonal array representation: two for rows and columns of the Latin squares, and k for the symbols found in each cell.

The *binary code representation* of a pair of MOLS of order n consists of n^2 rows and $4n$ columns of binary values. Each row of the code is equivalent to a row of the orthogonal array representation. Four 1's are present in each row of the code, indicating a row, a column, the symbol in the first Latin square, and the symbol in the second Latin square. The following is the code of the previous pair of MOLS:

	r_0	r_1	r_2	c_0	c_1	c_2	s_0	s_1	s_2	t_0	t_1	t_2
(0,0,0,0):	1	0	0	1	0	0	1	0	0	1	0	0
(0,1,2,1):	1	0	0	0	1	0	0	0	1	0	1	0
(0,2,1,2):	1	0	0	0	0	1	0	1	0	0	0	1
(1,0,1,1):	0	1	0	1	0	0	0	1	0	0	1	0
(1,1,0,2):	0	1	0	0	1	0	1	0	0	0	0	1
(1,2,2,0):	0	1	0	0	0	1	0	0	1	1	0	0
(2,0,2,2):	0	0	1	1	0	0	0	0	1	0	0	1
(2,1,1,0):	0	0	1	0	1	0	0	1	0	1	0	0
(2,2,0,1):	0	0	1	0	0	1	1	0	0	0	1	0

The *rank*, or *dimension*, of a pair of MOLS is the number of linearly independent columns in the binary code representation. To calculate the rank of any given pair of MOLS, convert to its binary code representation and use Gaussian elimination to zero out as many columns as possible.

A *relation* is a subset of the columns of the binary code representation of a Latin square such that these columns sum to zero modulo two. The following is a relation for a pair of MOLS of order four, along with a pair of MOLS which satisfies the relation:

R columns	C columns	S columns	T columns
01	01	01	01

00 22 13 31

33 11 20 02

12 30 01 23

21 03 32 10

Two Latin squares are said to be *isotopic* if there exists a permutation of the rows, permutation of the columns, and permutation of the symbols that will make one Latin square equal to the other [19]. Sets of Latin squares which are isotopic form equivalence classes called *isotopy classes*. A set of MOLS is *isotopic* to another if a permutation of rows, permutation of columns, and a permutation for each symbol set makes the two sets of Latin squares identical.

Two Latin squares are said to be *main class isotopic* if in addition to the isotopy permutations, a permutation of the roles of the rows, columns, and symbols will cause one Latin square to be equivalent to the other. Permuting the roles is equivalent to permuting the columns in the orthogonal array representation. For example, exchanging the roles of rows and columns is equivalent to taking the transpose of the Latin square. Sets of MOLS are *main class isotopic* if they are equivalent under composition of either the isotopy operations, or exchanging any of the columns of the corresponding orthogonal array representation.

Main class isotopy is also referred to as *paratopy*. The set of paratopy operations which map a Latin square to itself form a group called an *autoparatopy group*.

A *canonical form* is a standardized form for a combinatorial object such that all objects with the same canonical form are equivalent up to some operations. For Latin squares we can define canonical forms for both isotopy and main class operations.

A *graph* is a set of vertices and a set of edges in which each edge is an unordered pair of vertices. Vertices a, b are said to be *adjacent* if the pair (a, b) is an edge. A *clique* is a subset of vertices C such that all vertices in C are pairwise adjacent.

1.2 Background

Mutually orthogonal Latin squares have provided interesting open questions for a long time. A brief summary of background material relevant to this work is contained here.

Interest in orthogonal Latin squares began with the 36 officer problem by Euler [13]: Can 36 officers, of six ranks from six different regiments, be arranged in a 6×6 square such that a representative from each regiment and rank exists in each row or column? This is equivalent to finding a pair of orthogonal Latin squares of order six. Constructions were found for orthogonal pairs of odd order and of order divisible by four. The apparent inability to solve the 36 officer problem led to the conjecture by Euler that for order $n = 4t + 2, t \geq 0$, no pair of MOLS exists.

In 1900, Tarry proved no pair of MOLS exists for order six [28]. His proof was by exhaustive search, done by hand. It was not until Bose and Shrikrande later found a counterexample of order 22 in 1959 that the conjecture was disproved [3]. With Parker they managed to generate an set of counterexamples for all orders $n = 4t+2 \geq 10$ [4].

An early computer search for orthogonal mates to random Latin squares of order ten was performed by Parker by finding disjoint transversals [25]. The Latin squares checked had roughly between 950 and 1100 transversals each. Pairs of MOLS proved to be very common. He wanted to find a triple of MOLS, but results were negative for the limited set of squares checked. Computational power at the time of the search was severely limited by modern standards.

In another attempt to find a triple of MOLS, Parker investigated turn-squares, which possess many more transversals than random Latin squares [26, 27, 6]. A Latin

square is a *turn-square* if it can be written as

$$\begin{array}{cc} A+5B & A+5\bar{B} \\ A+5\bar{B} & A+5B \end{array}$$

where A is an order five Latin square, B is a 5×5 array of binary values and $\bar{B}$ is the negation of B. The largest known number of transversals for any Latin square of order ten, 5504, belongs to a turn-square [18]. The maximum number of pairwise disjoint transversals for pairs of MOLS of order ten to date is four, first found in a turn-square [7].

Some other works of note:

- A computer search for triples of MOLS of order ten led to the discovery of large numbers of "almost orthogonal" triples of Latin squares, though no completely orthogonal triple [16].
- A set of four MOLS of order ten with a 2×2 hole (blank entries) has been constructed [5].
- Computational constructions of sets of five MOLS of order twelve were successful [1, 2].

Information regarding sets of MOLS for orders other ten can be found in [9, 10]. This includes various constructions of sets of MOLS. For more information regarding the theory, history and applications Latin squares, see [11, 12].

1.2.1 Negative Conditions for Sets of MOLS of Order Ten

Few negative conditions for the existence of sets of MOLS of order ten are known. Known negative conditions are described here.

Theorems concerning MOLS by Parker, generalized over all orders, provide some negative results with respect to triples of order ten [23, 24]. Specifically, any order ten pair of MOLS with a mutually orthogonal Latin subsquare of order three or more can not be extended to a triple of MOLS. Work by Myrvold [21] shows that for 20 of 28 cases for a Latin square of order ten with a 4×4 subsquare, that square can not be part of a triple of MOLS.

Sets of nine MOLS have been shown not to exist for order ten [17]. Combined with an earlier result, this implies that no set of seven MOLS exists for order ten [8]. Sets of four MOLS of order ten must have some nontrivial relations in order to exist [14].

1.2.2 Recent Work

Recent work on sets of MOLS of order ten motivates this work. Some relevant recent results are described here.

A recent large scale computer search [19] failed to find a triple of MOLS of order ten, but did not rule out the existence of such a triple. All Latin squares of order ten having nontrivial autoparatopy groups were generated, then tested for extendability to triples of MOLS. This work concluded that modern computational power is insufficient to answer the existence of triples of MOLS of order ten by first generating all pairs.

Work by Howard [14] characterizes the structure of possible relations for pairs of MOLS of order ten. Using this, all sets of relations up to main class equivalence for order ten can be generated for a given dimension. For detail regarding these relations, see Section 2.2.

For order ten, pairs of MOLS with dimension less than 33 are shown not to exist [14]. In [15], the existence of orthogonal pairs of dimension 33 is stated as an open problem. The maximum dimension achievable for a pair of MOLS of order ten is 37, due to innate linearly independent relations of Latin squares [15].

A conjecture by Moorehouse [20] implied that no pairs of MOLS exist for order ten, dimension 34. This was recently disproven by counterexample [15]. This thesis proves that the number of such counterexamples is six up to main class equivalence.

1.3 Problem Statement

Recent results have provided both new questions and new strategies for old unanswered questions. The questions and tasks addressed by this work are detailed here.

The existence of a triple of MOLS of order ten has not been resolved despite numerous attempts. A very large number of pairs of MOLS exist [19]. Searching through all pairs of MOLS for a triple would be an impossible task.

Recent results regarding relations [14] for pairs of MOLS of order ten have provided

more restricted pairs of MOLS to search. This characterization of relations of order ten has allowed the generation of all pairs of MOLS of order ten for dimension 35 or less. Such a generation answers the following questions:

- Does there exist a pair of MOLS of order ten of dimension 35 or less which can be extended to a triple?
- Does there exist any pairs of MOLS of order ten of dimension 33?
- How many pairs of MOLS of order ten of dimensions 35 or less exist up to main class equivalence?

To answer these questions, an exhaustive computer generation of these restricted pairs of MOLS has been performed. All pairs of MOLS of order ten for dimensions less than or equal to 35 have been counted up to main class equivalence, then tested for possible extension to a triple of MOLS. No triples of MOLS of order ten have been found. Pairs of MOLS of order ten and dimension 33 are shown not to exist.

The method used for generating pairs of MOLS of order ten with dimension 35 or less is detailed in Chapter 2. This includes how data regarding the generated pairs was obtained after generation. The results from this process are found in Chapter 3. Concluding remarks and directions for future research are contained in Chapter 4.

Chapter 2

Method

This chapter details the methods used to generate and analyze all pairs of MOLS of order ten and dimension 35 or less. A description of the overall process can be found in Section 2.1. Theory regarding the relations that must be considered is contained in Section 2.2.

Sections 2.3 and 2.4 describe the generation procedure. In Section 2.3, pairs of MOLS are partially completed in a way so as to reduce combinatorial symmetry. Section 2.4 details the exhaustive method for completing the partially finished pairs of MOLS of Section 2.3.

A canonical form routine for pairs of MOLS with respect to main class equivalence is described in Section 2.5. The method used for determining if a generated pair of MOLS can be extended to a triple is described in Section 2.6. For details of the results found using these methods, see Chapter 3.

2.1 Overview

The process of generating and analyzing all pairs of MOLS of order ten and dimension 35 or less is performed in several distinct stages. The order of those stages and how they interact is described here.

To generate all pairs of MOLS of order ten for dimension 35 or less, a two stage process is used. Selected cells of the pairs of MOLS are first filled in all ways up to main class equivalence. Then an exhaustive backtracking procedure completes the partial pairs of MOLS in all ways possible without regard to symmetry.

When symbol pairs are first placed into cells of a pair of MOLS, there are many combinatorially symmetrical ways to do so with respect to main class operations. Minimizing the number of partially completed pairs of MOLS with respect to main class equivalence during the first stage reduces the number of pairs generated during the second stage, having a significant impact on the overall computation time.

With symmetries due to main class operations reduced, the exhaustive backtracking can complete the partial pairs of MOLS while ignoring main class symmetries. The second stage for completing the pairs of MOLS is designed to finish as fast as possible; isolating attention regarding main class operations to the first stage is critical in accomplishing this.

The partial pairs of MOLS produced by the first stage can be split over a number of different processors. This allows the second stage to be easily distributed, making maximal use of available computational resources.

Since the generation process does not necessarily produce a unique pair of MOLS for each main class, a canonical form routine is used to reduce the generated pairs down to a unique representative per main class. Converting each pair of the remaining pairs of MOLS to a binary code representation and then using Gaussian elimination provides the dimension for each pair.

To determine if a generated pair of MOLS can be extended to a triple, all transversals for the pair are generated. If a group of ten mutually disjoint transversals can be found, then the pair can be extended to a triple. If the pair cannot be extended, the data regarding the number of transversals and the maximum number of disjoint transversals is recorded.

2.2 The Relations

To generate all pairs of MOLS of order ten and dimension 35 or less, all possible relations for such pairs must be known. All such relations up to main class equivalence are characterized here.

In Proposition 5.4 of [14], Howard proves that for a pair of MOLS of order ten, any nontrivial relation must be representable in a form with four columns chosen for each of the R, C, S, and T sets of columns. Note that for each of R, C, S, and T, a choice of six columns is equivalent to a choice of the other four columns in the relation. If there are two relations, they must intersect for some number of columns for each of R, C, S, and T. Any two linearly independent relations will produce a third linearly dependent relation, which must also respect Proposition 5.4 of [14]. Only relations which intersect in one or two columns for each of R, C, S, and T produce allowable

linearly dependent relations. This gives the following five sets of relations up to main class equivalence, though further analysis indicates that cases 2 and 4 are not possible:

	R columns	C columns	S columns	T columns
$R_{1,1}$	0123	0123	0123	0123
$R_{1,2}$	0456	0456	0456	0456
$R_{2,1}$	0123	0123	0123	0123
$R_{2,2}$	0456	0456	0456	0145
$R_{3,1}$	0123	0123	0123	0123
$R_{3,2}$	0456	0456	0145	0145
$R_{4,1}$	0123	0123	0123	0123
$R_{4,2}$	0456	0145	0145	0145
$R_{5,1}$	0123	0123	0123	0123
$R_{5,2}$	0145	0145	0145	0145

For a cell, (r, c), and two relations, R_1 and R_2, consider the following definitions:

- $BR(r, i) = 1$ if the column of the binary code corresponding to row r of the pair of MOLS is included in the relation R_i and 0 otherwise for $i = 0, 1$.
- $BC(c, i) = 1$ if the column of the binary code corresponding to column c of the pair of MOLS is included in relation R_i and 0 otherwise for $i = 0, 1$.
- $RC\_CLASS(r, c) = 2*[(BR(r, 1)+BC(c, 1))$ modulo $2]+[(BR(r, 2)+BC(c, 2))$ modulo $2]$

RC_CLASS forms an equivalence relation with four equivalence classes for cells under the relations R_1 and R_2. Cells (r, c) and (r', c') are in the same equivalence class if $RC\_CLASS(r, c) = RC\_CLASS(r', c')$.

For a symbol pair, (s,t), and two relations, R_1 and R_2, consider the following definitions:

- $BS(s,i) = 1$ if the column of the binary code corresponding to first symbol s of the pair of MOLS is included in the relation R_i and 0 otherwise for $i = 0, 1$.
- $BT(t,i) = 1$ if the column of the binary code corresponding to second symbol t of the pair of MOLS is included in relation R_i and 0 otherwise for $i = 0, 1$.
- $ST\_CLASS(s,t) = 2 * [(BS(s,1) + BT(t,1)) \text{ modulo } 2] + [(BS(s,2) + BT(t,2)) \text{ modulo } 2]$

ST_CLASS forms an equivalence relation for symbol pairs with four equivalence classes similar to the equivalence relation defined by RC_CLASS for cells. To satisfy the relations, symbol pair (s,t) can only be placed in cell (r,c) if $RC\_CLASS(r,c) = ST\_CLASS(s,t)$.

The cell classes and symbol pair classes are listed for each of the possible sets of relations in Section 2.2.1. Counting the sizes of each cell class and symbol pair class for the cases gives the following data:

Relation set	1				2				3			
Class	0	1	2	3	0	1	2	3	0	1	2	3
Symbol pairs	28	24	24	24	26	26	26	22	28	24	24	24
Cells	28	24	24	24	28	24	24	24	28	24	24	24

Relation set	4				5			
Class	0	1	2	3	0	1	2	3
Symbol pairs	28	24	24	24	28	24	24	24
Cells	26	26	26	22	28	24	24	24

Mismatch in the sizes of cell classes and symbol pair classes for cases 2 and 4 means that no such pairs of MOLS can exist. For the remaining three cases, generation of all pairs of MOLS satisfying them proceeds.

For all relations in consideration, the S and T sets of columns have the same columns chosen. This allows the definition of an equivalence class for symbols similar to that for cells and symbol pairs. For a symbol s and relations R_1 and R_2, consider the following definitions:

- $BZ(s, i) = 1$ if a column of the binary code corresponding to either first symbol s or second symbol s of the pair of MOLS is included in the relation R_i and 0 otherwise for $i = 1, 2$.
- $SYMBOLCLASS(s) = 2 * BZ(s, 1) + BZ(s, 2)$

Symbols s and s' are in the same equivalence class if $SYMBOLCLASS(s) = SYMBOLCLASS(s')$. Symbol permutations which preserve symbol class will also preserve the form of the relations.

2.2.1 Cell, Symbol Pair and Symbol Class Information

For each of the sets of relations, four equivalences classes exist for cells, symbol pairs and symbols. The following figures give information regarding these classes for each of the possible sets of relations:

Figure 2.1: Class by cell, R_1

0	1	1	1	2	2	2	3	3	3
1	0	0	0	3	3	3	2	2	2
1	0	0	0	3	3	3	2	2	2
1	0	0	0	3	3	3	2	2	2
2	3	3	3	0	0	0	1	1	1
2	3	3	3	0	0	0	1	1	1
2	3	3	3	0	0	0	1	1	1
3	2	2	2	1	1	1	0	0	0
3	2	2	2	1	1	1	0	0	0
3	2	2	2	1	1	1	0	0	0

Figure 2.2: Symbol pairs by class, R_1

Class 0	00 11 12 13 21 22 23 31 32 33 44 45 46 54 55 56 64 65 66 77 78 79 87 88 89 97 98 99
Class 1	01 02 03 10 20 30 47 48 49 57 58 59 67 68 69 74 75 76 84 85 86 94 95 96
Class 2	04 05 06 17 18 19 27 28 29 37 38 39 40 50 60 71 72 73 81 82 83 91 92 93
Class 3	07 08 09 14 15 16 24 25 26 34 35 36 41 42 43 51 52 53 61 62 63 70 80 90

Figure 2.3: Symbols by class, R_1

Symbol	0	1	2	3	4	5	6	7	8	9
Class	3	2	2	2	1	1	1	0	0	0

Figure 2.4: Class by cell, R_3

0	1	1	1	2	2	2	3	3	3
1	0	0	0	3	3	3	2	2	2
1	0	0	0	3	3	3	2	2	2
1	0	0	0	3	3	3	2	2	2
2	3	3	3	0	0	0	1	1	1
2	3	3	3	0	0	0	1	1	1
2	3	3	3	0	0	0	1	1	1
3	2	2	2	1	1	1	0	0	0
3	2	2	2	1	1	1	0	0	0
3	2	2	2	1	1	1	0	0	0

Figure 2.5: Symbol pairs by class, R_3

Class 0	00 01 10 11 22 23 32 33 44 45 54 55 66 67 68 69 76 77 78 79 86 87 88 89 96 97 98 99
Class 1	02 03 12 13 20 21 30 31 46 47 48 49 56 57 58 59 64 65 74 75 84 85 94 95
Class 2	04 05 14 15 26 27 28 29 36 37 38 39 40 41 50 51 62 63 72 73 82 83 92 93
Class 3	06 07 08 09 16 17 18 19 24 25 34 35 42 43 52 53 60 61 70 71 80 81 90 91

Figure 2.6: Symbols by class, R_3

Symbol	0	1	2	3	4	5	6	7	8	9
Class	3	2	2	2	1	1	1	0	0	0

Figure 2.7: Class by cell, R_5

0 0 1 1 2 2 3 3 3 3

0 0 1 1 2 2 3 3 3 3

1 1 0 0 3 3 2 2 2 2

1 1 0 0 3 3 2 2 2 2

2 2 3 3 0 0 1 1 1 1

2 2 3 3 0 0 1 1 1 1

3 3 2 2 1 1 0 0 0 0

3 3 2 2 1 1 0 0 0 0

3 3 2 2 1 1 0 0 0 0

3 3 2 2 1 1 0 0 0 0

Figure 2.8: Symbol pairs by class, R_5

Class 0	00 01 10 11 22 23 32 33 44 45 54 55 66 67 68 69 76 77 78 79 86 87 88 89 96 97 98 99
Class 1	02 03 12 13 20 21 30 31 46 47 48 49 56 57 58 59 64 65 74 75 84 85 94 95
Class 2	04 05 14 15 26 27 28 29 36 37 38 39 40 41 50 51 62 63 72 73 82 83 92 93
Class 3	06 07 08 09 16 17 18 19 24 25 34 35 42 43 52 53 60 61 70 71 80 81 90 91

Figure 2.9: Symbols by class, R_5

Symbol	0	1	2	3	4	5	6	7	8	9
Class	3	3	2	2	1	1	0	0	0	0

2.3 Filling the First Column and Row

Filling a pair of MOLS for select cells up to main class equivalence prior to completing the pair helps reduce the number of main class equivalent pairs of MOLS that will be produced. The method used to do so is described here.

For the selected cells to fill initially, the first column and row are chosen. As a column or row is filled, the remaining cells of that column or row become increasingly constrained by the Latin square property; each symbol pair present prevents the placement of future symbol pairs. Combined with restrictions imposed by relations, very few choices will be available for placing the last few symbol pairs in a column or row. This will help keep the total number of first column and row cases manageable.

The first column and row are filled recursively, the first column first, then the first row. For a given cell, all possible symbol pairs are considered. If a symbol pair is inconsistent with a cell for either forming a pair of MOLS or satisfying the relations under consideration, it is rejected. An additional set of constraints is imposed to reduce the number of first column and row cases generated which are main class equivalent.

Let $L[r][c]$ be the symbol pair located in cell (r, c) for pair of MOLS L. Define

symbol pair (s,t) to be less than symbol pair (s',t') if $s < s'$ or $s = s'$ and $t < t'$. A filled first column and row, L, is smaller than another filled first column and row, L', if for the first cell in which they differ in a symbol pair with respect to the order in which they are filled, the symbol pair in L is smaller than that in L'.

The basic properties that the cells and symbol pairs of a pair of MOLS must fulfill are:

- Latin square property: For two distinct cells, (r,c) and (r',c'), sharing a row or column, with $L[r][c] = (s,t)$ and $L[r'][c'] = (s',t')$, $s \neq s'$ and $t \neq t'$
- Orthogonality: $L[r][c] = L[r'][c'] \implies (r,c) = (r',c')$
- Relation constraints: For all cells (r,c), $RC\_CLASS(r,c) = ST\_CLASS(L[r][c])$

Satisfying the Latin square property and orthogonality ensures that what is generated is consistent with being a pair of MOLS. The relation constraints ensure that the current set of relations hold.

Additional constraints detect cases where a main class operation could potentially make the current first column and row smaller while preserving the form of the relations. The operations in consideration are: permutations of rows, permutations of columns, permutations of either symbol set S or T, exchanging columns R and C of the orthogonal array representation (taking the transpose), and exchanging columns S and T of the orthogonal array representation.

An algorithm which can produce a minimum symbol relabeling of a first column and row with respect to the order in which its cells are filled can be used to detect permutations of symbols sets S or T which could produce a smaller first column and row. In order to preserve the form of the relations, symbols are only permuted to

symbols of the same class. To find the minimum relabeling, when a symbol is first encountered, relabel it to the smallest available symbol in the same class. If this relabeling procedure does not produce the existing labeling, then a smaller relabeling exists and the current first column and row are not minimized with respect to a permutation of the symbol sets.

The following definitions are useful for the symbol relabeling algorithm. Let $alternate\_1[s]$ be the relabeled value of symbol s in the first Latin square. Define $alternate\_2[t]$ respectively. Initially, let $alternate\_1[s] = -1$ for all s and $alternate\_2[t] = -1$ for all t. The value -1 for $alternate\_1[s]$ indicates that symbol s has not yet been relabeled in the first Latin square. A value of -1 for $alternate\_2[t]$ similarly means that symbol t has not been relabeled in the second Latin square. Let $minimum\_1[x]$ be the minimum symbol with no symbol relabeled to it in symbol class x in the first Latin square. Define $minimum\_2[y]$ respectively.

{Attempt to Create M, a smaller first column and row than L}
SYMBOLRELABEL(L):
for $(r, c) = (0, 0)$ to $(9, 0)$ **do**
  $(s, t) \leftarrow L[r][c]$
  if $alternate\_1[s] = -1$ **then**
    $alternate\_1[s] \leftarrow minimum\_1[SYMBOLCLASS(s)]$
    $minimum\_1[CLASS(s)] \leftarrow minimum\_1[SYMBOLCLASS(s)] + 1$
  end if
  if $alternate\_2[t] = -1$ **then**
    $alternate\_2[t] \leftarrow minimum\_2[SYMBOLCLASS(t)]$
    $minimum\_2[CLASS(t)] \leftarrow minimum\_2[SYMBOLCLASS(t)] + 1$

end if

$M[r][c] \leftarrow (alternate\_1[a], alternate\_2[t])$

if $M[r][c] < L[r][c]$ **then**

return a smaller symbol relabeling exists

end if

end for

return no smaller symbol relabeling exists

The additional constraints applied to reduce main class duplication in filling in the first column and row while preserving the form of the relations are:

- Cells of same class in a column or row are sorted in increasing order by symbol pair
- Impose that $L[1][0] < L[0][1]$
- For the first cell (r, c) in the order which the first column is filled, $L[r][c] = (s, t)$, such that $s \neq t$, impose that $s < t$
- For any first column and row L, impose that SYMBOLRELABEL(L) finds no smaller symbol relabeling

Lemma 2.3.1. *For a choice of first column and row L, if there exists cells (r, c) and (r', c') in the same column or row such that $RC\_CLASS(r, c) = RC\_CLASS(r', c')$ and cell (r', c') is filled after cell (r, c), then if $L[r'][c'] < L[r][c]$, then a smaller choice of first column and row exists.*

Proof. Since $RC\_CLASS(r, c) = RC\_CLASS(r', c')$, a permutation of the columns or rows can swap the symbol pairs in cells (r, c) and (r', c') to form a new first column and row M respecting the original relations. For the first cell in which L and M differ, M is smaller as $L[r'][c'] < L[r][c]$. Therefore $M < L$, therefore a smaller first column and row with respect to main class operations exists. □

Lemma 2.3.2. *For a choice of first column and row L, if $L[0][1] < L[1][0]$, then a smaller choice of first column and row with respect to main class operations exists.*

Proof. Consider the transpose of L, M. Taking the transpose preserves relations, therefore M has the same relations as L. $M[0][0] = L[0][0]$, $M[1][0] = L[0][1]$. Since $L[0][1] < L[1][0]$, $M[1][0] < L[1][0]$. Therefore $M < L$, therefore a smaller choice of first column and row with respect to main class operations exists. □

Lemma 2.3.3. *For the first cell, (r, c), with symbol pair (s, t), of a first column and row L such that $s \neq t$, if $t < s$, then a smaller choice of first column and row with respect to main class operations exists.*

Proof. Consider the choice of first column and row M formed from L by swapping columns S and T in the orthogonal array representation. Swapping columns S and T of the orthogonal array representation will preserve the original relations, as the S and T columns have the same columns selected for all sets of relations under consideration. For all (r, c), if $L[r][c] = (s, t)$ then $M[r][c] = (t, s)$. Since for the first cell (r, c), $L[r][c] = (s, t)$, for which $s \neq t$, $t < s$ in L, the corresponding cell in M has symbol pair (t, s) such that $(t, s) < (s, t)$. Therefore $M < L$. Therefore a smaller choice of first column and row with respect to main class operations exists. □

Lemma 2.3.4. *For a choice of first column and row L, if SYMBOLRELABEL(L) returns that a smaller relabeling exists, then there exists a way to relabel the symbols so that the result still respects the form of the relations, but is smaller.*

Proof. SYMBOLRELABEL(L) only returns that a smaller relabeling exists if it manages to construct a smaller choice of first column and row M by permuting symbols of L in way respecting the form of the relations. □

For all of the additional constraints, if they are violated a smaller choice of first column and row will exist with respect to main class operations which preserve the

form of the relations. Therefore, any such minimum first column and row must conform to these constraints. Generating all first columns and rows with respect to these constraints will therefore generate all minimum first columns and rows with respect to main class operations which preserve the form of the relations.

After all choices of first column and row have been generated, they are divided up over different processors and serve as input to the main backtracking procedure. No further reduction in the number of main class equivalent pairs of MOLS created during the generation process occurs.

2.4 Backtracking to Complete Pairs of MOLS

It is necessary to complete the choices of first column and row produced in Section 2.3 in all ways such that the relations are present in order to generate all pairs of MOLS of order ten and dimension 35 or less. The method chosen to do so as quickly as possible is detailed here.

Given a pair of MOLS with only the first column and row completed, a backtracking procedure is used to produce all corresponding completed pairs of MOLS. This is done by placing symbol pairs in cells in the most constrained ways possible.

Any branch of the execution terminates early if there is a cell for which no symbol pair can be placed; backtracking occurs when this happens. When a completed pair of MOLS is created it is given as output, then the program backtracks to create any remaining pairs of MOLS satisfying the relations. In this phase, only restrictions related to the Latin square property, the relations, and orthogonality are considered

for which symbol pairs can be placed at a given cell.

Choosing a cell with a minimum number of symbol pairs placeable to that cell helps reduce the number of partially completed pairs of MOLS which are generated from any given partially completed pair of MOLS. Lowering this branching width means that partially completed pairs of MOLS which cannot be fully completed hopefully consume a smaller portion of the overall computation, resulting in faster execution.

Situations can occur where all cells have symbol pairs that can be placed at them, but there exists some symbol pair that cannot be placed in any cell. An example is shown in Figure 2.10, with Figure 2.11 giving the number of options available for each empty cell.

Figure 2.10: A partially filled pair of MOLS for relation case 5

00	67	46	95	29	73	18	34	52	81
66	–	–	–	–	28	–	–	–	–
47	21	55	78	60	06	39	82	14	93
58	–	–	–	–	–	–	–	–	–
72	–	–	–	–	–	–	–	–	–
83	–	–	–	–	–	–	–	–	–
19	–	–	–	–	–	–	–	–	–
24	–	–	–	–	–	–	–	–	–
35	–	–	–	–	–	–	–	–	–
91	–	–	–	–	–	–	–	–	–

Figure 2.11: Number of symbol pairs placeable in each cell of Figure 2.10

–	–	–	–	–	–	–	–	–	–
–	9	9	10	7	–	8	8	7	5
–	–	–	–	–	–	–	–	–	–
–	11	14	12	6	5	9	7	8	9
–	7	7	8	12	10	9	9	8	9
–	8	8	6	11	9	9	8	7	13
–	7	7	11	11	8	12	9	13	8
–	9	5	8	10	8	8	13	16	8
–	9	5	9	10	7	11	13	15	9
–	8	6	9	11	9	9	8	12	11

Despite each free cell in Figure 2.10 having at least five potential symbol pairs that can be placed, the symbol pair $(2, 5)$ cannot be placed in any of the free cells. The Latin square property constraints prevent $(2, 5)$ from being placed in all but the bottom right 6×4 block, for which the relations prevent placement.

Recognizing these cases allows earlier elimination of pairs of MOLS which cannot be completed. If there are fewer ways to assign cells to some symbol pair than there are ways to assign symbol pairs to any cell, the branching width, and thus overall running time, can potentially be reduced.

At each step of the backtrack, the algorithm computes for each unplaced symbol pair and empty cell the number of choices available to place the symbol pair or fill the cell. The symbol pair or cell with a minimum number of choices is either placed or filled in all ways possible.

The following algorithm describes how to determine which cell or symbol pair to fill in. For details regarding data structure implementations for counting the number of available symbol pairs or cells, see Sections 2.4.1 and 2.4.2.

{For partially filled pair of MOLS L, find the most constrained cell or symbol pair}
FINDMINIMUMCHOICE(L **):**
$min1 \leftarrow \infty$
for all choices of cell (r, c) **do**
  if $L[r][c]$ is not filled in **then**
    if the number of choices for $L[r][c]$ is less than $min1$ **then**
      $min1 \leftarrow$ the number of choices for $L[r][c]$
      $mincell \leftarrow (r, c)$
    end if
  end if
end for
$min2 \leftarrow \infty$
for all choices of symbol pair (s, t) **do**
  if symbol pair (s, t) has not been placed **then**
    if the number of ways to place symbol pair (s, t) is less than $min2$ **then**
      $min2 \leftarrow$ the number of ways to place symbol pair (s, t)
      $minsymbolpair \leftarrow (s, t)$
    end if
  end if
end for
if $min1 < min2$ **then**

return the minimum choice is cell *mincell*

else

return the minimum choice is symbol pair *minsymbolpair*

end if

Using the above algorithm to determine which cell or symbol pair has the least number of choices to fill, the following algorithm recursively generates all pairs of MOLS satisfying the relations. For a partially completed pair of MOLS L, FILLMOLS(L) will print all pairs of MOLS completable from L satisfying the relations and return how many pairs of MOLS were completed.

{For pair of MOLS L, recursively complete L in all ways possible}

FILLMOLS(L **):**

if pair of MOLS L is complete **then**

print L

return (1)

end if

$minimumchoice \leftarrow$ FINDMINIMUMCHOICE(L)

$MOLScount \leftarrow 0$

if the *minimumchoice* is a cell **then**

$(r, c) \leftarrow minimumchoice$

for all (s, t) such that (s, t) can be placed in cell (r, c) **do**

$L[r][c] \leftarrow (s, t)$

$MOLScount \leftarrow MOLScount + FILLMOLS(L)$

$L[r][c] \leftarrow empty$

end for

else {*minimumchoice* is a symbol pair}

$(s,t) \leftarrow minimumchoice$

for all (r,c) such that (s,t) can be placed in cell (r,c) **do**

$L[r][c] \leftarrow (s,t)$

$MOLScount \leftarrow MOLScount + FILLMOLS(L)$

$L[r][c] \leftarrow empty$

end for

end if

return $(MOLScount)$

2.4.1 Implementing Sets with Bit Arrays

Set data structures are needed to track which symbol pairs or cells are available for a given symbol pair or cell. An implementation of these sets is described here. For an alternate implementation providing improved speed, see Section 2.4.2.

Counting the number of symbol pairs that can be placed at each empty cell and how many cells each unplaced symbol pair can be placed to is an intensive task repeated at every step of the backtrack. Initially, there will be 81 empty cells and 81 unplaced symbol pairs. This provides strong incentive to do these calculations in the least time possible.

For each symbol pair there is an associated set of cells to which the symbol pair can be placed. Similarly, for each cell there is a set of symbol pairs which can be placed in the cell. The function that counts the size of these sets returns the number of available choices for a cell or symbol pair. To reduce the overhead of updating sets, instead of storing one set for each cell and symbol pair, sets for cells and symbol pairs are computed from sets requiring fewer updates. The following sets are used:

- $SymbolPairsByRow[r]$: The set of symbol pairs that can be placed in row r without violating the Latin square property for the row, $r = 0, \ldots, 9$
- $SymbolPairsByColumn[c]$: The set of symbol pairs that can be placed in column c without violating the Latin square property for the column, $c = 0, \ldots, 9$
- $SymbolPairsAvailable$: The set of symbol pairs which have not yet been placed
- $SymbolPairsByClass[RC\_CLASS(r, c)]$: The set of symbol pairs with class matching that of cell (r, c)
- $CellsByFirstSymbol[s]$: The set of cells for which a symbol pair with first symbol s can be placed without violating the Latin square property, $s = 0, \ldots, 9$
- $CellsBySecondSymbol[t]$: The set of cells for which a symbol pair with second symbol t can be placed without violating the Latin square property, $t = 0, \ldots, 9$
- $CellsEmpty$: The set of cells which have no symbol pair
- $CellsByClass[ST\_CLASS(s, t)]$: The set of cells with class matching that of symbol pair (s, t)

To calculate the symbol pair set for a given cell (r, c):

$$\begin{aligned} SymbolPairsByCell[r][c] = \; & SymbolPairsByRow[r] \cap \\ & SymbolPairsByColumn[c] \cap \\ & SymbolPairsAvailable \cap \\ & SymbolPairsByClass[RC\_CLASS(r, c)] \end{aligned}$$

To calculate the cell set for a symbol pair (s, t):

$$\begin{aligned} CellsBySymbolPair[s][t] = \; & CellsByFirstSymbol[s] \cap \\ & CellsBySecondSymbol[t] \cap \\ & CellsEmpty \cap \\ & CellsByClass[ST\_CLASS(s, t)] \end{aligned}$$

When placing or removing a symbol pair (s, t) from a cell (r, c), the following six sets need to be updated:

- $SymbolPairsByRow[r]$
- $SymbolPairsByColumn[c]$
- $SymbolPairsAvailable$
- $CellsByFirstSymbol[s]$
- $CellsBySecondSymbol[t]$
- $CellsEmpty$.

For quick execution, the sets are implemented as arrays of binary values (bit arrays). A 32-bit or 64-bit integer can hold respectively 32 or 64 binary values each. These sets, of size 100, would require either two 64-bit integers or four 32-bit integers.

Other than being very compact with respect to storage, an advantage of using bit arrays is that intersection of two sets can be very quickly computed using bitwise-and operations. This will aid in the frequent calculations for such sets for the symbol pairs and cells.

To aid in updating the following bit arrays:

- $SymbolPairsByRow[r]$
- $SymbolPairsByColumn[c]$
- $CellsByFirstSymbol[s]$
- $CellsBySecondSymbol[t]$

The following precomputed bit masks are used:

- $SymbolPairsWithoutFirstSymbol[s]$: Contains all symbol pairs without first symbol s, $s = 0, \ldots, 9$

- $SymbolPairsWithoutSecondSymbol[t]$: Contains all symbol pairs without second symbol t, $t = 0, \ldots, 9$
- $CellsWithoutRow[r]$: Contains all cells without row r, $r = 0, \ldots, 9$
- $CellsWithoutColumn[c]$: Contains all cells without column c, $c = 0, \ldots, 9$

When symbol pair (s, t) is placed in cell (r, c), the following updates can be performed using only the intersection of sets:

$$\begin{aligned} SymbolPairsByRow[r] \leftarrow\ & SymbolPairsByRow[r] \cap \\ & SymbolPairsWithoutFirstSymbol[s] \cap \\ & SymbolPairsWithoutSecondSymbol[t] \end{aligned}$$

$$\begin{aligned} SymbolPairsByColumn[c] \leftarrow\ & SymbolPairsByColumn[c] \cap \\ & SymbolPairsWithoutFirstSymbol[s] \cap \\ & SymbolPairsWithoutSecondSymbol[t] \end{aligned}$$

$$\begin{aligned} CellsByFirstSymbol[s] \leftarrow\ & CellsByFirstSymbol[s] \cap \\ & CellsWithoutRow[r] \cap \\ & CellsWithoutColumn[c] \end{aligned}$$

$$\begin{aligned} CellsBySecondSymbol[t] \leftarrow\ & CellsBySecondSymbol[t] \cap \\ & CellsWithoutRow[r] \cap \\ & CellsWithoutColumn[c] \end{aligned}$$

Counting the number of entries in a set stored in a bit array is done in 16-bit chunks. A precomputed table holds the number of 1-bits for any given 16-bit value. Adding these indexed values together gives the set size.

2.4.2 Implementing Sets using Reduced Size Bit Arrays

The set implementation of Section 2.4.1 can be improved to reduce the amount of time spent counting which symbol pairs and cells are available for a given symbol pair or cell. The improved implementation is described here.

Bit arrays of size 100 require at minimum two 64-bit or four 32-bit integers to store. The classes for symbol pairs and cells contain at most 28 elements for the relations of interest. If for each cell and symbol pair we restrict storage to those symbol pairs and cells of matching equivalence class, we can reduce the size of the bit arrays so they can be stored in a single 32-bit integer.

The row, column, first symbol, and second symbol bit arrays are replaced with four bit arrays each (one for each class). These new arrays occupy 32 bits as opposed to the original 128 bits each. Existing availability arrays are merged with the class arrays and altered to the reduced size. The cells or symbol pairs represented by the new bit arrays are mapped to their actual values using an index array.

The new sets are:

- $SymbolPairsByRow[r][x]$: The set of symbol pairs in class x that can be placed in row r without violating the Latin square property for the row
- $SymbolPairsByColumn[c][x]$: The set of symbol pairs in class x that can be placed in column c without violating the Latin square property for the column
- $SymbolPairsByClass[x]$: The set of symbol pairs with class x
- $CellsByFirstSymbol[s][x]$: The set of cells in class x for which a symbol pair with first symbol s can be placed without violating the Latin square property
- $CellsBySecondSymbol[t][x]$: The set of cells in class x for which a symbol pair

with second symbol t can be placed without violating the Latin square property

- $CellsByClass[x]$: The set of cells with class x

To calculate the symbol pair set for a given cell (r, c):

$$\begin{aligned} SymbolPairsByCell[r][c] = \; & SymbolPairsByRow[r][RC\_CLASS(r,c)] \cap \\ & SymbolPairsByColumn[c][RC\_CLASS(r,c)] \cap \\ & SymbolPairsByClass[RC\_CLASS(r,c)] \end{aligned}$$

To calculate the cell set for a symbol pair (s, t):

$$\begin{aligned} CellsBySymbolPair[s][t] = \; & CellsByFirstSymbol[s][ST\_CLASS(s,t)] \cap \\ & CellsBySecondSymbol[t][ST\_CLASS(s,t)] \cap \\ & CellsByClass[ST\_CLASS(ST\_(s,t))] \end{aligned}$$

To aid in updating the following bit arrays:

- $SymbolPairsByRow[r][x]$
- $SymbolPairsByColumn[c][x]$
- $CellsByFirstSymbol[s][x]$
- $CellsBySecondSymbol[t][x]$

We use the following precomputed bit masks:

- $SymbolPairsWithoutFirstSymbol[s][x]$: Contains all symbol pairs of class x without first symbol s
- $SymbolPairsWithoutSecondSymbol[t][x]$: Contains all symbol pairs of class x without second symbol t
- $CellsWithoutRow[r][x]$: Contains all cells of class x without row r

- $CellsWithoutColumn[c][x]$: Contains all cells of class x without column c

When placing symbol pair (s,t) in cell (r,c) with $RC\_CLASS(r,c) = x$, updates occur as follow:

$$\begin{aligned} SymbolPairsByRow[r][x] \leftarrow\ & SymbolPairsByRow[r][x] \cap \\ & SymbolPairsWithoutFirstSymbol[s][x] \cap \\ & SymbolPairsWithoutSecondSymbol[t][x] \end{aligned}$$

$$\begin{aligned} SymbolPairsByColumn[c][x] \leftarrow\ & SymbolPairsByColumn[c][x] \cap \\ & SymbolPairsWithoutFirstSymbol[s][x] \cap \\ & SymbolPairsWithoutSecondSymbol[t][x] \end{aligned}$$

$$\begin{aligned} CellsByFirstSymbol[s][x] \leftarrow\ & CellsByFirstSymbol[s][x] \cap \\ & CellsWithoutRow[r][x] \cap \\ & CellsWithoutColumn[c][x] \end{aligned}$$

$$\begin{aligned} CellsBySecondSymbol[t][x] \leftarrow\ & CellsBySecondSymbol[t][x] \cap \\ & CellsWithoutRow[r][x] \cap \\ & CellsWithoutColumn[c][x] \end{aligned}$$

Relative to the implementation of Section 2.4.1, more bit arrays are updated. For the implementation of Section 2.4.1, 24 bitwise-and operations are required to update the data structures if 64-bit integers are used. For this implementation, 48 bitwise-and operations are required. Counting is faster in this implementation, as 28 bits can be counted with two 16-bit lookups, whereas 100 bits requires seven 16-bit lookups. If there are n empty cells, the implementation of Section 2.4.1 will use $16n$ bitwise-and

operations in computing the sets to count, this implementation will use $6n$.

Initially there are 81 empty cells, 81 unplaced symbol pairs and at most 28 possible ways to place a symbol pair at a cell due to class sizes. After placing a symbol pair, the number of empty cells is reduced by one; the number of ways to match a symbol pair with a cell has the potential to be reduced by more than one. There can not be fewer empty cells than there are ways to match a symbol pair with a cell. Counting operations therefore occur more frequently than updating operations. The faster counting operations of this implementation should therefore result in reduced execution time.

2.5 Canonical Form Testing

After generating all pairs of MOLS, some of these pairs may be main class equivalent. In order to meaningfully count the generated pairs of MOLS, a canonical form with respect to main class equivalence for each pair of MOLS is used. The canonical form is described here.

Note that isotopy and main class operations do not always preserve the original relations. Two pairs of MOLS are stored for each generated pair of MOLS: the generated pair which preserves the relations, and the canonical pair which does not.

The canonical form routine used for pairs of MOLS is adapted from the canonical form routine for Latin squares of McKay, Myrvold, and Meynert [19]. First the canonical form for Latin squares with respect to isotopy is described. This canonical form is altered to be used for a pair of MOLS, then for main class rather than isotopic

canonicity.

Consider the first two rows of a Latin square with the first row in reduced form. The second row can be interpreted as a permutation of the first. We can use a cycle based notation to represent this permutation.

The following is two rows of a Latin square and an associated permutation cycle structure:

$$\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 0 & 3 & 6 & 5 & 2 & 4 & 8 & 9 & 7 \end{array}$$

$$(01)(23645)(789)$$

We use the cycle structure to define a standard form for the first two rows. In the standard form, the first row is in reduced form, cycles are contiguous, cycles are ordered largest to smallest, and cycles go from left to right.

Some combinations of column and symbol permutations can rearrange the cycles in the second row to an order conforming to this standard. Here are the first two rows of the earlier example, after permuting columns and relabeling the symbols to put it in standard form:

$$\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 0 & 6 & 7 & 5 & 9 & 8 \end{array}$$

The new cycle structure is:

$$(01234)(567)(89)$$

The *penalty* of a cycle structure is the number of column permutations preserving the standard form after symbol relabeling. The first two rows are chosen such that

the penalty is minimized. The advantage of insisting that the first two rows are in this standard form is that it reduces the number of column permutations considered. Under a more simplistic approach, all 10! column permutations would be considered. We stipulate that the first column is in reduced form, which means that given choice of first column and a symbol permutation, there is only one row permutation preserving the standard form.

Table 2.1: Penalties for cycle structures of order ten

Cycle Structure	Penalty
(10)	10
(8)(2)	$8 \times 2 = 16$
(7)(3)	$7 \times 3 = 21$
(6)(4)	$6 \times 4 = 24$
(5)(3)(2)	$5 \times 3 \times 2 = 30$
(6)(2)(2)	$6 \times 2 \times 2 \times 2! = 48$
(5)(5)	$5 \times 5 \times 2! = 50$
(4)(4)(2)	$4 \times 4 \times 2 \times 2! = 64$
(4)(3)(3)	$4 \times 3 \times 3 \times 2! = 72$
(3)(3)(2)(2)	$3 \times 3 \times 2 \times 2 \times 2! \times 2! = 144$
(4)(2)(2)(2)	$4 \times 2 \times 2 \times 2 \times 3! = 192$
(2)(2)(2)(2)(2)	$2 \times 2 \times 2 \times 2 \times 2 \times 5! = 3840$

If all possible standard forms for a given Latin square are compared, the lexicographical minimum can be taken as the canonical representative. Lexicographic order for a Latin square, or a pair of MOLS, is row by row, left to right from top to bottom.

The following is an algorithm describing this process:

{Return the canonical representative of Latin square L}
CANONICALLATINSQUARE(L)
$canL \leftarrow$ null
for all choices of first two rows of L such that the penalty is minimum **do**
  for all column and symbol permutations that will achieve standard form **do**
    Choose the unique row permutation to sort the first column
    { Let L' be the Latin square constructed from L via these permutations}
    if $L' < canL$ **then**
      $canL \leftarrow L$
    end if
  end for
end for
return the canonical representative of L, $canL$

To expand this routine from a single Latin square to get a canonical form for the isotopy class of a pair of MOLS, apply the canonical form procedure to the first Latin square of the pair. Only a choice of symbol permutation for the second square remains. Putting the first row of the second square into sorted order selects a unique second symbol permutation. Take the lexicographically minimum candidate to be the canonical form for the pair of MOLS in question.

To obtain a main class canonical form, prior to considering all choices of first and second row put the Latin squares into the orthogonal array representation. All 24 permutations of the columns of the orthogonal array representation, which correspond to exchanging the roles of rows, columns, and symbols, are considered to obtain the

main class canonical form for the pair of MOLS. The following algorithm describes the process:

CANONICALPAIROFMOLS(L **)**

$canL \leftarrow$ null

for all permutations of the columns of the orthogonal array representation of L **do**

{Let L_1 be the first Latin square of pair of MOLS L}

Use **CANONICALLATINSQUARE(** L_1 **)** to put the first Latin square of L into canonical form

Choose the second symbol permutation which puts the first row of the second Latin square in sorted order

{ Let L' be the pair of MOLS constructed from L via these permutations}

if $L' < canL$ for a lexicographical comparison **then**

$canL \leftarrow L$

end if

end for

return the canonical representative of L, $canL$

2.6 Triple Detection

Once all pairs of MOLS of order ten and dimension 35 or less are generated, the next task determining the extendability of these pairs to triples. Described here is a method for checking the extendability of a pair of MOLS of order ten.

A pair of MOLS of order ten can be extended to a triple of MOLS if and only

if the pair has ten mutually disjoint transversals. Determining if a pair of MOLS extends to a triple of MOLS is resolved by finding the maximum number of disjoint transversals for the pair. If there are ten disjoint transversals for the pair of MOLS, a triple can be constructed.

To determine the number of disjoint transversals for a pair of MOLS, the following steps are used:

1. All transversals of the pair of MOLS are generated
2. A graph, G, with the generated transversals as vertices is created
3. Edges are added between pairs of vertices of G which represent disjoint transversals
4. A maximum clique of G is computed, the order of this clique is the number of disjoint transversals

Transversals of the pairs of MOLS are generated with a backtracking procedure. For each row, a column which has not been previously selected is chosen to form a cell of the transversal. If any two chosen cells have symbol pairs which share a first or second symbol, the current transversal is rejected.

Cliques of G are generated exhaustively by backtracking. Starting with zero vertices, cliques are expanded by a single vertex in all ways possible. All cliques of G are generated and a maximum clique is recorded. This algorithm is chosen knowing the size and edge density of G are small.

Chapter 3

Results

The generation of all pairs of MOLS of order ten and dimension 35 or less described in Chapter 2 was programmed in C. Results and related analysis of the generation are described here.

The primary goal of this work was to generate all pairs of MOLS of dimension 35 or less in reasonable time. Interest in program speed was focused on the feasibility of this generation. A group of 64 Pentium 4 processors were available for use.

The number of partially completed pairs of MOLS created by filling in the first column and row are as follows:

Relation Case	1	3	5
First Column and Row Cases	25,782	12,413	33,745

Time for completion was under three seconds for each set of relations. Given the negligible time that generation of the first row and column took, a more aggressive approach to eliminating isotopic and main class symmetries may have improved the speed of the backtracking portion of the computation. Filling in additional cells may

have led further reductions in symmetries. This would benefit the exhaustive completion of the pairs of MOLS through a reduced search space.

An independent implementation of the algorithm for filling in the first row and column was undertaken [22]. Results from the independent implementation concur with the generated counts listed here. For any computational result, this independent verification is important for ensuring that there were no errors in the implementation.

Numbers of completed pairs of MOLS generated by the backtracking procedure are as follows:

Relation Case	1	3	5
Pairs of MOLS	7,808	0	22,320

Individual cases as divided by the different ways to fill in the first row and column differed in time to completion. Conflicting CPU intensive processes on occasion running alongside these programs interfered with gathering timing data. On the 64 machines, the following approximate times were taken to complete the generation of the pairs of MOLS:

Relation Case	1	3	5
Total Time for Generation (CPU days)	326	23	139

Overall, generation took a total of 488 CPU days, or 7.6 days per machine. Less interference may have resulted in slightly less overall time.

The exhaustive completion of these pairs of MOLS was again implemented independently to confirm the results were accurate [22]. All generated counts from the

independent implementation concurred with those listed here.

Interestingly, no pairs of MOLS are completable for case 3. Some theoretical conditions possibly exist to explain this, but are not known at this time. For the other relation sets the number of pairs of MOLS generated is small.

After running through the canonical form routine, the number of canonical Latin squares and canonical pairs of MOLS (with respected to main class equivalence) are as follows:

Dimension	Relation Case	Initial MOLS	Canonical Latin squares	Canonical MOLS
34	5	1,440	4	6
35	5	20,880	178	78
35	1	7,808	11	7

No pairs of MOLS of dimension 33 exist. A total of six pairs of MOLS exist up to main class equivalence for dimension 34. All of these pairs are counterexamples to Moorehouse's conjecture [20]. Counterexamples have been previously found [15], but these six form all of the dimension 34 pairs of MOLS (up to main class equivalence) once conjectured not to exist. See Section 3.1 for these pairs of MOLS.

Counts for the canonical form routine were checked against an independent implementation [22]. The number of canonical Latin squares and MOLS for the independent implementation concur with the results listed here.

With few total pairs of MOLS generated by the exhaustive backtrack, the canonical form routine took negligible time. Analyzing the transversals of the canonical

pairs of MOLS gives the following information:

Transversals by number and frequency:

Dimension	34		35				35			
Relation Case	5		5				1			
Number of Transversals	0	21	0	1	2	3	0	1	4	16
Frequency	2	4	36	34	7	1	1	4	1	1

Only five pairs of MOLS have more than ten transversals, the minimum required to achieve ten disjoint transversals and form a triple. These five pairs of MOLS have at most three disjoint transversals. Only one pair of MOLS for dimension 35 has more than ten transversals; this pair of MOLS is given in Section 3.1. Though finding maximum cliques is NP-complete, the small number of transversals and disjoint transversals meant that the graphs had few vertices and low edge density, making computation easy.

Disjoint transversals by number and frequency:

Dimension	34		35			35	
Relation Case	5		5			1	
Number of Disjoint Transversals	0	3	0	1	2	0	1
Frequency	2	4	33	37	5	1	6

An existing implementation of the triple detection algorithm by Myrvold [22] was used instead of creating a new implementation. This implementation was tested against an alternate implementation [19], which also found no triple of MOLS. The alternate implementation did not give frequency counts for transversals or disjoint transversals with which to compare.

The maximum number of disjoint transversals amongst the pairs generated, three, is low, and does not come close to the ten required to form a triple of MOLS. The number of disjoint transversals was higher for the dimension 34 squares than that of the dimension 35.

3.1 Pairs of MOLS of Interest

Of the generated pairs of MOLS, six rank 34 counterexamples to Moorehouse's conjecture were found (shown in Figures 3.1 to 3.6). One of the rank 35 pairs of MOLS possessed 16 transversals (shown if Figure 3.7), the second largest number of transversals for rank 35 pairs of MOLS being only 4. These seven pairs of MOLS of interest are listed here.

Figure 3.1: Dimension 34 pair of MOLS 1, no transversals

00	79	56	95	38	83	17	24	42	61
66	11	48	57	93	72	80	09	25	34
47	94	89	32	70	18	26	63	51	05
74	85	23	68	16	07	92	41	39	50
28	36	90	81	67	54	75	12	03	49
82	27	71	06	45	99	58	30	64	13
19	52	04	40	21	65	33	76	87	98
35	43	62	14	59	20	01	88	96	77
53	60	15	29	02	31	44	97	78	86
91	08	37	73	84	46	69	55	10	22

Figure 3.2: Dimension 34 pair of MOLS 2, no transversals

00	79	56	95	38	83	17	24	42	61
66	11	48	57	93	72	80	09	25	34
47	94	89	32	70	18	26	63	51	05
74	85	23	68	16	07	92	41	39	50
28	36	90	81	67	54	75	12	03	49
82	27	71	06	45	99	58	30	64	13
19	52	04	40	21	65	33	78	87	96
35	43	62	14	59	20	01	86	98	77
53	60	15	29	02	31	44	97	76	88
91	08	37	73	84	46	69	55	10	22

Figure 3.3: Dimension 34 pair of MOLS 3, 21 transversals (3 disjoint)

00	79	56	95	38	83	17	24	42	61
66	11	48	57	93	72	80	09	25	34
47	94	89	32	70	18	26	63	51	05
74	85	23	68	16	07	92	41	39	50
28	36	90	81	67	54	75	12	03	49
82	27	71	06	45	99	58	30	64	13
19	52	04	40	21	65	33	77	88	96
35	43	62	14	59	20	01	86	97	78
53	60	15	29	02	31	44	98	76	87
91	08	37	73	84	46	69	55	10	22

Figure 3.4: Dimension 34 pair of MOLS 4, 21 transversals (3 disjoint)

00	79	56	95	38	83	17	24	42	61
66	11	48	57	93	72	80	09	25	34
47	94	89	32	70	18	26	63	51	05
74	85	23	68	16	07	92	41	39	50
28	36	90	81	67	54	75	12	03	49
82	27	71	06	45	99	58	30	64	13
19	52	04	40	21	65	33	76	88	97
35	43	62	14	59	20	01	87	96	78
53	60	15	29	02	31	44	98	77	86
91	08	37	73	84	46	69	55	10	22

Figure 3.5: Dimension 34 pair of MOLS 5, 21 transversals (3 disjoint)

00	79	56	95	38	83	17	24	42	61
66	11	48	57	93	72	80	09	25	34
47	94	89	32	70	18	26	63	51	05
74	85	23	68	16	07	92	41	39	50
28	36	90	81	67	54	75	12	03	49
82	27	71	06	45	99	58	30	64	13
19	52	04	40	21	65	33	78	86	97
35	43	62	14	59	20	01	87	98	76
53	60	15	29	02	31	44	96	77	88
91	08	37	73	84	46	69	55	10	22

Figure 3.6: Dimension 34 pair of MOLS 6, 21 transversals (3 disjoint)

00	79	56	95	38	83	17	24	42	61
66	11	48	57	93	72	80	09	25	34
47	94	89	32	70	18	26	63	51	05
74	85	23	68	16	07	92	41	39	50
28	36	90	81	67	54	75	12	03	49
82	27	71	06	45	99	58	30	64	13
19	52	04	40	21	65	33	78	86	97
35	43	62	14	59	20	01	87	98	76
53	60	15	29	02	31	44	96	77	88
91	08	37	73	84	46	69	55	10	22

Figure 3.7: Dimension 35 pair of MOLS, 16 transversals (0 disjoint)

00	76	85	94	17	29	38	43	51	62
47	98	22	56	70	15	63	81	39	04
58	65	79	13	34	41	80	06	92	27
69	31	44	87	52	90	26	18	05	73
71	07	53	35	89	64	12	20	48	96
82	24	08	61	46	33	97	75	10	59
93	42	16	09	21	78	55	67	84	30
14	50	37	72	68	86	01	99	23	45
25	19	91	40	03	57	74	32	66	88
36	83	60	28	95	02	49	54	77	11

Chapter 4

Conclusions and Future Work

Concluding remarks about this work are found in Section 4.1. Potential future research directions originating from this work can be found in Section 4.2.

4.1 Conclusions

All order ten pairs of MOLS for dimension 35 and lower were successfully generated. A feasible procedure for generating these MOLS was developed and used to do so.

No triple of mutually orthogonal Latin squares of order ten could be created from a pair of MOLS of order ten with rank 35 or less. No pairs of orthogonal Latin squares of rank 33 exists.

Six order ten dimension 34 pairs of MOLS, which are counterexamples to Moorehouse's conjecture [20], were generated up to main class equivalence. Counterexamples were already known [15], but these six form all such dimension 34 counterexamples up to main class equivalence.

The generated pairs of MOLS had at most 21 transversals, with at most three disjoint transversals. The best known result is 21 transversals, with four disjoint [7, 22]. None of the dimension 35 pairs of MOLS have more than two disjoint transversals.

All results were double checked by an alternate implementation [22]. All counts for generated results matched for filling in the first row and column, completing the pairs of MOLS, and finding canonical representatives for pairs of MOLS. No triples of MOLS were found by either implementation. These matching results increase confidence that no implementation errors were present.

4.2 Future Work

Though all pairs of MOLS of order ten and dimension 35 or less were generated, questions such as the existence of a triple of MOLS of order ten remain unanswered. Potential future research building on this work is described here.

Having generated all order ten pairs of MOLS and dimension 35 or less, the next obvious step is to generate all pairs of MOLS of rank 36. With one less relation the size of the symbol pair and position classes will approximately double. This will result in many more choices for as how to place symbol pairs in the Latin squares at each stage. With the current strategy, this would lead to a large increase in the overall generation time. Given this and the time taken for generating all pairs of MOLS of rank 35 or less, some additional ways of significantly reducing generation time will be needed to apply a similar strategy to rank 36.

After rank 36, the total number of rank 37 pairs of MOLS is prohibitively large

to generate [19]. Isolating focus to finding pairs of MOLS which might be extendable to triples may allow further progress in determining the existence of a triple of MOLS.

Negative theoretical conditions for triples of MOLS may be key in providing the necessary restrictions for generating pairs of MOLS which might possibly be extended to triples. Existing negative conditions, when computationally practical, should be be used as additional generation constraints. More theorems regarding negative conditions will likely need to be developed. Determining why no pairs of MOLS could be generated for relation set 3 may help find these negative conditions.

To speed up the backtracker, the number of cells available to each symbol pair is calculated alongside symbol pairs available to cells. In terms of the orthogonal array representation, available options for (R,C) pairs and (S,T) pairs were calculated. We could also consider choices available to any two of the columns, such as (R,S), which would calculate choices of column c and second symbol t available for each choice of a row r and first symbol s. Analyzing all six possible sets of constraints may result in further speed improvements. Future work is to do timing experiments to determine if these additional constraints have an overall beneficial effect.

The pairs of MOLS generated had at most three mutually disjoint transversals. Pairs of MOLS have been found having four disjoint transversals [7, 22], but it is still an open question if there exists a pair of MOLS of order ten with five or more disjoint transversals. With so few disjoint transversals in most pairs of MOLS, restricting attention to pairs of MOLS with at least a certain number of transversals may be ideal for answering this question. Attempting to place a small number of disjoint transversals in all ways possible would aid in the search for a triple of MOLS by

eliminating large quantities of pairs which can not be extended to a triple.

Bibliography

[1] R. C. Bose, I. M. Chakravarti, and D. E. Knuth. On methods of constructing sets of mutually orthogonal Latin squares using a computer. I. *Technometrics*, 2:507–516, 1960.

[2] R. C. Bose, I. M. Chakravarti, and D. E. Knuth. On methods of constructing sets of mutually orthogonal Latin squares using a computer. II. *Technometrics*, 3:111–117, 1961.

[3] R. C. Bose and S. S. Shrikhande. On the falsity of Eulers conjecture about the non-existence of two orthogonal Latin squares of order 4t + 2. *Proceedings of the National Academy of Sciences*, 45:734–737, 1959.

[4] R. C. Bose, S. S. Shrikhande, and E. T. Parker. Further results on the construction of mutually orthogonal Latin squares and the falsity of Eulers conjecture. *Canadian Journal of Mathematics*, 12:189–203, 1960.

[5] A. E. Brouwer. Four mols of order 10 with a hole of order 2. *Journal of Statistical Planning and Inference*, 10:203–205, 1984.

[6] J. W. Brown and E. T. Parker. Some attempts to construct orthogonal Latin squares. *Congressus Numerantium*, 43:201–202, 1984.

[7] J. W. Brown and E. T. Parker. More on order 10 turn-squares. *Ars Combinatoria*, 35:125–127, 1993.

[8] R. H. Bruck. Finite nets. II. uniqueness and embedding. *Pacific Journal of Mathematics*, 13:421–457, 1963.

[9] C. J. Colbourn and J. H. Dinitz. Mutually orthogonal latin squares: a brief survey of constructions. *Journal of Statistical Planning and Inference*, 95:9–48, 2001.

[10] C. J. Colbourn and J. H. Dinitz. *The CRC Handbook of Combinatorial Designs*. CRC Press, Boca Raton, second edition edition, 2007.

[11] J. Dénes and A. D. Keedwell. *Latin Squares and their Applications*. The English Universities Press, London, 1974.

[12] J. Dénes and A. D. Keedwell. *Latin Squares: New Developments in the Theory and Applications*. North-Holland, Amsterdam, 1991.

[13] L. Euler. Recherches sur une nouvelle espèce de carrés magiques. *Verhandelingen / uitgegeven door het zeeuwsch Genootschap der Wetenschappen te Vlissingen*, 9:85–239, 1782.

[14] L. Howard. *Nets of Order 4m + 2: Linear Dependence and Dimensions of Codes*. PhD thesis, University of Victoria, 2009.

[15] L. Howard and W. Myrvold. A counterexample to Moorhouse's conjecture on the rank of nets. *Accepted to the Bulletin of the ICA*, November 2009. 4 pages.

[16] A. D. Keedwell. Concerning the existence of triples of pairwise almost orthogonal 10x10 Latin squares. *Ars Combinatoria*, 9:3–10, 1980.

[17] C. W. H. Lam, L. Thiel, and S. Swiercz. The non-existence of finite projective planes of order 10. *Canadian Journal of Mathematics*, 41:1117–1123, 1989.

[18] B. D. McKay, J. C. McLeod, and I. M. Wanless. The number of transversals in a Latin square. *Designs, Codes and Cryptography*, 40:269–284, 2006.

[19] B. D. McKay, A. Meynert, and W. Myrvold. Small Latin squares, quasigroups and loops. *Journal of Combinatorial Designs*, 15:98–119, 2007.

[20] G. E. Moorhouse. Bruck nets, codes and characters of loops. *Designs, Codes and Cryptography*, 1:7–29, 1991.

[21] W. Myrvold. Negative results for orthogonal triples of Latin squares of order 10. *The Journal of Combinatorial Mathematics and Combinatorial Computing*, 29:95–105, 1999.

[22] W. Myrvold. Private Communication, 2010.

[23] E. T. Parker. Nonextendibility conditions on mutually orthogonal Latin squares. *Proceedings of the American Mathematical Society*, 13:219–221, 1962.

[24] E. T. Parker. On orthogonal Latin squares. In *1960 Institute on Finite Groups*, volume 6, pages 43–36, 1962.

[25] E. T. Parker. Computer investigation of orthogonal Latin squares of order ten. *Proceedings of Symposia in Pure Mathematics*, 15:73–81, 1963.

[26] E. T. Parker. A collapsed image of a completion of a "turn-square". *Journal of Combinatorial Theory*, 24:128–129, 1978.

[27] E. T. Parker and J. W. Brown. A try for three order-10 orthogonal Latin squares. *Congressus Numerantium*, 36:43–46, 1982.

[28] G. Tarry. Le problème des 36 officers. *Compte Rendu de l'Association franaise pour l'avancement de science naturel*, 2:170–203, 1901.