Factor Pair Latin Squares

by

James M. Hammer, III

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 10, 2015

Keywords: Factor Pair, Latin Square, Factor Pair Latin Square, Quasi-Factor Pair Latin Square, Sudoku Design, Factor Pair Latin Square Design, Gerechte Design, Mutliple Gerechte Design

Approved by

Dean G. Hoffman, Professor, Professor of Mathematics
Chris A. Rodger, Don Logan Endowed Chair in Mathematics, Professor of Mathematics
Peter D. Johnson, Professor, Professor of Mathematics
Douglas A. Leonard, Professor, Professor of Mathematics

Abstract

Sudoku has risen in popularity over the past few years. The rules are simple, yet the solutions are often less than trivial. Mathematically, these puzzles are interesting in their own right. This dissertation will use the idea of a Sudoku Puzzle to define a new kind of $n \times n$ array. Further, we will aim to prove some necessary (and on occasion sufficient) conditions for the existence of these arrays. To that end, we define a latin square of order n as an $n \times n$ array where every row and every column contain every symbol $1, 2, \ldots, n$ exactly once. We say $a \times b$ is an ordered factor pair of the integer n if $n = a \times b$. An (a, b)-Sudoku latin square is a latin square where in addition to each row and column containing every symbol exactly once, each $a \times b$ rectangle also contains every symbol exactly once when the $n \times n$ array is tiled with $a \times b$ rectangles in the natural way. A factor pair latin square of order n (denoted FPLS(n)) is an (a, b)-Sudoku latin square for every factor pair (a, b) of n. This dissertation will mainly be concerned with the existence of such designs as well as related problems to such designs.

Acknowledgments

I would like to thank the entire Mathematics Department at Auburn University. In particular, I would like to express my deepest gratitude to my advisory committee, without whom this would not be possible. Dr. Chris Rodger, Dr. Douglas Leonard, and Dr. Pete Johnson have always been there for me whenever I needed an ear to talk through an idea. Each one of you has taught me a great lesson about what it means to be a professor, a mathematician, and a mentor. Thank you to Dr. Dean G. Hoffman for seeing me through this problem and through the entirety of my graduate career at Auburn University. You have always believed in me and given me room to explore wherever this problem led me, encouraging me the entire way. This problem was Dr. Hoffman's brain child. Thank you for your willingness to share it with me.

I would also like to thank my advisors from my undergraduate university – Kutztown University of Pennsylvania. Thank you Dr. Padraig McLoughlin, who believed in me and ensured that I got into a good graduate school. To Dr. Anke Walz, Words cannot describe how much your constant guidance and friendship has meant to me through the years. You have been my mentor and role model. I have never stopped learning from you, and I suspect that I never will.

To my colleagues at Auburn. It sounds cliché to all of you, but my Auburn Family has been my family away from home. I tried to enumerate the people that I would have to thank in this paragraph; however, the list is just too extensive. It will have to suffice to say that you all know who you are. I couldn't have made it through without you all. We've had a lot of fun, laughed, cried, and worked together. You've been with me through everything, and I think you all for that. I will never forget these times. I hope to stay in touch with each and every one of you.

To my friends in from Pennsylvania. You're more like family to me than friends. Like my family, you have been there with me through the good times and the bad times. I would do anything for you, and I know you would do anything for me. In particular, thank you Brian Levering, Nate Ohlinger, Mark Shivers, Mike Ricco, James Corson, and Jim Bellizzi. I have looked up to all of you all for different reasons throughout my life, and I continue to do so.

To my Family. In particular, I would like to thank Gina Hammer, Jim Hammer, Mary Assetto, Jack Assetto, Danny Concordia, and Darlene Concordia. What can I say? Each and every one of you have been there with me through thick and thin. You have always stood by me, no matter what. You have gotten me through some hard times. Each one of you have been there when I really needed you. You believed in me when I didn't believe in myself. What more can I say but I love you all dearly. I wouldn't be the person I am today if it weren't for all of you believing in me and supporting me through the years. From the bottom of my heart, thank you!

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 History

In recent years, Sudoku puzzles have become extremely popular. The modern day Sudoku puzzle first appeared in 1979 as a puzzle called "Number Place" in Dell Magazine. [5] They were designed by Howard Garns [1], a freelance puzzle constructor and retired architect. A *Sudoku puzzle* is a 9×9 square grids in which every cell contains exactly one *symbol*, typically denoted with integers 1 through 9, in such a way that each 1×9 row contains each symbol exactly once, each 9×1 column contains each symbol exactly once, and each 3×3 sub-square (often called *blocks* or *regions*) contains each symbol exactly once. For ease of speech, we say a subset S of n cells of an $n \times n$ grid is *latin* if S contains each symbol exactly once. The property that every row and every column is latin is an important property. Arrays in which every row and every column is latin are called *latin squares*.

So, how would one go about extending the idea of what a Sudoku puzzle is for orders larger than nine? To define a higher order Sudoku puzzle, one must determine what size the blocks will be. A positive integer n is said to have *ordered factor pairs* (a, b) if $n = a \times b$. Naturally, if (a, b) is an ordered factor pair of a positive integer then (b, a) is also an ordered factor pair. The distinction is important in this case, because when we are defining regions over the $n \times n$ grid, the $a \times b$ regions may be different from the $b \times a$ regions. More specifically, if $a > b$ then the $a \times b$ regions have more rows than columns while the $b \times a$ regions have more columns than rows. One might observe that the $a \times b$ regions look like the $b \times a$ regions if one were to transpose the $n \times n$ array. So, given an ordered factor pair (a, b) of a positive integer n, the $n \times n$ grid can then be partitioned into $a \times b$ regions in a very natural way, namely starting in the top left corner. An (a, b)-*Sudoku latin square* of order n is a latin

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

.

Figure 1.1: FPLS(6)

square on the symbol set $\{1, 2, \ldots n\}$ where each $a \times b$ region in the natural tiling contains all of the symbols exactly once [6].

As this dissertation is mainly concerned with existence of different kinds of designs, it is interesting to see when an (a, b)-Sudoku latin square exists. One might observe that there exists an (a, b)-Sudoku latin square for each ordered factor pair (a, b) of a positive integer n, as will be shown in Chapter 2 by giving an explicit construction. We will then take this definition a step further and say a *factor pair latin square* of order n, denoted FPLS(n), is an (a, b)-Sudoku latin square of order n for every ordered factor pair (a, b) of n.

Certainly as the number of ordered factor pairs increase, the problem gets more complex. In order to solidify this idea, perhaps an example is in order. First, observe that the number six has ordered factor pairs $1 \times 6, 6 \times 1, 2 \times 3$, and 3×2. Therefore, a FPLS(6) would have the following regions being Latin: $1 \times 6, 6 \times 1, 3 \times 2$, and 2×3. An example of a FPLS(6) is presented in Figure 1.1.

Of course, the first question that arises with such a definition is to whether these designs exist or not. Originally, it was conjectured that there existed a factor pair latin square for every order n; however, as we will see in Chapter 4, this conjecture is false since there exist several infinite families of positive integers n such that there does not exist a factor pair latin square of order n. Therefore, finding necessary and sufficient conditions for the existence of such designs is non-trivial.

There are other natural questions to ask as we are striving to generalize the definition of a Sudoku puzzle. One such way is to change the shape of the regions. These designs are called *Gerechte designs* [3]. Gerechte designs have been useful in designing agricultural experiments. Moreover, an $n \times n$ square that satisfies multiple Gerechte designs at the same time is called a *multiple Gerechte design* [1]. This dissertation will focus on finding some necessary (and sometimes sufficient) conditions for the existence of factor pair latin squares, which are special types of multiple Gerechte designs.

1.2 Plan of Attack

This dissertation will mainly be concerned with finding some necessary and sufficient conditions for the existence of factor pair latin squares. That is to say, for what orders can it be guaranteed that a factor pair latin squares exists and for what orders can it be guaranteed that a factor pair latin squares does not exist? To that end, Chapter 2 will look at a few small examples as well as some natural constructions of factor pair latin squres. Afterwards, Chapter 4 will produce some infinite families of factor pair latin squares that do not exist.

One of the most basic factor pair latin square that can be constructed is one of a prime order; since if p is prime, the only ordered factor pairs of p are $1 \times p$ and $p \times 1$. This simply means that every row and every column in the $p \times p$ array is latin. Since latin squares exist for every order n (as stated in [7]), if one is looking for a factor pair latin square of prime order p, one can pick any latin square of order p as a representative.

Chapter 2 will be concerned with some general constructions of factor pair latin squares. Section 2.2 will give a construction for when n is the power of a prime number and Section 2.1 will construct a factor pair latin square of order $2p$, where p is a prime number.

It is sometimes helpful to word a problem in different ways. To that avail, a *hypergraph* is a pair $H = (V, E)$ where V is a set of vertices and E is a set of subsets of V called *edges* or *hyperedges*. Moreover, a *k-uniform hypergraph* is a hypergraph in which every edge is of size k (that is every edge is incident to k vertices). Constructing a partial factor pair latin

square of order n is equivalent to properly rainbow n-coloring the vertices (meaning that the colors on the vertices of each edge are all different) of an n-uniform hypergraph with nN edges, where N is the number of ordered factor pairs. Chapter 2 will give necessary and sufficient conditions for a rainbow n-coloring when there are less than or equal to four edges in the n-uniform hypergraph that intersect. When more than four edges can intersect, the conditions are shown to be necessary and not sufficient.

Further, one can construct examples of factor pair latin squares up to and including order eleven with constructions given in Chapter 2. Order twelve, however, is an interesting case. It will be discussed in Section 4.1 as to why there does not exist a factor pair latin square of order twelve. The remainder of Chapter 4 will give other infinite families of orders for which factor pair latin squares do not exist.

Section 6.1 will map the completion of a partial factor pair latin square to the exact cover problem. In doing this, Donald Knuth's DLX algorithm can be used as discussed in [9]. Programs to brute force a partially filled factor pair latin square have been put in Appendices C and D. Chapter 6 will address the problem of completing a factor pair latin square in terms of the pigeon-hole principle (Section 6.4).

When there does not exist a factor pair latin square, Chapter 5 will discuss when a maximum set of ordered factor pairs F exists such that for every ordered factor pair (a, b) from F there is an (a, b)-Sudoku latin square. The aforementioned designs will be called *quasi-factor pair latin squares.*

Lastly, Chapter 7 will discuss open problems and future research.

Chapter 2

Known Constructions of Factor Pair Latin Squares

Provided $n = ab$, and n, a, and b are positive integers, an (a, b)-*Sudoku latin square* of order n is a latin square on the symbol set $\{1, 2, \ldots n\}$ where each $a \times b$ region in the natural tiling contains all of the symbols exactly once [6]. A *factor pair latin square* is an (a, b)-Sudoku latin square for every ordered factor pair of order n. Since we are using (a, b)-Sudoku latin squares to define a factor pair latin square, it is good to start out with the existence of (a, b)-Sudoku latin squares.

Theorem 2.1. *There exists an (a, b)-Sudoku latin square for each ordered factor pair (a, b) of order n.*

Proof. This proof will be constructive. That is to say that it will give an algorithm for creating an (a, b)-Sudoku latin square followed by a proof that the algorithm works.

Algorithm. Construct an $ab \times ab$ array as in Figure 2.1 as follows:

1. Fill in the top left $a \times b$ so that each symbol appears exactly once. Call this $a \times b$ matrix A.

2. Let the $a \times a$ matrix
$$P = \left[\begin{array}{c|c} 0 & 1 \\ \hline I_{a-1,a-1} & 0 \end{array}\right]$$
define the permutation matrix of order a that cyclically shifts the rows by one and the $b \times b$ matrix
$$Q = \left[\begin{array}{c|c} 0 & I_{b-1,b-1} \\ \hline 1 & 0 \end{array}\right]$$
denote the permutation matrix of order b that cyclically shifts the columns by one.

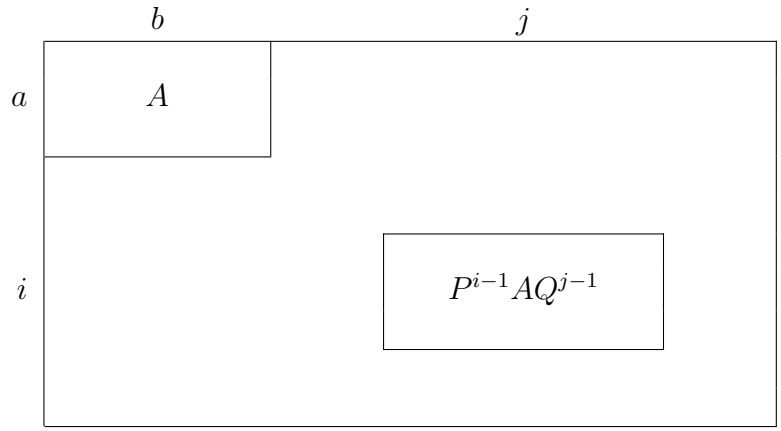

Figure 2.1: (a, b)-Sudoku Latin Square Construction

3. For the $1 \leq i \leq b$ and $1 \leq j \leq a$, to compute the rectangle in the i^{th} row (with respect to the rectangles) and j^{th} column (with respect to the rectangles), multiply $P^{i-1}AQ^{j-1}$.

Justification. Since every symbol occurs exactly once in the top left $a \times b$ rectangle, the top left rectangle is clearly latin. For $1 \leq i \leq b$, since P^i is a permutation matrix of order a which rotates the rows cyclically down by i, each of the first a rows in the $n \times n$ array are latin. Also, since the original top left $a \times b$ rectangle was latin to begin with, rectangles in the first row of $a \times b$ rectangles are also latin. In a similar fashion, for $1 \leq j \leq a$, since Q^j is a permutation matrix of order b which rotates the columns cyclically right by j, each of the columns in the $n \times n$ array is latin. Similarly, the remaining rows of the $n \times n$ array are latin since the first a rows were latin. Once again, since the original top left $a \times b$ rectangle was latin to begin with, each of the subsequent rectangles obtained by multiplying by these permutation matrices will also be latin, since the permutations are only effecting how the symbols are arranged within each $a \times b$ rectangle. □

However, there is not a similar theorem for factor pair latin squares. As we will see in Chapter 4, there does not exist a factor pair latin square for every order n. This section will be mostly be concerned with constructing infinite families of factor pair latin squares.

2.1 Power of Primes

One might observe that there exists a factor pair latin square of every prime order. A number p is said to be *prime* if it is greater than one and has no positive factors other than one and itself. Given a prime number p, the only two ordered factor pairs that need to be latin are the $1 \times p$ rows and the $p \times 1$ columns. By the definition of a latin square, every latin square of order p is a factor pair latin square of order p.

On a less trivial note, it is natural to look at the prime factorization of a number n, since factor pair latin squares are designs that are concerned with all of the ordered factors of the number n. Before we get to the following theorem, we first need to define what a word is. A *word* is an ordered sequence of symbols often called *letters* from a particular set which is often called the *alphabet*. For example, $\mathbb{Z}_p^\alpha$ will have words of length α with respect to the alphabet $\mathbb{Z}_p$. That is to say that every symbol must come from $\mathbb{Z}_p$ (for example, the word 1101 would be a word in $\mathbb{Z}_2^4$). We will also need to know what a quasigroup is. A quasigroup is a latin square with a headline, a sideline, and a binary operation defined. The following theorem ensures that a factor pair latin square can be constructed if n is a power of a prime number.

Theorem 2.2. *Let p be a prime number and let α be a positive integer. Then there exists a factor pair latin square of order p^α.*

Proof. This proof will be constructive. That is to say that it will give an algorithm for creating a factor pair latin square of order $n = p^\alpha$, followed by a proof that the algorithm does what is intended.

Algorithm. Let $\mathbb{Z}_p^\alpha$ denote the words of length α from the alphabet $\mathbb{Z}_p$. Consider the following quasigroup with entries from $\mathbb{Z}_p^\alpha$. Label the headline with $v_1, v_2, \ldots, v_n$ such that $\bigcup_{i=1}^n v_i = \mathbb{Z}_p^\alpha$ and the v_i's are ordered lexicographically. Label the sideline with $u_1, u_2, \ldots, u_n$ where u_i is obtained from v_i by applying the permutation $(\alpha, \alpha - 1, \ldots, 2, 1)$. Let the entry of cell (a, b) be $a + b \pmod p$.

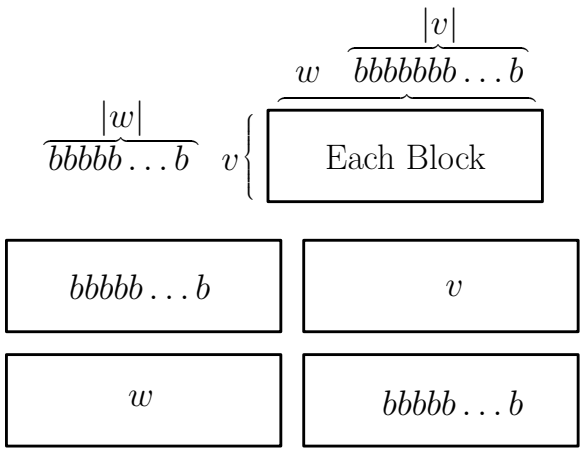

Figure 2.2: Power of Primes

Justification. Define a block of the array to be the projection of the block we are interested in onto the headline and sideline. Let $\beta \in \{0, 1, \ldots, \alpha\}$ and note that each block in the headline consists of a word w of length β concatenated with every word of length $\alpha - \beta$. The sideline has every word of length β concatenated with some fixed word v of length $\alpha - \beta$. So, within each block, v and w are set words that run through every combination of words of size $|v|$ and $|w|$ respectively. Moreover, this means that every rectangle of size $p^{\alpha-\beta} \times p^{\beta}$ must be latin. This can be exemplified by the following example as well as Figure 2.2. □

Perhaps an example is in order. A factor pair latin square of order eight can be constructed using the method outlined in Theorem 2.2, since eight has a prime factor decomposition of 2^3. What follows is the construction from Theorem 2.2 for a factor pair latin square of order eight. If one examines the boxed in cells in Figure 2.3, the w in the above proof represents the word 1. Notice that all of the words of length two are concatenated to w. Similarly, v is the word 01. Notice that v is concatenated to all possible words of length one, producing the latin 2×4 block in boxed in cells.

For constructing a FPLS(8), we construct $(\mathbb{Z}_2^3)$ as in Figure 2.3.

+	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
100	100	101	110	111	000	001	010	011
010	010	011	000	001	110	111	100	101
110	110	111	100	101	010	011	000	001
001	001	000	011	010	101	100	111	110
101	101	100	111	110	001	000	011	010
011	011	010	001	000	111	110	101	100
111	111	110	101	100	011	010	001	000

Figure 2.3: FPLS(8)

2.2 Twice a Prime

One of the goals is to get constructions for as many factor pair latin squares as we can. To that avail, if n can be decomposed into twice a prime number, then a factor pair latin square of order n can be constructed. That is there exists a factor pair latin square of order $n = 2p$, where p is a prime number.

Theorem 2.3. *Let p be a prime number. Then there exists a factor pair latin square of order $2p$.*

Proof. This proof will be constructive. That is to say that when p is a prime number an algorithm is presented for creating a factor pair latin square of order $2p$. This will be followed by a proof that the algorithm does what is intended.

Algorithm. Since $n = 2p$, we will be looking at elements from $\mathbb{Z}_2 \times \mathbb{Z}_p$. That is to say that every element in the $2p \times 2p$ array will be an ordered pair (a, b) where a is an element of $\mathbb{Z}_2$ and b is an element of $\mathbb{Z}_p$. For ease of construction, we will construct two different $2p \times 2p$ arrays. The first array, $\mathcal{Z}_1$, will be with symbols from $\mathbb{Z}_2$, while the second array, $\mathcal{Z}_p$, will correspond to symbols from $\mathbb{Z}_p$. Let x_{ij} denote the symbol in cell (i, j) of $\mathcal{Z}_2$ and y_{ij} denote the symbol in cell (i, j) of $\mathcal{Z}_p$. Cell (i, j) of the constructed factor pair latin square of order $2p$ will be filled with the ordered pair (x_{ij}, y_{ij}). Construct $\mathcal{Z}_2$ and $\mathcal{Z}_p$ as follows:

To construct $\mathcal{Z}_2$:

- In the first row, fill the first p cells up with the symbol 0 and cells $p+1$ through $2p$ with the symbol 1.
- In the second row, fill the first p cells up with the symbol 1 and cells $p+1$ through $2p$ with the symbol 0.
- For rows 3 through $2p$, use the addition table for $\mathbb{Z}_2$ with the headline of alternating 0's and 1's and the sideline of alternating 0's and 1's.

To construct $\mathcal{Z}_p$:

- For the first two rows, use the headline 0 through $p-1$ twice and the sideline 0.
- For the third row through the $(p-1)^{\text{st}}$ row, group the rows in two. The sideline for the third and fourth row should be 2, the fifth and sixth should be 4, and so on to the $(p-1)^{\text{st}}$ row.
- For rows $(p+2)$ through $(2p-2)$, group the rows in two. The sideline for the $(p+2)^{\text{th}}$ and $(p+3)^{\text{th}}$ row should be 1, the $(p+4)^{\text{th}}$ and the $(p+5)^{\text{th}}$ rows should be 3, and so on.
- For the p^{th} row and the $(2p)^{\text{th}}$ row, use the headline of $0, 0, 2, 2, 4, 4, \ldots, p, p$. The sideline should be $p-1$.
- For the $(p+1)^{\text{st}}$ row and the $(2p-1)^{\text{st}}$ row, use the headline $p-1$, 1, 1, 3, 3, $\ldots$, $p-1$ with the sideline $p-1$.

Justification. Notice that in $\mathcal{Z}_p$, each $2 \times p$ block and each $p \times 2$ block has symbols 1 through $2p$ exactly twice. This fact is obvious for the first two rows, and is true for every subsequent pair of two rows from the third row through the $(p-1)^{\text{st}}$ row as well as the $(p+2)^{\text{nd}}$ row through the $(2p-2)^{\text{nd}}$ row since we are simply repeating the symbols (that is to say the third row is the same as the fourth row and so on). Similarly, the p^{th} row and the $(p+1)^{\text{st}}$ row as well as the $(2p)^{\text{th}}$ row and the $(2p-1)^{\text{st}}$ row have each symbol exactly twice since for

0	0	0	0	0	0	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0

Figure 2.4: $\mathcal{Z}_2$

every symbol, it's partner is next to it accept for the p^{th} symbol, which is the first symbol of the next row. Moreover, each repeated symbol is next to it's partner (it's partner is either to the left, right, above or below it) accept for the $(p+1)^{\text{st}}$ row and the $(2p-1)^{\text{st}}$ row, which have exactly one symbol that wraps around the grid. In any case, the $\mathcal{Z}_2$ grid alternates between symbols 0 and 1, making every ordered pair occur exactly once within each row, column, and block of the grid. □

As an illustration of this, a factor pair latin square of order fourteen will be constructed using the above algorithm. By the above algorithm, we have $\mathcal{Z}_2$ as in Figure 2.4 and $\mathcal{Z}_7$ as in Figure 2.6 and Figure 2.5. Putting both $\mathcal{Z}_2$ and $\mathcal{Z}_7$ together, a completed factor pair latin square of order fourteen will be produced in Figure 2.7

0	1	2	3	4	5	6	0	1	2	3	4	5	6
0	1	2	3	4	5	6	0	1	2	3	4	5	6
2	3	4	5	6	0	1	2	3	4	5	6	0	1
2	3	4	5	6	0	1	2	3	4	5	6	0	1
4	5	6	0	1	2	3	4	5	6	0	1	2	3
4	5	6	0	1	2	3	4	5	6	0	1	2	3
6	6	1	1	3	3	5	5	0	0	2	2	4	4
5	0	0	2	2	4	4	6	6	1	1	3	3	5
1	2	3	4	5	6	0	1	2	3	4	5	6	0
1	2	3	4	5	6	0	1	2	3	4	5	6	0
3	4	5	6	0	1	2	3	4	5	6	0	1	2
3	4	5	6	0	1	2	3	4	5	6	0	1	2
5	0	0	2	2	4	4	6	6	1	1	3	3	5
6	6	1	1	3	3	5	5	0	0	2	2	4	4

.

Figure 2.5: $\mathcal{Z}_7$

	0	1	2	3	4	5	6	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6	0	1	2	3	4	5	6
2	2	3	4	5	6	0	1	2	3	4	5	6	0	1
2	2	3	4	5	6	0	1	2	3	4	5	6	0	1
4	4	5	6	0	1	2	3	4	5	6	0	1	2	3
4	4	5	6	0	1	2	3	4	5	6	0	1	2	3

	0	0	2	2	4	4	6	6	1	1	3	3	5	5
6	6	6	1	1	3	3	5	5	0	0	2	2	4	4

	6	1	1	3	3	5	5	0	0	2	2	4	4	6
6	5	0	0	2	2	4	4	6	6	1	1	3	3	5

	0	1	2	3	4	5	6	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0	1	2	3	4	5	6	0
1	1	2	3	4	5	6	0	1	2	3	4	5	6	0
3	3	4	5	6	0	1	2	3	4	5	6	0	1	2
3	3	4	5	6	0	1	2	3	4	5	6	0	1	2

	6	1	1	3	3	5	5	0	0	2	2	4	4	6
6	5	0	0	2	2	4	4	6	6	1	1	3	3	5

	0	0	2	2	4	4	6	6	1	1	3	3	5	5
6	6	6	1	1	3	3	5	5	0	0	2	2	4	4

Figure 2.6: $\mathcal{Z}_7$ Construction

0,0	0,1	0,2	0,3	0,4	0,5	0,6	1,0	1,1	1,2	1,3	1,4	1,5	1,6
1,0	1,1	1,2	1,3	1,4	1,5	1,6	0,0	0,1	0,2	0,3	0,4	0,5	0,6
0,2	1,3	0,4	1,5	0,6	1,0	0,1	1,2	0,3	1,4	0,5	1,6	0,0	1,1
1,2	0,3	1,4	0,5	1,6	0,0	1,1	0,2	1,3	0,4	1,5	0,6	1,0	0,1
0,4	1,5	0,6	1,0	0,1	1,2	0,3	1,4	0,5	1,6	0,0	1,1	0,2	1,3
1,4	0,5	1,6	0,0	1,1	0,2	1,3	0,4	1,5	0,6	1,0	0,1	1,2	0,3
0,6	1,6	0,1	1,1	0,3	1,3	0,5	1,5	0,0	1,0	0,2	1,2	0,4	1,4
1,5	0,0	1,0	0,2	1,2	0,4	1,4	0,6	1,6	0,1	1,1	0,3	1,3	0,5
0,1	1,2	0,3	1,4	0,5	1,6	0,0	1,1	0,2	1,3	0,4	1,5	0,6	1,0
1,1	0,2	1,3	0,4	1,5	0,6	1,0	0,1	1,2	0,3	1,4	0,5	1,6	0,0
0,3	1,4	0,5	1,6	0,0	1,1	0,2	1,3	0,4	1,5	0,6	1,0	0,1	1,2
1,3	0,4	1,5	0,6	1,0	0,1	1,2	0,3	1,4	0,5	1,6	0,0	1,1	0,2
0,5	1,0	0,0	1,2	0,2	1,4	0,4	1,6	0,6	1,1	0,1	1,3	0,3	1,5
1,6	0,6	1,1	0,1	1,3	0,3	1,5	0,5	1,0	0,0	1,2	0,2	1,4	0,4

Figure 2.7: FPLS(14)

Chapter 3

Wording as a k-Uniform Hypergraph

In order to properly reword this problem in terms of Graph Theory, some preliminary definitions must be presented. A *hypergraph* is a pair $H = (V, E)$ where V is a set of *vertices* and E is a set of subsets of V called *edges* or *hyperedges*.

Let V_n be the n^2 vertices corresponding to the cells of an $n \times n$ array. Let J be the set of indices of the form $(a, b, j), 1 \leq j \leq n$, where (a, b) is an ordered factor pair of n. So, $A(a, b, j)$ is the j^{th} sub-rectangle of the $n \times n$ array of cells in the natural partition ordering them in "typewriter" order. For all $j \in J$, let $A_j \subseteq V_n$ be the set of n cells of the $n \times n$ array corresponding to the n cells of the j^{th} rectangle of the factor pair latin square. So, for each ordered factor pair, the set of nonempty rectangles A_i associated with that ordered factor pair partitions V_n. Let $E_n = \{A_j \mid 1 \leq j \leq fn\}$ and define the hypergraph $H_n = (V_n, E_n)$. Further, let $\mathcal{A}_I = \bigcap_{i \in I} A_i \setminus \bigcup_{j \in I^C} A_j$. So , for all $i \in I$, the set of all nonempty $\mathcal{A}_i$ also partitions V_n.

A *k-uniform hypergraph* is a hypergraph where every edge is of size k. It should be noted that each edge of the hypergraph generated by a factor pair latin square will have n vertices in it, making the hypergraph formed from the factor pair latin square of order n an n-uniform hypergraph. A coloring C of the hypergraph H is said to be a *proper rainbow coloring* of H if no edge of H contains two vertices of the same color. If a hypergraph has a proper rainbow coloring in c colors, we say that the hypergraph is *c-rainbow colorable*. When is a k-uniform hypergraph k-rainbow colorable? In general, this question is NP-complete; however, for k-uniform hypergraphs with four edges or less, necessary and sufficient conditions can be found.

3.1 Three Edges

The following theorem demonstrates when a k-uniform hypergraph with three edges is k-rainbow colorable.

Theorem 3.1. *Let H be an n-uniform hypergraph with $N = 3$ edges. Then, H is n-rainbow colorable if and only if*

$$|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| \leq n.$$

Proof. Clearly, if H is a n-rainbow colorable then $|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| \leq n$, since any two vertices are in at least one edge together. That is to say they must all be different colors. Since there are only n colors to choose from, the above summation must be less than n.

Since H is an n-uniform hypergraph, the following conditions are always true:

$$\begin{aligned} |\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{1,2,3}| &\leq n \\ |\mathcal{A}_{1,2}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| &\leq n \\ |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| &\leq n. \end{aligned}$$

If $|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| \leq n$, then there are five different color classes as depicted in Figure 3.1, Where v, w, x, y, and z denote the number of color classes that have the configuration depicted above them. Summing them all up gives us n, which is also the number of colors. Moreover, this will give us a proper n-rainbow coloring of H. □

Naturally, if one can find a $|\mathcal{A}_I|$ which has greater than n cells in it where every two cells are in at least one edge together, there cannot exist a factor pair latin square of order n, since there are not enough symbols to fill all of the cells of $|\mathcal{A}_I|$. The following corollary says that these conditions are necessary for three rectangles which are allowed to overlap in a factor pair latin square of order n.

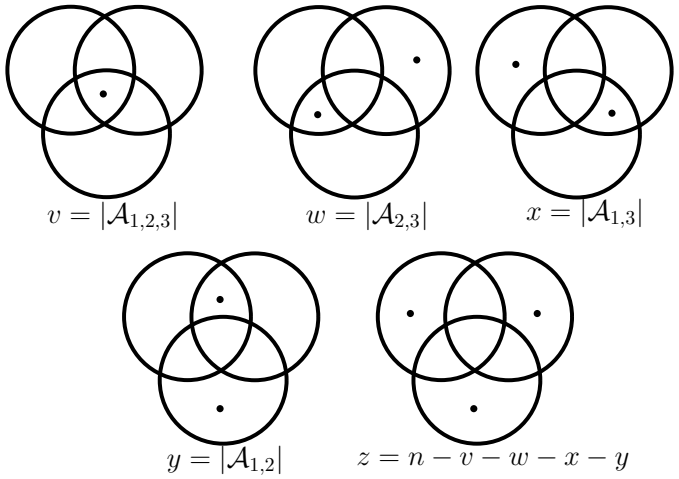

Figure 3.1: n-Coloring n-Uniform Hyper-Graph with Three Edges

Corollary 3.2. *There does not exist a factor pair latin square of order n if there exist $N = 3$ rectangles A_1, A_2, and A_3 such that*

$$|\mathcal{A}_{1,3}| + |\mathcal{A}_{1,2}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| > n.$$

Proof. This follows from the contrapositive of Theorem 3.1. If the above inequality holds, then the n-uniform hypergraph is not n-rainbow colorable. Moreover, this means that there does not exist a factor pair latin square. □

The above fact will be used in an alternate proof of Theorem 4.2.

3.2 Four Edges

Since all that was needed from Theorem 3.1 was sufficiency when discussing factor pair latin squares, as we increase the number of edges in the n-uniform hypergraph, sufficiency will suffice. Nevertheless, for an n-uniform hypergraph with four edges, inequalities can be found that are both necessary and sufficient.

Theorem 3.3. *Let H be an n-uniform hypergraph with $N = 4$ edges. Then, H is n-rainbow colorable if and only if*

$$\begin{aligned}
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{23}| + |\mathcal{A}_{24}| + |\mathcal{A}_{34}| &\leq n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{14}| + |\mathcal{A}_{24}| + |\mathcal{A}_{34}| &\leq n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{23}| + |\mathcal{A}_{24}| &\leq n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{14}| + |\mathcal{A}_{24}| &\leq n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{13}| + |\mathcal{A}_{23}| + |\mathcal{A}_{34}| &\leq n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{13}| + |\mathcal{A}_{14}| + |\mathcal{A}_{34}| &\leq n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{13}| + |\mathcal{A}_{23}| &\leq n, \text{ and}\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{13}| + |\mathcal{A}_{14}| &\leq n.
\end{aligned}$$

Proof. A similar proof to Theorem 3.1 could be used; however, an alternative proof will be given here using set theory and linear algebra. A *set partition* of a set U (often called a *universal set*) is a non-empty set of non-empty subsets of U such that every element u of U is in exactly one subset. In other words, if P is a partition of a set $\mathscr{N} = \{1, 2, 3, 4\}$, then *a)* $\varnothing \notin P$, *b)* $\bigcup_{A \in P} A = \mathscr{N}$, and *c)* If $A, B \in P$ and $A \neq B$, then $A \cap B = \varnothing$. Let $\mathscr{E}$ denote the set of all set partitions of $\mathscr{N}$. Construct a $(2^4 - 1) \times |\mathscr{E}|$ matrix whose headline is $\mathscr{E}$ and sideline is $2^{\mathscr{N}} \setminus \varnothing$, filling cell (i, j) with a 1 if and only if the set in the i^{th} sideline is contained in the set partition in the j^{th} column and a 0 otherwise as done in Figure 3.2. A $\left(2^N - 1\right) \times \left(2^N - 1\right)$ identity matrix has been appended to the matrix in order to keep track of the elementary row operations. Using elementary row operations to row reduce the matrix, we find that there are $\left(2^N - 1\right)$ inequalities along with three free variables as depicted in Figure 3.3.

From the construction of this matrix, it was necessary that each row had to be greater than or equal to zero (that is, no subset of a partition can be negative). Let $i \in \mathscr{N}$ and let K_i be the set of subsets of $2^{\mathscr{N}}$ that contain i except for the singleton set $\{i\}$. With this

notation, it is clear that $|\mathcal{A}_i| = n - \sum_{j \in K_i} |\mathcal{A}_j|$. Using this substitution along with Fourier-Motzkin elimination [4] gives us the above inequalities, showing that these conditions are necessary.

Sufficiency holds as well. To show this, split the matrix into two pieces. On the side with the set partitions, we have an identity matrix. On the other side, we have the elementary row operations done in order to row reduce the set partition side. Reading this matrix from left to right gives us that the set partition indicated on the left is equal to the sum of the row operations depicted on the right. Moreover, if the above inequalities hold, these give us our color classes as in the proof of sufficiency in Theorem 3.1. □

As before, if one can find an $|\mathcal{A}_I|$ which has greater than n cells in it such that every two cells are in at least one edge together then there cannot exist a factor pair latin square of order n, since there are not enough symbols to fill all of the cells of $|\mathcal{A}_I|$. The following corollary says that for four rectangles which are allowed to overlap, these conditions are necessary.

Corollary 3.4. *There does not exist a factor pair latin square of order n if there exist $N = 4$ rectangles $A_1, A_2, A_3,$ and A_4 such that*

$$\begin{aligned}
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{23}| + |\mathcal{A}_{24}| + |\mathcal{A}_{34}| &> n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{14}| + |\mathcal{A}_{24}| + |\mathcal{A}_{34}| &> n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{23}| + |\mathcal{A}_{24}| &> n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{14}| + |\mathcal{A}_{24}| &> n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{13}| + |\mathcal{A}_{23}| + |\mathcal{A}_{34}| &> n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{13}| + |\mathcal{A}_{14}| + |\mathcal{A}_{34}| &> n,\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{13}| + |\mathcal{A}_{23}| &> n, \text{ or}\\
|\mathcal{A}_{1234}| + |\mathcal{A}_{123}| + |\mathcal{A}_{124}| + |\mathcal{A}_{134}| + |\mathcal{A}_{234}| + |\mathcal{A}_{12}| + |\mathcal{A}_{13}| + |\mathcal{A}_{14}| &> n.
\end{aligned}$$

	1234	123|4	124|3	134|2	234|1	12|34	13|24	14|23	12|3|4	13|2|4	14|2|3	23|1|4	24|1|3	34|1|2	1|2|3|4															
1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
24	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
34	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
123	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
124	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
134	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
234	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1234	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 3.2: Four Rectangles

												α	β	γ	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_{12}$	$\mathcal{A}_{13}$	$\mathcal{A}_{14}$	$\mathcal{A}_{23}$	$\mathcal{A}_{24}$	$\mathcal{A}_{34}$	$\mathcal{A}_{123}$	$\mathcal{A}_{124}$	$\mathcal{A}_{134}$	$\mathcal{A}_{234}$	$\mathcal{A}_{1234}$
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	-1	-1	-1	0	0	0	-1	1	1	0	1	-1	-1	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	-1	-1	1	-1	1	1	-1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	-1	-1	0	0	1	1	-1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	-1	1	1	0	0	-1	-1	1	0	0	-1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	1	0	-1	1	0	-1	1	0	-1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	1	-1	1	-1	0	1	-1	0	0	0

Figure 3.3: Row Reduced Four Rectangles

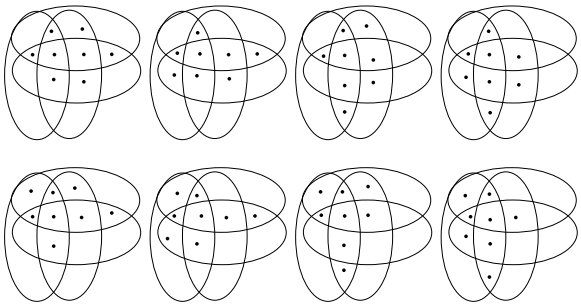

Figure 3.4: Pictoral Version of Theorem 3.3

Proof. This follows from the contrapositive of Theorem 3.3. If the above inequality holds, then the n-uniform hypergraph is not n-rainbow colorable. Moreover, this means that there does not exist a factor pair latin square of order n. □

3.3 Greater than Four Edges

This observation shows that the necessary conditions for an n-rainbow coloring of an n-uniform hypergraph is all that is needed when concerned with factor pair latin squares. Moreover, if we were to depict the inequalities for Theorem 3.3 as done in Figure 3.4, it becomes evident that all that is needed is a maximal set with the property that any two intersections are in at least one edge together.

The above observation leads to the following theorem:

Theorem 3.5. *Let N denote the number of edges in an n-uniform hypergraph H, $\mathscr{N} = \{1, 2, 3, \ldots, N\}$, and $\mathscr{B}$ be a maximal subset of $2^{\mathscr{N}}$ with the property that for α and $\beta \in \mathscr{B}, A_\alpha \cap A_\beta \neq \varnothing$. If a hypergraph with N edges is n-rainbow colorable, then*

$$\sum_{B \in \mathscr{B}} |\mathcal{A}_B| \leq n.$$

This is also sufficient if $n \leq 4$ as shown in Theorem 3.1 and Theorem 3.3; however, it is not sufficient for $n \geq 5$.

Proof. The set $\mathscr{B}$ having the property that every two sets, B_1 and B_2, intersect means that the vertices within $\bigcap_{J\subseteq B_1} A_J$ and the vertices within $\bigcap_{J\subseteq B_2} A_J$ must be in at least one edge together. In other words, let $B_1 \cap B_2 = I$. Then the vertices within $\bigcap_{J\in B_1} A_J$ and the vertices within $\bigcap_{J\in B_2} A_J$ must be in rectangle A_i together where $i \in I$. Hence, each of the vertices in $\bigcup_{B\in\mathscr{B}} \mathcal{A}_B$ must be different colors. Therefore, there must be less than n vertices in $\sum_{B\in\mathscr{B}} |\mathcal{A}_B|$.

These conditions are not sufficient, however, since C_5 is a 2-uniform hypergraph which requires three colors. Since the intersection of any two edges is at most one vertex, the necessary conditions hold, but C_5 cannot be three colored. □

Corollary 3.6. *Let N denote the number of rectangles in a factor pair latin square of order n, $\mathscr{N} = \{1, 2, 3, \ldots, N\}$, and $\mathscr{B}$ be a maximal subset of $2^{\mathscr{N}}$ with the property that for α and $\beta \in \mathscr{B}$, $A_\alpha \cap A_\beta \neq \varnothing$. There does not exists a factor pair latin square of order n if there exist N rectangles such that*

$$\sum_{B\in\mathscr{B}} |\mathcal{A}_B| > n.$$

Proof. This is a direct consequence (the contrapositive) of Theorem 3.5. □

Chapter 4

Negative Results on Factor Pair Latin Squares

4.1 Generalizing Order Twelve

At the onset of this problem, it was conjectured that a factor pair latin square existed for every order. With the proper time and patience, one can easily show that there does not exist a factor pair latin square of order twelve, since the ordered factor pairs of twelve are $\{1 \times 12, 12 \times 1, 2 \times 6, 6 \times 2, 3 \times 4, 4 \times 3\}$.

Theorem 4.1. *There does not exist a factor pair latin square of order twelve.*

Proof. Since the first 2×6 rectangle must be latin, fill the top left most 2×6 rectangle with the twelve symbols. Since $6 = 3 \cdot 2$, there are 2 rectangles of size 3×4 that cover the top 2×6 block. There is also a 4×3 block that covers the top of the first 2×6 block. Moreover, cell $(3, 4)$ cannot be filled, since it cannot be any symbol from the first $\frac{4}{6}$ of the top left 2×6 rectangle and it cannot be any symbol from the second $\frac{3}{6}$ of the top left 2×6 rectangle. □

This can be thought of in terms of hypergraphs as well. An alternate proof will now be presented.

Proof. Assume that the conditions in Theorem 4.1 hold. Let $H = (V, E)$ define a hypergraph with three edges, A_1 the edge defined by the top left $a \times b$ rectangle, A_2 the edge defined by the top left $c \times d$ rectangle, and A_3 defined by the last $b \times a$ rectangle that is incident with the $a \times b$ rectangle. Then $|\mathcal{A}_{12}| = 6, |\mathcal{A}_{1,3}| = 4, |\mathcal{A}_{2,3}| = 1$, and $|\mathcal{A}_{1,2,3}| = 2$. Moreover,

$$|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| = 13 > 12.$$

□

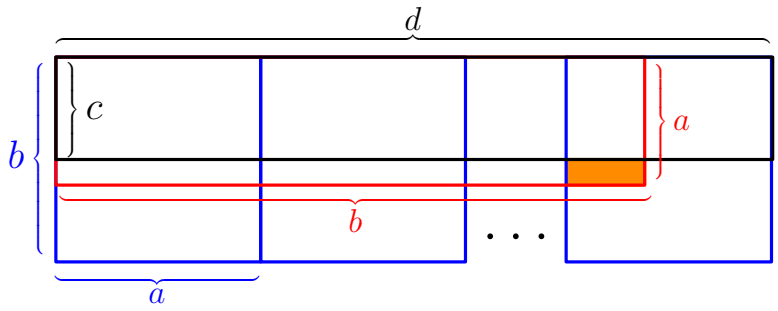

Figure 4.1: Similar Proof to Order 12

The idea of this proof can be generalized to higher orders. In general, we have the following theorem that describes in terms of ordered factor pairs what orders are inadmissible by this technique.

Theorem 4.2. *There does not exist a Factor Pair Latin Square if:*

$$\begin{aligned} n &= a \cdot b = c \cdot d, \\ c &< a, \quad b < d, \\ a + b &> d, \quad \textit{and} \\ a &\mid d. \end{aligned}$$

Proof. Since the first $c \times d$ rectangle must be latin, fill the top left most $c \times d$ rectangle with the n symbols. Since $a \mid d$, we can say that $d = a \cdot k$. So, there are k rectangles of size $b \times a$ that cover the top $c \times d$ block (and more, since $c < a < b$). Now, since $a + b > d$, by substitution, we have that $b > a(k-1)$. Since this is a strict inequality, the top left most $a \times b$ rectangle must pass through strictly more than $k-1$ rectangles of size $b \times a$. Moreover, this means that there is at least one cell that is in the k^{th} $b \times a$ rectangle and not in the original $c \times d$ rectangle that cannot be filled, since it covers the first $k-1$ columns of the top left $c \times d$ rectangle as well as the other entries of the top left $c \times d$ rectangle since that cell is in the k^{th} $b \times a$ rectangle. The top left corner of the proposed factor pair latin square is depicted in Figure 4.1. □

This can also be thought of in terms of hypergraphs as well. An alternate proof will now be presented.

Proof. Assume that the conditions in Theorem 4.2 hold. Let $H = (V, E)$ define a hypergraph with three edges, A_1 the edge defined by the top left $a \times b$ rectangle, A_2 the edge defined by the top left $c \times d$ rectangle, and A_3 defined by the last $b \times a$ rectangle that is incident with the $a \times b$ rectangle. Then $|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{1,2,3}| = n$, since it completely encompasses the top left $a \times b$ block. Moreover, $|\mathcal{A}_{2,3}| \neq 0$, since $|\mathcal{A}_{1,2,3}| \neq 0$. Hence

$$|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| > n.$$

□

A natural question arises. Namely, can we generalize this result to three different sized rectangles instead of two? The following theorem generalizes the above result to three rectangles of different sizes.

Theorem 4.3. *There does not exist a Factor Pair Latin Square if:*

$$\begin{aligned} n \;=\; a \cdot b \;&=\; c \cdot d \;&=\; f \cdot g, \\ a < c < f \quad &, \quad g < d < b, \\ g\left\lfloor \tfrac{d}{g} \right\rfloor \;&<\; d, \qquad \textit{and} \\ g\left\lceil \tfrac{d}{g} \right\rceil \;&\geq\; b. \end{aligned}$$

Proof. Fill in the first $a \times b$ rectangle with the n symbols. Since $g \mid b$, we can say that $b = k \cdot g$. So, there are k rectangles of size $f \times g$ that cover the top $a \times b$ block. Now, since $d + g > b$, by substitution, we have that $d > g(k-1)$. Since this is a strict inequality, the top left most $c \times d$ rectangle must pass through strictly more than $k-1$ rectangles of size $f \times g$. Moreover, this means that there is at least one cell that is in the k^{th} $c \times d$ rectangle and not in the original $a \times b$ rectangle that cannot be filled, since it is covered by the first

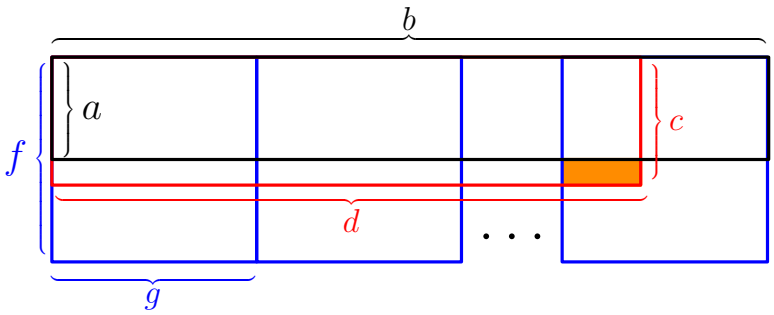

Figure 4.2: Three Conditions

$k-1$ columns of the top left $a \times b$ rectangle as well as the other entries of the top left $a \times b$ rectangle since that cell is in the k^{th} $c \times d$ rectangle. The top left corner of the proposed factor pair latin square is depicted in Figure 4.2. □

As before, this can be thought of in terms of hypergraphs as well. An alternate proof will now be presented.

Proof. Assume that the conditions in Theorem 4.3 hold. Let $H = (V, E)$ define a hypergraph with three edges, A_1 the edge defined by the top left $a \times b$ rectangle, A_2 the edge defined by the top left $c \times d$ rectangle, and A_3 defined by the last $f \times g$ rectangle that is incident with the $a \times b$ rectangle. Then $|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{1,2,3}| = n$, since it completely encompasses the top left $a \times b$ block. Moreover, $|\mathcal{A}_{2,3}| \neq 0$, since $|\mathcal{A}_{1,2,3}| \neq 0$. Hence

$$|\mathcal{A}_{1,2}| + |\mathcal{A}_{1,3}| + |\mathcal{A}_{2,3}| + |\mathcal{A}_{1,2,3}| > n.$$

□

This all started with the observation that there does not exist a factor pair latin square of order twelve. Moreover, the following theorem generalizes this to say that there cannot be a factor pair latin square of any order that is divisible by twelve.

Theorem 4.4. *There does not exist a factor pair latin square of order n if $n \equiv 0 \pmod{12}$.*

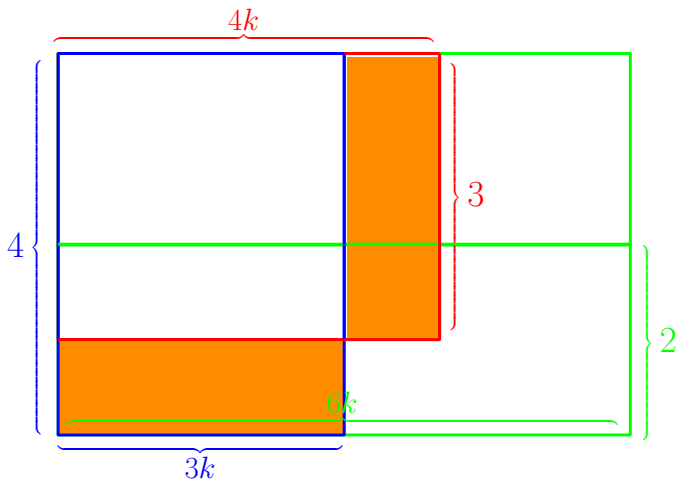

Figure 4.3: $n \equiv 0 \pmod{12}$

Proof. Since $n \equiv 0 \pmod{12}$, $n = 12k$ for some $k \in \mathbb{Z}^+$. Moreover, $n = 2(6k) = 3(4k)$. Hence, $F = \{\ldots, 2 \times 6k, 6k \times 2, 6 \times 2k, 2k \times 6, 3 \times 4k, 4k \times 3, 4 \times 3k, 3k \times 4, \ldots\}$. Examining the $3 \times 4k$ block along with the $4 \times 3k$ block, we see that the symbols to the right of the $4 \times 3k$ block that are within the $3 \times 4k$ block must have the same symbol set as those below the $3 \times 4k$ block contained within the $4 \times 3k$ block. Now, if one were to examine the $2 \times 6k$ blocks, one would see that the second $2 \times 6k$ block cannot be latin, as demonstrated by Figure 4.3 □

Naturally, one might ask whether or not a similar thing can be said for other orders that satisfy Theorem 4.3. The following corollary states that if a number m satisfies the hypothesis to Theorem 4.3 then every multiple of m is inadmissible as a factor pair latin square.

Corollary 4.5. *If m satisfies the hypothesis of Theorem 4.3 and $n \equiv 0 \pmod{m}$ then there does not exist a factor pair latin square of order n.*

Proof. Let m satisfy the hypothesis of Theorem 4.3. That is to say that

$$\begin{aligned} m &= a \cdot b &&= c \cdot d &&= f \cdot g, \\ & a < c < f \quad , \quad g < d < b, \\ & g\left\lfloor \tfrac{d}{g} \right\rfloor &&< d, && \text{and} \\ & g\left\lceil \tfrac{d}{g} \right\rceil &&\geq b. \end{aligned}$$

Moreover, since $n \equiv 0 \pmod{m}$, $n = km$ for some positive integer k. That is to say that there exist three ordered factor pairs such that

$$\begin{aligned} n &= a \cdot bk &&= c \cdot dk &&= f \cdot gk, \\ & a < c < f \quad , \quad gk < dk < bk, \\ gk\left\lfloor \tfrac{dk}{gk} \right\rfloor &= gk\left\lfloor \tfrac{d}{g} \right\rfloor &&< dk, && \text{and} \\ gk\left\lceil \tfrac{dk}{gk} \right\rceil &= gk\left\lceil \tfrac{d}{g} \right\rceil &&\geq bk. \end{aligned}$$

Since each inequality is simply multiplied by k, the above inequalities also satisfy the hypothesis for Theorem 4.3. Hence, there does not exist a factor pair latin square of order n. □

4.2 Generalizing Order Twenty

Factor pair latin squares can be constructed using methods from Chapter 2 for orders thirteen, fourteen, sixteen, seventeen, and nineteen. Moreover, there exists a factor pair latin square of order fifteen as shown in Figure 4.4 as well as a factor pair latin square of order eighteen as shown in Figure 4.5. One might notice, however, that order twenty is missing from this list. The following theorem give a reasoning as to why that is the case.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	7	8	9	10	11	12	13	14	15	1	2	3	4	5
11	12	13	14	15	1	2	3	4	5	6	7	8	9	10
4	5	9	2	3	7	1	10	15	8	13	14	6	11	12
10	14	15	8	12	13	5	6	11	3	4	9	1	2	7
7	1	6	11	13	2	4	9	12	14	3	5	10	15	8
2	3	4	1	6	5	8	7	10	9	12	15	11	13	14
5	8	10	7	9	12	14	15	13	11	2	1	4	3	6
12	13	11	15	14	3	6	1	2	4	8	10	5	7	9
9	15	14	10	4	8	3	11	5	6	7	13	2	12	1
3	6	1	5	2	4	9	12	7	13	14	8	15	10	11
8	11	12	13	7	10	15	14	1	2	5	3	9	6	4
13	4	2	12	1	9	10	5	6	7	15	11	14	8	3
14	9	5	3	11	15	13	4	8	12	10	6	7	1	2
15	10	7	6	8	14	11	2	3	1	9	4	12	5	13

Figure 4.4: FPLS(15)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
12	13	11	10	17	14	18	16	15	2	4	5	7	1	8	6	3	9
8	15	7	16	18	9	14	3	6	1	13	17	5	4	10	12	2	11
4	10	5	2	12	11	1	17	13	14	16	18	3	6	9	7	8	15
14	16	9	1	8	7	4	2	11	3	15	6	12	17	18	13	10	5
6	17	18	15	13	3	10	12	5	7	8	9	11	16	2	4	1	14
3	7	17	14	1	4	9	15	16	12	10	2	8	18	11	5	13	6
5	11	8	12	2	10	6	13	18	4	1	14	9	15	3	17	7	16
9	18	6	13	16	15	5	11	8	17	3	7	10	2	1	14	12	4
10	1	2	3	7	17	12	4	14	16	9	15	6	13	5	11	18	8
15	12	16	5	11	8	17	10	3	18	6	13	4	7	14	1	9	2
13	14	4	9	6	18	2	1	7	8	5	11	16	12	17	3	15	10
11	9	15	6	10	2	13	14	1	5	7	3	17	8	4	18	16	12
7	5	12	17	3	16	8	18	4	15	2	10	1	11	6	9	14	13
18	4	13	8	14	1	11	9	12	6	17	16	15	10	7	2	5	3
2	3	10	7	15	5	16	6	17	13	14	4	18	9	12	8	11	1
17	6	1	18	9	13	15	5	2	11	12	8	14	3	16	10	4	7
16	8	14	11	4	12	3	7	10	9	18	1	2	5	13	15	6	17

Figure 4.5: FPLS(18)

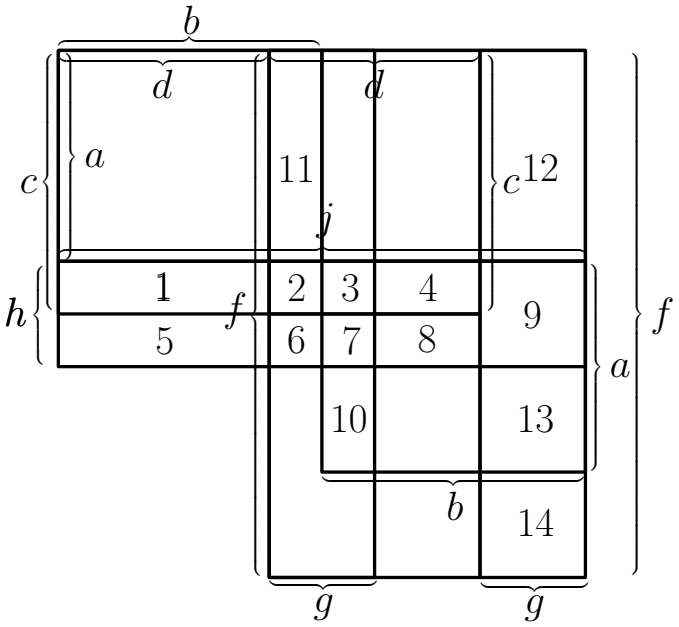

Figure 4.6: Generalization of Order Twenty

Theorem 4.6. *There does not exist a factor pair latin square if:*

$$\begin{aligned} n = a \times b = c \times d &= f \times g = h \times j, \\ h < a < c < f &, g < d < b < j, \\ b \mid j, g \mid d &, h \mid a, c \mid f, \\ h > c - a &, g > b - d, \text{ and} \\ \left(b - g\left\lfloor \tfrac{b}{g} \right\rfloor\right)\left(h\left\lceil \tfrac{c}{h} \right\rceil - c\right) &< \left(2a - h\left\lceil \tfrac{c}{h} \right\rceil\right)\left(g\left\lceil \tfrac{b}{g} \right\rceil - b\right). \end{aligned}$$

Proof. First and foremost, it should be noted that if the above conditions are satisfied, then there exists a configuration of rectangles as seen in Figure 4.6 in the top left corner of the proposed factor pair latin square. This proof will show that if the above conditions are satisfied, then no symbol can go into Region 10 as depicted in Figure 4.6. First, fill in the $h \times j$ rectangle in Figure 4.6. By the top left $a \times b$ rectangle and the top left $b \times c$ rectangle, Region 1 must have the same symbol set as Region 11. Moreover, by this fact and by the intersection of the top right $a \times b$ rectangle along with the top right $c \times d$ rectangle, the union of Regions $1, 2, 3$, and 4 must have the same symbol set as Region 12. Moreover,

every symbol from Region 8 must go into Region 14 since the right most $f \times g$ rectangle must have those symbols and they cannot be in Regions 9 or 13 (due to the bottom right $a \times b$ rectangle). So, the symbols in Region 6 can only either go in Region 13 or Region 14. Notice that the condition $\left(b - g\left\lfloor \frac{b}{g} \right\rfloor\right)\left(h\left\lceil \frac{c}{h} \right\rceil - c\right) < \left(2a - h\left\lceil \frac{c}{h} \right\rceil\right)\left(g\left\lceil \frac{b}{g} \right\rceil - b\right)$ essentially says that there are more symbols in Region 10 than there are in Region 6. If the symbols in Region 6 are placed in region 14, then all of the symbols from Region 5 must go into Region 13. Moreover, Region 10 cannot be filled, since it has seen every symbol. Moreover, if the symbols from Region 6 are in Region 13, then Region 10 can have $\left(b - g\left\lfloor \frac{b}{g} \right\rfloor\right)\left(h\left\lceil \frac{c}{h} \right\rceil - c\right)$ symbols in it. But, we said before that $\left(b - g\left\lfloor \frac{b}{g} \right\rfloor\right)\left(h\left\lceil \frac{c}{h} \right\rceil - c\right) < \left(2a - h\left\lceil \frac{c}{h} \right\rceil\right)\left(g\left\lceil \frac{b}{g} \right\rceil - b\right)$; so, there will be at least one cell in Region 10 that has seen every symbol and is therefore unable to be filled. □

Corollary 4.7. *There does not exist a factor pair latin square of order twenty*

Proof. By Theorem 4.6, let $a = 4$, $b = 5$, $c = 5$, $d = 4$, $f = 10$, $g = 2$, $h = 2$, and $j = 10$. □

As before, Theorem 4.6 can extended to say that if order m satisfies the hypothesis of Theorem 4.6 and $n \equiv 0 \pmod{m}$, then there does not exist a factor pair latin square of order n.

Corollary 4.8. *If m satisfies the hypothesis of Theorem 4.6 and $n \equiv 0 \pmod{m}$, then there does not exist a factor pair latin square of order n.*

Proof. Say m satisfies the hypothesis of Theorem 4.6. That is, there exist a set of four ordered factor pairs such that

$$\begin{aligned} m = a \times b = c \times d &= f \times g = h \times j, \\ h < a < c < f &, g < d < b < j, \\ b \mid j, g \mid d &, h \mid a, c \mid f, \\ h > c - a &, g > b - d, \text{ and} \\ \left(b - g\left\lfloor \frac{b}{g} \right\rfloor\right)\left(h\left\lceil \frac{c}{h} \right\rceil - c\right) &< \left(2a - h\left\lceil \frac{c}{h} \right\rceil\right)\left(g\left\lceil \frac{b}{g} \right\rceil - b\right). \end{aligned}$$

Let $n \equiv 0 \pmod{m}$. That is to say that there exists a positive integer k such that $n = km$. Moreover, there exist a set of four ordered factor pairs such that

$$\begin{aligned} n = a \times bk = c \times dk &= f \times gk = h \times jk, \\ h < a < c < f &, gk < dk < bk < jk, \\ bk \mid jk, gk \mid dk &, h \mid a, c \mid f, \\ h > c - a &, gk > bk - dk, \text{ and} \\ \left(bk - gk \left\lfloor \tfrac{bk}{gk} \right\rfloor\right)\left(h \left\lceil \tfrac{c}{h} \right\rceil - c\right) &< \left(2a - h \left\lceil \tfrac{c}{h} \right\rceil\right)\left(gk \left\lceil \tfrac{bk}{gk} \right\rceil - bk\right). \end{aligned}$$

Since all of the inequalities are either the same as before or multiples of k, n also satisfies the hypothesis of Theorem 4.6. Hence, there does not exist a factor pair latin square of order n. □

4.3 General Order Twenty-Eight

A factor pair latin square of order twenty-one has been constructed in Figure 4.7. One might also notice that Chapter 2 gives us constructions for factor pair latin squares for orders twenty-two, twenty-three, and twenty-five through twenty-seven. Also, there cannot exist a factor pair latin square of order twenty-four since twenty-four is divisible by twelve. Twenty eight; however, is one that is not covered by these constructions. The reason for that is that there does not exist a factor pair latin square of order twenty-eight as we will see in Corollary 4.10.

Before proving this corollary, however, we notice that if we look at how many times it takes for rectangle of size a to completely cover a rectangle of size b, it will take approximately the same number of b sized rectangles to completely cover a rectangle of size a.

Now we turn our attention to showing that there does not exist a factor pair latin square of order twenty-eight. In fact, we will display another infinite family of forbidden orders of which twenty eight is the first member.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
15	16	17	18	19	20	21	1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15	16	17	18	19	20	21	1	2	3	4	5	6	7
19	20	21	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
12	13	14	15	16	17	18	19	20	21	1	2	3	4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	1	2	3	4
4	11	18	7	14	21	3	10	17	6	13	20	2	9	16	5	12	19	1	8	15
16	5	12	19	1	8	15	4	11	18	7	14	21	3	10	17	6	13	20	2	9
10	17	6	13	20	2	9	16	5	12	19	1	8	15	4	11	18	7	14	21	3
2	1	4	3	6	5	8	7	10	9	16	11	12	13	14	15	20	21	17	18	19
7	14	9	10	11	12	13	2	18	15	17	21	19	20	6	1	4	3	8	5	16
18	15	19	17	21	16	20	3	14	4	6	8	1	5	2	10	9	12	7	11	13
3	8	11	9	4	15	1	21	19	2	20	13	7	17	18	14	5	16	6	12	10
13	21	20	14	7	18	17	6	12	5	3	10	9	16	11	8	19	2	4	15	1
6	12	2	5	10	19	16	11	4	1	14	15	18	8	3	7	13	17	21	9	20
9	3	1	2	8	4	5	13	7	11	10	6	14	12	21	18	16	20	15	19	17
11	10	13	6	15	7	12	17	1	20	21	18	16	19	9	4	2	5	3	14	8
20	18	16	21	17	14	19	9	3	8	5	4	15	2	7	12	11	10	13	1	6
14	4	5	16	3	1	2	20	8	19	9	17	11	10	13	21	15	6	18	7	12
17	7	15	12	18	11	6	14	21	13	2	3	4	1	20	19	8	9	10	16	5
21	19	8	20	13	9	10	18	15	16	12	7	5	6	17	3	1	14	11	4	2

Figure 4.7: FPLS(21)

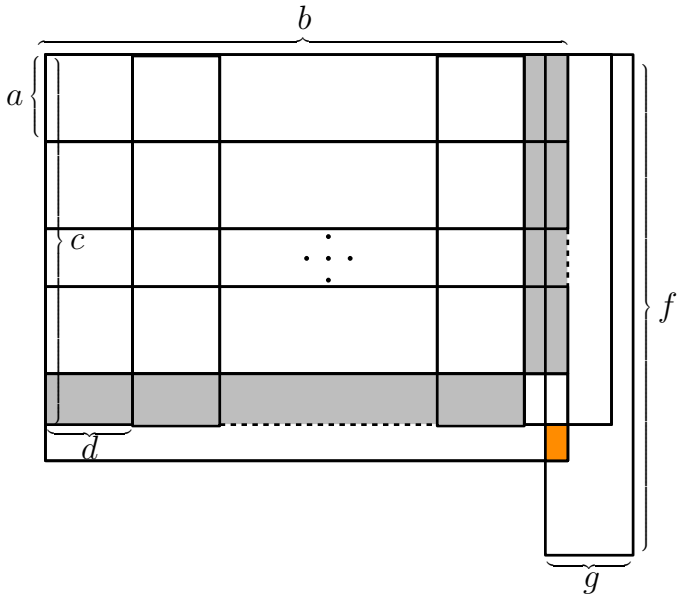

Figure 4.8: Generalization of Order Twenty-Eight

Theorem 4.9. *There does not exist a factor pair latin square if*

$$\begin{aligned} n = a \times b &= c \times d = f \times g, \\ a < c < f, &\quad g < d < b, \\ d &\nmid b, \qquad \textit{and} \\ d \left\lfloor \tfrac{b}{d} \right\rfloor < g \left\lfloor \tfrac{b}{g} \right\rfloor &< d \left\lceil \tfrac{b}{d} \right\rceil \leq g \left\lceil \tfrac{b}{g} \right\rceil . \end{aligned}$$

Proof. If the above conditions hold, the top left corner of the $n \times n$ grid looks like Figure 4.8. Let $s = \frac{a}{c} = \frac{d}{b}$. Since the rectangles overlap each other in the same number of times, the region from the first row to the $(sa)^{\text{th}}$ row and from the first column to the $(sd)^{\text{th}}$ column sees some symbols s times and others $s-1$ times. The symbols that it sees $s-1$ times must be in the region R_1 defined as the region from the $(sa+1)^{\text{th}}$ row to the c^{th} row in the vertical direction and from the first column to the $(sd)^{\text{th}}$ column in the horizontal direction as well as the region R_2 defined as the region from the first row to the $(sa)^{\text{th}}$ row in the vertical direction and from the $(sd+1)^{\text{th}}$ column to the b^{th} column in the horizontal direction. Moreover, R_1 and R_2 must have the same symbol set. In a similar fashion, the region R_3 defined as the region from the b^{th} column to the $(s\,(d+1))^{\text{th}}$ column and from

the first row to the c^{th} row must have the same symbol set as the region R_4 defined as the region from the c^{th} row to the $(s\,(a+1))^{\text{th}}$ row and the first column to the g^{th} column, since the $(s\,(a+1))^{\text{th}}$ rectangle intersects the $(s\,(d+1))^{\text{th}}$ rectangle. Fill R_3 with the n symbols, ensuring that R_1 and R_2 have the same symbol set and R_3 and R_4 have the same symbol set. Since $d \cdot \left\lfloor \frac{b}{d} \right\rfloor < g \cdot \left\lfloor \frac{b}{g} \right\rfloor < d \cdot \left\lceil \frac{b}{d} \right\rceil \leq g \cdot \left\lceil \frac{b}{g} \right\rceil$, the region from the $(c+1)^{\text{th}}$ row to the $(s\,(a+1))^{\text{th}}$ row and from the g^{th} column to the b^{th} column cannot be filled. □

Corollary 4.10. *There does not exist a factor pair latin square of order twenty-eight*

Proof. By Theorem 4.9, let $a = 4$, $b = 7$, $c = 7$, $d = 4$, $f = 14$, and $g = 2$. □

Corollary 4.10 cannot be proven by the hypergraph results proven in Chapter 3, however, since we can find an embedded C_5 which has to be 2-colored inside of the hypergraph produced by the constraints in Theorem 4.9. Define a graph whose vertices are the union of cells in one rectangle but not another. Two vertices are connected if only if their symbol sets must be distinct. Weight each vertex with the number of cells within the intersection. More specifically, define the vertices v_1 as the cells in the uppermost 4×7 rectangle but not the cells in the second 7×4 rectangle (weight 16), v_2 as the cells in the uppermost 4×7 rectangle but not the cells in the first 7×4 rectangle (weight 12), v_3 as the cells in the second 4×7 rectangle but not the cells in the second 7×4 rectangle (weight 12), v_4 as the cells in the second 4×7 rectangle but not the cells in the first 7×4 rectangle (weight 16), and v_5 as the cell $(8, 8)$, which is in the fourth 14×2 rectangle (weight 1). This produces $C_5 = (v_1, v_2, v_4, v_5, v_3)$ which must be three colored while there are only two colors available, since there are only two possible symbol sets in this graph. This construction is depicted in Figure 4.9 Moreover, this falls into the case when there are five edges in a hypergraph and the necessary conditions are not sufficient.

As before, we can extend this to orders which are multiple of a number that is deemed inadmissible by Theorem 4.9.

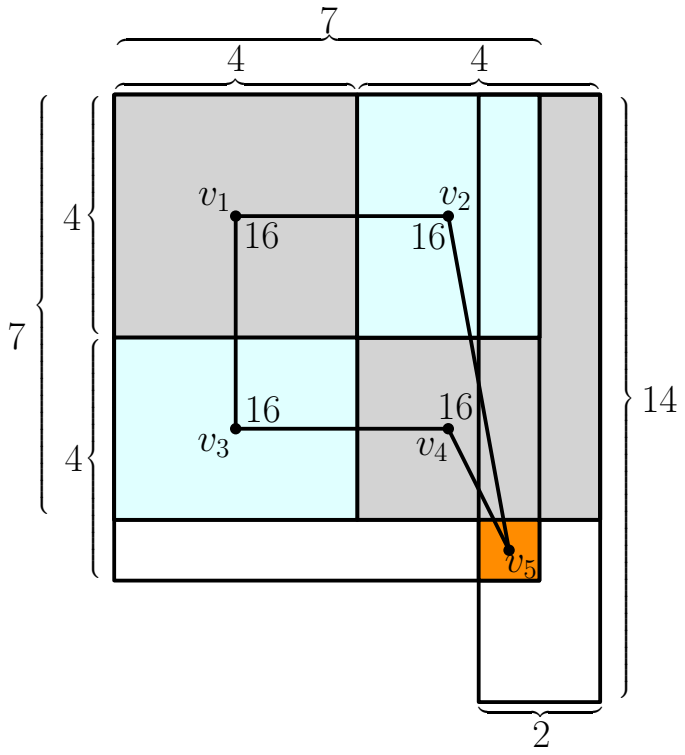

Figure 4.9: Five Hyper Edges in Order Twenty Eight

Corollary 4.11. *If m satisfies the hypothesis of Theorem 4.9 and $n \equiv 0 \pmod{m}$ then there does not exist a factor pair latin square of order n.*

Proof. If m satisfies Theorem 4.9, then there exist three ordered factor pairs such that

$$\begin{aligned} m = a \times b &= c \times d = f \times g, \\ a < c < f, &\quad g < d < b, \\ d &\nmid b, \qquad \text{and} \\ d \left\lfloor \tfrac{b}{d} \right\rfloor < g \left\lfloor \tfrac{b}{g} \right\rfloor &< d \left\lceil \tfrac{b}{d} \right\rceil \leq g \left\lceil \tfrac{b}{g} \right\rceil . \end{aligned}$$

Moreover, if n is a multiple of m, then $n = mk$ for some positive integer k. Furthermore, there exist three ordered factor pairs of n such that

$$\begin{aligned} n = a \times bk &= c \times dk = f \times gk, \\ a < c < f, &\quad gk < dk < bk, \\ dk &\nmid bk, \qquad \text{and} \\ dk \left\lfloor \tfrac{bk}{dk} \right\rfloor < gk \left\lfloor \tfrac{bk}{gk} \right\rfloor &< dk \left\lceil \tfrac{bk}{dk} \right\rceil \leq gk \left\lceil \tfrac{bk}{gk} \right\rceil . \end{aligned}$$

Since every inequality is either the same as m or it is a multiple of k. In either case, the above set of inequalities satisfy the hypothesis of Theorem 4.9. Henceforth, if n is a multiple of a number that satisfies Theorem 4.9, then there does not exist a factor pair latin square of order n. □

The theorems discussed in this Chapter give us an infinite family of inadmissible orders for factor pair latin squares. A program which can be found in Appendix A has been written to determine the inadmissible values given by these results up to five thousand. The results of the code can be found in Appendix B.

Chapter 5

Results on Quasi-Factor Pair Latin Squares

Up until this point, we have been concerned with making every ordered factor pair of an $n \times n$ grid an (a, b)-Sudoku latin square. We have seen that this can sometimes be done; however, for some orders it is simply impossible. What happens if we are only concerned with some of the ordered factor pairs? When a factor pair latin square does not exist, how many ordered factor pairs can we have and still satisfy the conditions of an (a, b)-Sudoku latin square? Let P be a list of ordered factor pairs. A *semi-factor pair latin square* of order n, denoted SFPLS(n, P), is an (a, b)-Sudoku latin square of order n for every ordered factor pair in P. Let F be the largest of such sets of ordered factor pairs such that a SFPLS(n, P) exists. A *quasi-factor pair latin square* of order n, denoted QFPLS(n, F), is an (a, b)-Sudoku latin square of order n for every ordered factor pair in F.

5.1 Order Twelve

The first order for which there does not exist a factor pair latin square is order twelve; so, that is a natural place to start. It should be noted that removing either the 6×2 or the 2×6 ordered factor pairs from the list of ordered factor pairs of order twelve will not yield a QFPLS$(12, \{1 \times 12, 12 \times 1, 2 \times 6, 3 \times 4, 4 \times 3\})$ or a QFPLS$(12, \{1 \times 12, 12 \times 1, 6 \times 2, 3 \times 4, 4 \times 3\})$ by the exact same proof of Theorem 4.1 (the transpose of the $n \times n$ array would violate Theorem 4.1). Removing either the 3×4 or the 4×3, however, will yield a QFPLS$(12, \{1 \times 12, 12 \times 1, 2 \times 6, 6 \times 2, 3 \times 4\})$ or QFPLS$(12, \{1 \times 12, 12 \times 1, 2 \times 6, 6 \times 2, 4 \times 3\})$. A QFPLS$(12, \{1 \times 12, 12 \times 1, 2 \times 6, 6 \times 2, 3 \times 4\})$ is presented in Figure 5.1.

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

Figure 5.1: QFPLS$(12, \{1 \times 12, 12 \times 1, 2 \times 6, 6 \times 2, 3 \times 4\})$

5.2 Order Twenty-Four

We took the proof of Theorem 4.1 and generalized it to Theorem 4.2. Moreover, the 3×4 as well as the 4×3 regions can be thought of as $c \times d$ blocks in Theorem 4.2. One is tempted to think that if we are given an inadmissible value n for a factor pair latin square that satisfies the hypothesis of Theorem 4.2 and remove the $c \times d$ ordered factor pair from the list of ordered factor pairs, then we can construct a quasi-factor pair latin square of that order where $|F|$ is one less than the number of ordered factor pairs of n. This is precisely what the following conjecture states.

Conjecture 5.1. *If we were to remove all of the $c \times d$ ordered factor pairs as stated in Theorem 4.2 from the list F, we would get a QFPLS(n, F).*

This conjecture is false; however, since Lemma 5.2 shows that there does not exist a quasi-factor pair latin square of order twenty-four which has seven ordered factor pairs. Twenty-four is an important number for this result as none of the other negative results from Chapter 4 covers order twenty-four; so, if this conjecture has any shot at being true, it would have to work for order twenty-four. Theorem 5.3, however, will show that this conjecture is false.

Lemma 5.2. *There does not exist a QFPLS*$(24, \{1\times24, 24\times1, 2\times12, 12\times2, 3\times8, 4\times6, 6\times4\})$ *or a QFPLS*$(24, \{1 \times 24, 24 \times 1, 2 \times 12, 12 \times 2, 8 \times 3, 4 \times 6, 6 \times 4\})$.

Proof. Without loss of generality, use $F = \{1 \times 24, 24 \times 1, 2 \times 12, 12 \times 2, 3 \times 8, 4 \times 6, 6 \times 4\}$. Note that it does not matter whether we use 3×8 or 8×3 as the bad ordered factor pair as a transposition will get the other. Fill in the first (top left) 2×12 factor pair with the twenty-four symbols. Now, cells $(3, 7)$ and $(3, 8)$ cannot be filled, for they are in a 3×8 block that contains the first eight columns of the above 2×12 and a 4×6 block that contains the last four columns of the above 2×12. Moreover, they contain all of the symbols, making it impossible to fill the remaining cells. □

As before, it may be possible to define a QFPLS(n, F) for these inadmissible values; however, it would be useless to pick any of the ordered factor pairs that satisfy Theorem 4.2 of the form $a \times b$, since the result of the Theorem 4.2 would hold for the transpose of the proposed quasi-factor pair latin square.

As for order twenty-four, removing the 2×12 or the 12×2 ordered factor pairs would not result in a quasi-factor pair latin square of order twenty-four with seven ordered factor pairs. Lemma 5.2 tells us that removing either the 8×3 or the 3×8 ordered factor pair will not produce the desired quasi-factor pair latin square either. The last ordered factor pair to worry about would be the 4×6 or the 6×4. Removing one of these factor pairs, however, will not produce a quasi-factor pair latin square of order twenty-four with seven ordered factor pairs as the following theorem will show.

Theorem 5.3. *There does not exist a quasi-factor pair latin square of order twenty-four with seven rectangles.*

Proof. We have already seen from Lemma 5.2 that removing the 3×8 or the 8×3 ordered factor pairs will not yield a quasi-factor pair latin square of order twenty-four with seven ordered factor pairs. We have also observed above that removing either the 2×12 or the 12×2 ordered factor pairs will not yield a quasi-factor pair latin square of

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
13	14	15	16	17	18	19	20	21	22	23	24	1	2	3	4	5	6	7	8	9	10	11	12
7	8	9	10	11	12	1	2	3	4	5	6	19	20	21	22	23	24	13	14	15	16	17	18
19	20	21	22	23	24	13	14	15	16	17	18	7	8	9	10	11	12	1	2	3	4	5	6
5	17	11	23	3	4	9	10	7	19	1	13	6	18	12	24	15	16	21	22	8	20	2	14
6	18	12	24	15	16	21	22	8	20	2	14	5	17	11	23	3	4	9	10	7	19	1	13
9	10	7	19	1	13	5	17	11	23	3	4	21	22	8	20	2	14	6	18	12	24	15	16
21	22	8	20	2	14	6	18	12	24	15	16	9	10	7	19	1	13	5	17	11	23	3	4
11	23	6	18	7	8	16	24	1	13	9	10	3	4	5	17	19	20	12	15	2	14	21	22
3	4	5	17	19	20	12	15	2	14	21	22	11	23	6	18	7	8	16	24	1	13	9	10
15	16	1	13	10	21	11	23	5	17	7	19	12	24	2	14	22	9	3	4	6	18	8	20
12	24	2	14	22	9	3	4	6	18	8	20	15	16	1	13	10	21	11	23	5	17	7	19
2	1	4	3	6	5	8	7	10	9	12	11	14	13	16	15	18	17	20	19	22	21	24	23
14	13	16	15	18	17	20	19	22	21	24	23	2	1	4	3	6	5	8	7	10	9	12	11
8	7	10	9	12	11	2	1	4	3	6	5	20	19	22	21	24	23	14	13	16	15	18	17
20	19	22	21	24	23	14	13	16	15	18	17	8	7	10	9	12	11	2	1	4	3	6	5
17	5	23	11	4	3	10	9	19	7	13	1	18	6	24	12	16	15	22	21	20	8	14	2
18	6	24	12	16	15	22	21	20	8	14	2	17	5	23	11	4	3	10	9	19	7	13	1
10	9	19	7	13	1	17	5	23	11	4	3	22	21	20	8	14	2	18	6	24	12	16	15
22	21	20	8	14	2	18	6	24	12	16	15	10	9	19	7	13	1	17	5	23	11	4	3
23	11	18	6	8	7	24	16	13	1	10	9	4	3	17	5	20	19	15	12	14	2	22	21
4	3	17	5	20	19	15	12	14	2	22	21	23	11	18	6	8	7	24	16	13	1	10	9
16	15	13	1	21	10	23	11	17	5	19	7	24	12	14	2	9	22	4	3	18	6	20	8
24	12	14	2	9	22	4	3	18	6	20	8	16	15	13	1	21	10	23	11	17	5	19	7

Figure 5.2: QFPLS$(24, \{1 \times 24, 24 \times 1, 2 \times 12, 12 \times 2, 4 \times 6, 6 \times 4\})$

order twenty-four with seven ordered factor pairs. Without loss of generality, let $F = \{1 \times 12, 12 \times 1, 2 \times 6, 6 \times 2, 3 \times 8, 8 \times 3, 4 \times 6\}$. A similar argument can be made if one were to remove the 4×6 ordered factor pair instead of the 6×4 ordered factor pair. Let $a = 2$, $b = 12$, $c = 3$, $d = 8$, $f = 4$, and $g = 6$ in Theorem 4.3. □

One can, however, produce a quasi-factor pair latin square by removing two ordered factor pairs. Figure 5.2 is a QFPLS$(24, \{1 \times 24, 24 \times 1, 2 \times 12, 12 \times 2, 4 \times 6, 6 \times 4\})$.

Chapter 6

Mappings to other Problems

It is important to be able to word this problems in different ways in order to use theorems and techniques from other problems in mathematics. The most natural problem to map the problem of finding a factor pair latin square of order n is the exact cover problem described below, which is an NP-complete problem.

Of course, if one were to attempt at finding factor pair latin squares of a given order that either hasn't been explicitly found or ruled out by a previous theorem, one could write a brute force backtracking algorithm to attempt to come across a factor pair latin square of that order. This of course is not a very efficient way to do it; however, we do this in Appendix C.

6.1 Exact Cover Problem

Of all of the NP-complete problems, the exact cover problem is the easiest NP-complete problem to map the problem of finding a factor pair latin square of order n. Given a collection $\mathscr{C}$ subsets of a set C, an *exact cover* is a sub-collection $\mathscr{C}^*$ of $\mathscr{C}$ such that each element in C is contained in exactly one subset in $\mathscr{C}^*$. That is to say that the intersection of any two sets in $\mathscr{C}^*$ is empty while the union of all of the subsets in $\mathscr{C}^*$ is all of C. Equivalently, this can be thought of as a bipartite graph. A *bipartite graph* is a graph G whose vertices can be partitioned into two sets A and B such that every edge in G connects a vertex in A to a vertex in B. Define a bipartite graph H with bipartition $(\mathscr{C}, C)$, where the vertices on the $\mathscr{C}$ side are subsets of C and the vertices on the C side are elements of the set C. We connect vertices on the $\mathscr{C}$ side with vertices on the C side if and only if the set on the C side is a

subset of the set on the $\mathscr{C}$ side. We are looking to pick a set of vertices on the $\mathscr{C}$ side that cover (or hit) each vertex on the C side exactly once.

To explicitly find a solution to a factor pair latin square of order n, one can translate the factor pair latin square of order n into an exact cover problem. We will use the notion of the bipartite graph in order to draw this connection. Define a bipartite graph with bipartition $(\mathscr{P}, C)$. Let the n^3 vertices on the $\mathscr{P}$ side be ordered triples (r, c, s) denoting that symbol s is in row r and column c. On the other hand, let the C side have $n^2\,(k+1)$ vertices where k is the number of ordered factor pairs of order n. The first n^2 of which can be identified as each cell in the $n \times n$ grid. That is to say that the first n^2 vertices on the C side can be ordered pairs (r, c) where r denotes the row and c denotes the column of each cell. Now, order the rectangles that are required to be latin in some fashion. Since each factor pair has to see every symbol exactly once, the rest of the n^2k vertices can be labeled as ordered pairs (f, s) where f denotes the rectangle for which the symbol s is in. Moreover, an edge can be drawn connecting vertices from $\mathscr{P}$ to vertices in C if and only if the triplet (r, c, s) from $\mathscr{P}$ is a cell in f of (f, s) on the C side and the symbols are the same. We are looking for a set of vertices on the $\mathscr{P}$ side to cover the C side in such a way that the vertices on the C side are covered by only one vertex from the $\mathscr{P}$ side.

This observation lets us use smarter algorithms such as Donald Knuth's Algorithm X (sometimes known as the Dancing Links Algorithm or the DLX algorithm for short) from [9] to aid in solving factor pair latin squares. Coding for this algorithm can be referenced in Appendix D. It should be noted that the exact cover problem is one of Richard M. Karp's NP-complete problems in [8].

6.2 Resolvable 1-Designs

A 1-*design*, $1-(v, k, \lambda)$, is a pair $(V, \mathcal{B})$ where V is a set of size v and $\mathcal{B}$ is a collection of b subsets of size k called *blocks* such that each element of V is contained in exactly λ blocks. It should be noted that repeated blocks are allowed at times. A *parallel class* is a set

of blocks that partition the set of points. A 1-design is said to be *resolvable* if there exists a partition R into parallel classes [6].

Theorem 6.1 (Modified from [1])**.** *Let f be the number of ordered factor pairs of order n. A factor pair latin square of order n is equivalent to a resolvable block design on fn points with block size f.*

Proof. Let R_i denote the set of cells which contain symbol i. Since each symbol occurs in every symbol and every factor pair exactly once, R_i forms a partition of the point set V. Moreover, since each cell contains exactly one symbol, every block will occur in exactly one parallel class.

Given a resolvable block design on fn points with block size f, one can create a factor pair latin square. Name each parallel class with a symbol 1 through n. Each parallel class will determine where the symbols should go within the $n \times n$ grid. □

6.3 Multiple Gerechte Designs

A *gerechte design* is an $n \times n$ grid partitioned into n regions (possibly of different shapes and possibly disconnected) with n cells in each region such that each row, column, and region is latin. A *mutliple gerechte design* is a latin square for which multiple gerechte designs are satisfied [1]. Factor pair latin squares are particular kinds of multiple gerechte designs.

A *gerechte skeleton* of order n is an $n \times n$ array whose n^2 cells are partitioned into n regions containing n cells each. E. R. Vaughan has shown in [11] that deciding if a given gerechte skeleton has a completing is NP-complete; however, if the gerechte skeleton is restricted to contiguous regions, the answer is unknown. Similarly, if the regions are required to be rectangles, the solution is unknown. The problem of finding a completion to a factor pair latin square is even more specific, since we are requiring multiple particular gerechte skeletons.

Perhaps more importantly to the design of experiments, a further question is whether or not a design has an orthogonal mate. Two latin squares of size n, $L = a_{i,j}$ on symbol set

S and $L' = b_{i,j}$ on symbol set S', are said to be *orthogonal* if every element in $S \times S'$ occurs exactly once among the n^2 pairs $(a_{i,j}, b_{i,j}), 1 \leq i, j \leq n$. A set of latin squares are *mutually orthogonal* if every pair of latin squares in the set are mutually orthogonal [6].

A natural question is how many mutually orthogonal factor pair latin squares can be found of a given order. The following theorem gives a maximum number of mutually orthogonal factor pair latin squares.

Theorem 6.2 (Modified from [1])**.** *Let d denote the maximum size of the intersection between any two rectangles in a factor pair latin square of order n. There exists at most $n-d$ mutually orthogonal factor pair latin squares of order n.*

Proof. Say that rectangles A_1 and A_2 have the biggest intersection. Moreover, let $d = |A_1 \cap A_2|$. Let c be a cell in $A_1 \setminus A_2$. By renaming the cells in each latin square in the set of mutually orthogonal factor pair latin squares, we can say that cell c has symbol 1 in cell c of each of the mutually orthogonal factor pair latin squares. Moreover, symbol 1 must occur exactly once in A_2 and not in A_1 in each of the factor pair latin squares; however, each subsequent factor pair latin square must have symbol 1 in a different cell within A_2, since $(1, 1)$ has already occurred in cell c. Hence, there can be at most $|A_2 \setminus A_1|$ mutually orthogonal factor pair latin squares. □

R. A. Fisher suggests that Latin squares can be used in many different agricultural experiments in [1]. Latin squares by themselves are good at eliminating two nuisance variables; however, they lack the ability to take into account different kinds of terrain or subtleties within soil samples around a given plot. In particular, gerechte designs along with factor pair latin squares can be used to ensure that treatments are evenly distributed among different types of soils present on a given square plot of land. Factor pair latin squares are nice for this purpose, since it is typical and convenient to have rectangular shaped plots. So, if an experiment is being done where there are two nuisance variables, but also want to take the difference in terrain into consideration, a factor pair latin square is something to consider.

It is also reasonable to remove more than two nuisance variables or to do another set of experiments on the same plot in a short period of time. In the latter case, it is sensible to treat the previous experiment as another nuisance variable that should be removed. This can be done by using a factor pair latin square that is mutually orthogonal factor pair latin square.

6.4 Pigeon-Hole Principle

As with Sudoku puzzles, finding a factor pair latin square of a given order can be worded as an integer programming problem. This method is adapted from a similar technique for completing Sudoku puzzles in [10]. Let f be the number of ordered factor pairs of order n. Notice that each factor pair emits n rectangular regions within the factor pair latin square. Moreover, there are fn rectangles that all must be latin (this includes rows and columns). This problem becomes difficult in that some of these regions overlap. Let $S = \{1, 2, \ldots, n\}$ denote the set of symbols, $C = \{1, 2, \ldots, n^2\}$ denote the set of cells, $\mathscr{F}$ denote the set of factor pairs $\{F_1, F_2, \ldots, F_f\}$. For each factor pair in $\mathscr{F}$, let $\mathscr{B}$ be the set of blocks $\{B_1, B_2, \ldots, B_n\}$. Define n^3 variables $x_{i,j}$, where $x_{i,j} = 1$ says that symbol j is assigned to cell i and $x_{i,j} = 0$ otherwise. The n^3 variables must follow two constraints. First, each cell must contain only one symbol. That is to say that for $c \in C$, $\sum_{s \in S} x_{c,s} = 1$. Moreover, each block within each factor pair must be latin; so, for every factor pair in $\mathscr{F}$ and every block $B \in \mathscr{B}$, $\sum_{b \in B} x_{b,s} = 1, s \in S$. Moreover, if the factor pair latin square is partially completed, one must make the prescribed cells equal to 1. That is to say that if cell x_i contains symbol j, one must force the variable $x_{i,j} = 1$. For all unassigned cells $x_{i,j} = 0$ for all $i \in C, j \in S$. Hence, we are looking for a $0, 1$ solution to the set of equations defined above.

The *pigeon-hole principle* says that given a subset $M \subseteq I$ of indices from the symbol set S and D a subset of squares all contained in a single block B such that *a*) $|M| = |D|$ and *b*) $C_p \subseteq M$ for every $p \in D$. Then the elements of M can be removed from C_p for each $p \in B \setminus D$.

The above technique, sometimes referred to as a *rule* gives a way to at least narrow down the choices one can make given a partially filled factor pair latin square of order n. This method is enough to solve most of the difficulty levels in Sudoku; however, it is not known when a factor pair latin square is uniquely determined. This would be a problem for another day.

Chapter 7

Open Problems

This dissertation has discussed several things when dealing with factor pair latin squares. Namely, it has addressed some existential problems; however, necessary and sufficient conditions have not been shown. Progress has been made towards that goal; however, it remains an open problem.

Open Problem 7.1. *What are the necessary and sufficient conditions for a factor pair latin square of order n to exist?*

In Chapter 3, we discussed necessary conditions for when factor pair latin squares exist. Moreover, if four or less rectangles overlap at any given point, necessary and sufficient conditions hold to fill that set of rectangles. A natural question is can this be extended to the entire factor pair latin square. Since we've shown that the necessary conditions in Theorem 3.5 are not sufficient, what conditions need to be added so that an n-uniform hypergraph with five or more edges can be n-rainbow colored?

Open Problem 7.2. *If four or less rectangles overlap throughout an $n \times n$ grid, can the completion of those four rectangles be extended to the entire factor pair latin square?*

Open Problem 7.3. *In an n-uniform hypergraph with five or more edges, what are the necessary and sufficient conditions for properly rainbow n-coloring the hypergraph?*

Simpler problems can be tackled first, however. Namely, are there other constructions for when a factor pair latin square of order n exists. Powers of primes and twice a prime number have been shown in Chapter 2, but what about a prime times another prime?

Open Problem 7.4. *Does there exist a factor pair latin square of order $3p$ where p is a prime numbers?*

Open Problem 7.5. *Does there exist a factor pair latin square of order pq where p and q are both prime numbers?*

Similarly, looking at the list of inadmissible values in Appendix B, it seems that the farther out that we go, the less likely a given number is to be a factor pair latin square. Asymptotically, it seems that perhaps the only admissible orders for factor pair latin squares are primes, powers of primes, and numbers that have prime factorization pq where p and q are both prime numbers.

Open Problem 7.6. *Asymptotically, does there exist any factor pair latin squares other than those that have prime factorization p^α or pq where p and q are prime numbers and alpha is a positive integer?*

We have also done some work in mapping the problem of finding a factor pair latin squares for a given order n. A natural question is what is the complexity of completing a partially filled factor pair latin square?

Open Problem 7.7. *What is the complexity of completing a partially filled factor pair latin square of order n?*

Similarly, only the surface of quasi-factor pair latin squares has been touched. Almost nothing is known as to when a factor pair latin square cannot be formed, what is the largest set of ordered factor pairs one can have within an $n \times n$ grid. A natural approach is to see how many ordered factor pairs need to be removed if n satisfies the hypothesis for Theorems 4.3, 4.6, or 4.9.

Open Problem 7.8. *For what orders are there quasi-factor pair latin squares where only one ordered factor pair needs to be removed?*

Another idea would be to look at the prime factorization of these numbers. It would be nice to have theorems that simply relied on the prime factorization as apposed to the ordered factor pairs.

Open Problem 7.9. *Given a prime factorization of n, what conditions are necessary to guarantee that there does not exist (or that there does exist) a factor pair latin square of order n?*

These are just a few open problems related to factor pair latin squares. There are many directions that one could take this problem. This dissertation aims to serve as a jumping off point into related problems with factor pair latin squares, quasi-factor pair latin squares, gerechte designs, and multiple gerechte designs.

Once we know for what orders factor pair latin squares exist, a natural question to ask would be to ask about how many cells must be prescribed in the factor pair latin square of order n in order for the solution to be unique.

Open Problem 7.10. *Given a partially filled factor pair latin square of order n, what is the least number of cells that need to be prescribed in order to make the solution of the factor pair latin square of order n unique?*

Bibliography

[1] R. A. Bailey, Peter J. Cameron, and Robert Connelly, *Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes*, Amer. Math. Monthly **115** (2008), no. 5, 383–404. MR 2408485

[2] bbi5291, *DLX and Sudoku Solver*, Tue May 26, 2009 3:07 pm.

[3] W. U. Behrens, *Mudra, a.: Statistische methoden für landwirtschaftliche versuche. verlag parey, berlin und hamburg 1958, 344 seiten mit 38 abb. ganzleinen, dm 58,60*, Zeitschrift für Pflanzenernährung, Düngung, Bodenkunde **81** (1958), no. 2, 160–161.

[4] Lloyd L. Dines, *Systems of linear inequalities*, Ann. of Math. (2) **20** (1919), no. 3, 191–199. MR 1502553

[5] Howard Garns, *Number place*, Dell Pencil Puzzles and Word Games **16** (1975), 6.

[6] R. L. Graham, M. Grötschel, and L. Lovász (eds.), *Handbook of combinatorics (vol. 2)*, MIT Press, Cambridge, MA, USA, 1995.

[7] R. Hill, *A first course in coding theory*, Oxford Applied Linguistics, Clarendon Press, 1986.

[8] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), Plenum, New York, 1972, pp. 85–103. MR 0378476 (51 #14644)

[9] Donald E. Knuth, *Dancing links*, Millenial Perspectives in Computer Science (2000), 187 – 214.

[10] J. Scott Provan, *Sudoku: strategy versus structure*, Amer. Math. Monthly **116** (2009), no. 8, 702–707. MR 2572105 (2010k:05044)

[11] E. R. Vaughan, *The complexity of constructing gerechte designs*, Electron. J. Combin. **16** (2009), no. 1, Research Paper 15, 8. MR 2475538 (2009k:05041)

Appendices

Appendix A
Inadmissibility Code

The following code to compute the inadmissible values described in Chapter 4 was written in Python:

```
import math
import sys
import time

# This program is intended to list some of the inadmissible numbers
# from 0 to some desired amount for Factor Pair Latin Squares (FPLS).

# Inadmissible values for factor pair latin square up to 5000 using
# Theorem 4.3, Theorem 4.6, and Theorem 4.9
# will be determined.

EndValue = 5000

# Find Factor pairs of n
def FactorPairs(value):
  if value < 1:
    return []
    # Append 1 × n factor pair
  factors = [[1, value]]
  # Append n × 1 factor pair
  factors.append([value,1])
  for i in xrange(2, int(math.sqrt(value))+1):
    if value % i == 0:
      #Append a × b factor pair
      factors.append([i, value / i])
      #If a ≠ b, append b × a factor pair
      if i != (value/i):
        factors.append([value/i, i])
  return factors

# define array to sort and print for later
inadmissible = []

# Known Inadmissible Values:
for n in xrange(0,EndValue):
```

```
  FP = FactorPairs(n)
  for s in xrange(0,len(FP)):
    for t in xrange(0,len(FP)):
      for u in xrange(0,len(FP)):
        for v in xrange(0,len(FP)):
          a = FP[s][0]
          b = FP[s][1]
          c = FP[t][0]
          d = FP[t][1]
          f = FP[u][0]
          g = FP[u][1]
          h = FP[v][0]
          j = FP[v][1]
          if n not in inadmissible:
            # Inadmissible Values from Theorem 4.3
            if ((a < c) and (c < f) and (g < d) and (d < b)):
              if ((g*int(math.floor(d/float(g))) < d) and
                  (g*int(math.ceil(d/float(g))) >= b)):
                    inadmissible.append(n)
            # Inadmissible Values from Theorem 4.6
            if ((a < c) and (c < f) and (g < d) and (d < b) and (b%d !=0)):
              if ((d*int(math.floor(b/float(d))) <
                  g*int(math.floor(b/float(g)))) and
                  (g*int(math.floor(b/float(g))) <
                  d*int(math.ceil(b/float(d)))) and
                  (d*int(math.ceil(b/float(d))) <=
                  g*int(math.ceil(b/float(g))))):
                    inadmissible.append(n)
            # Inadmissible Values from Theorem 4.9
            if ((h < a) and (a < c) and (c < f) and (g < d) and (d < b) and
                (b < j) and (d%g == 0) and (a%h == 0) and (f%c == 0) and
                (j%b == 0) and (h > c-a) and (g > b-d)):
              if (((b-g*math.floor(b/float(g)))*(h*math.ceil(c/float(h))-c))
                  < ((2*a-h*math.ceil(c/float(h)))*
                     (g*math.ceil(b/float(g))-b))):
                    inadmissible.append(n)

# Sort and Remove Duplicates
def RemoveDuplicates(list):
  list.sort()
  last = list[-1]
  for i in range(len(list)-2, -1, -1):
      if last == list[i]:
          del list[i]
      else:
```

```
            last = list[i]

f = open('InadmissibleValues.txt','w')

# Sort and print that list
if(len(inadmissible) != 0):
  RemoveDuplicates(inadmissible)
  f.write(str(inadmissible))
  f.write('\n')
  print inadmissible
  f.write('number of inadmissible values:')
  f.write(str(len(inadmissible)))
  print 'number inadmissible:', len(inadmissible)
else:
  print 'No inadmissible values.'
  f.write('No inadmissible values.')
```

Appendix B
Inadmissible Values

The first numbers below $5,000$ that are inadmissible by the above code: {12, 20, 24, 28, 30, 36, 40, 42, 44, 48, 56, 60, 63, 66, 70, 72, 76, 80, 84, 88, 90, 92, 96, 99, 100, 102, 104, 105, 108, 110, 112, 120, 124, 126, 130, 132, 135, 138, 140, 144, 150, 152, 153, 154, 156, 160, 165, 168, 170, 172, 174, 176, 180, 182, 184, 188, 189, 190, 192, 195, 196, 198, 200, 204, 208, 210, 216, 220, 224, 225, 228, 230, 232, 234, 236, 238, 240, 246, 248, 252, 255, 260, 264, 266, 268, 270, 272, 273, 276, 280, 282, 284, 285, 286, 288, 290, 294, 296, 297, 300, 304, 306, 308, 312, 315, 316, 318, 320, 322, 324, 330, 332, 336, 340, 342, 344, 348, 350, 351, 352, 354, 357, 360, 364, 368, 370, 372, 374, 376, 378, 380, 384, 385, 387, 390, 392, 396, 399, 400, 405, 408, 412, 414, 416, 418, 420, 424, 426, 428, 429, 430, 432, 435, 440, 441, 442, 444, 448, 450, 455, 456, 459, 460, 462, 464, 468, 470, 472, 476, 477, 480, 483, 484, 488, 490, 492, 494, 495, 496, 498, 500, 504, 506, 508, 510, 513, 516, 518, 520, 522, 524, 525, 528, 530, 532, 534, 536, 540, 544, 546, 549, 550, 552, 556, 558, 560, 564, 567, 568, 570, 572, 574, 575, 576, 580, 585, 588, 590, 592, 594, 598, 600, 604, 606, 608, 612, 615, 616, 620, 621, 624, 627, 630, 632, 636, 638, 639, 640, 642, 644, 645, 646, 648, 650, 652, 656, 658, 660, 663, 664, 668, 670, 672, 675, 678, 680, 682, 684, 688, 690, 693, 696, 700, 702, 704, 708, 711, 714, 715, 716, 720, 725, 726, 728, 730, 732, 735, 736, 738, 740, 741, 742, 744, 748, 750, 752, 754, 756, 759, 760, 764, 765, 768, 770, 774, 777, 780, 782, 783, 784, 786, 790, 792, 796, 798, 800, 801, 804, 806, 808, 810, 812, 814, 816, 819, 820, 822, 824, 825, 828, 830, 832, 836, 837, 840, 844, 846, 848, 850, 852, 854, 855, 856, 858, 860, 861, 864, 868, 870, 872, 873, 874, 876, 880, 882, 884, 885, 888, 890, 891, 892, 894, 896, 897, 900, 902, 908, 910, 912, 918, 920, 924, 928, 930, 936, 938, 940, 944, 945, 946, 948, 950, 952, 954, 956, 957, 960, 962, 963, 966, 968, 969, 970, 972, 975, 976, 980, 984, 986, 987, 988, 990, 992, 996, 1000, 1001, 1002, 1004, 1008, 1012, 1015, 1016, 1020, 1023, 1026, 1030, 1032, 1035, 1036, 1038, 1040, 1044, 1048, 1050, 1052, 1053, 1054, 1056, 1060, 1062, 1064, 1065, 1066, 1068, 1070, 1071, 1072, 1074, 1078, 1080, 1084, 1085, 1088, 1089, 1090, 1092, 1095, 1098, 1100, 1102, 1104, 1105, 1106, 1110, 1112, 1116, 1118, 1120, 1122, 1125, 1128, 1130, 1131, 1132, 1134, 1136, 1140, 1144, 1146, 1148, 1150, 1152, 1155, 1160, 1161, 1162, 1164, 1166, 1168, 1170, 1173, 1175, 1176, 1178, 1180, 1182, 1184, 1185, 1188, 1190, 1192, 1196, 1197, 1200, 1204, 1206, 1208, 1209, 1210, 1212, 1215, 1216, 1218, 1220, 1221, 1222, 1224, 1225, 1228, 1230, 1232, 1236, 1239, 1240, 1242, 1244, 1246, 1248, 1254, 1256, 1258, 1260, 1264, 1269, 1270, 1272, 1274, 1275, 1276, 1278, 1280, 1281, 1284, 1287, 1288, 1290, 1292, 1295, 1296, 1298, 1300, 1302, 1304, 1305, 1308, 1311, 1312, 1316, 1320, 1323, 1324, 1326, 1328, 1330, 1332, 1334, 1335, 1336, 1340, 1342, 1344, 1350, 1352, 1353, 1356, 1358, 1359, 1360, 1362, 1364, 1365, 1368, 1370, 1372, 1376, 1377, 1380, 1384, 1386, 1388, 1390, 1392, 1394, 1395, 1398, 1400, 1404, 1406, 1408, 1410, 1416, 1419, 1420, 1422, 1424, 1425, 1426, 1428, 1430, 1431, 1432, 1434, 1435, 1436, 1440, 1442, 1443, 1444, 1448, 1449, 1450, 1452, 1456, 1460, 1462, 1464, 1468, 1470, 1472, 1476, 1479, 1480, 1482, 1484, 1485, 1488, 1490, 1494, 1496, 1498, 1500, 1504, 1506, 1508, 1512, 1515, 1516, 1518, 1520, 1521, 1524, 1526, 1528, 1530, 1532, 1534, 1536, 1539, 1540, 1542, 1545, 1547, 1548, 1550, 1554,

1558, 1560, 1564, 1566, 1568, 1570, 1572, 1575, 1576, 1578, 1580, 1581, 1584, 1586, 1590,
1592, 1596, 1598, 1599, 1600, 1602, 1606, 1608, 1610, 1611, 1612, 1614, 1615, 1616, 1617,
1620, 1624, 1625, 1628, 1630, 1632, 1634, 1635, 1638, 1640, 1644, 1645, 1647, 1648, 1650,
1652, 1656, 1659, 1660, 1664, 1665, 1666, 1668, 1670, 1672, 1674, 1676, 1677, 1680, 1683,
1686, 1688, 1690, 1692, 1694, 1696, 1700, 1701, 1702, 1704, 1705, 1708, 1710, 1712, 1716,
1720, 1722, 1724, 1725, 1728, 1729, 1730, 1734, 1736, 1738, 1740, 1742, 1743, 1744, 1746,
1748, 1749, 1750, 1752, 1755, 1756, 1758, 1760, 1764, 1768, 1770, 1772, 1773, 1775, 1776,
1780, 1782, 1784, 1785, 1786, 1788, 1790, 1792, 1794, 1798, 1800, 1802, 1804, 1806, 1812,
1815, 1816, 1818, 1820, 1824, 1825, 1826, 1827, 1830, 1832, 1833, 1834, 1836, 1840, 1845,
1846, 1848, 1850, 1852, 1854, 1856, 1860, 1862, 1863, 1866, 1868, 1869, 1870, 1872, 1876,
1880, 1881, 1884, 1886, 1887, 1888, 1890, 1892, 1896, 1898, 1900, 1902, 1904, 1908, 1911,
1912, 1914, 1916, 1917, 1918, 1920, 1924, 1925, 1926, 1930, 1932, 1935, 1936, 1938, 1940,
1944, 1946, 1947, 1948, 1950, 1952, 1953, 1956, 1960, 1964, 1965, 1968, 1970, 1971, 1972,
1974, 1976, 1978, 1980, 1984, 1988, 1989, 1990, 1992, 1995, 1996, 1998, 2000, 2001, 2002,
2004, 2006, 2007, 2008, 2010, 2012, 2013, 2014, 2016, 2020, 2024, 2025, 2028, 2030, 2032,
2034, 2037, 2040, 2044, 2046, 2050, 2052, 2058, 2060, 2064, 2068, 2070, 2072, 2074, 2076,
2079, 2080, 2082, 2085, 2086, 2088, 2090, 2091, 2092, 2093, 2096, 2097, 2100, 2104, 2106,
2107, 2108, 2109, 2112, 2114, 2115, 2116, 2118, 2120, 2121, 2124, 2128, 2130, 2132, 2133,
2134, 2135, 2136, 2139, 2140, 2142, 2144, 2145, 2146, 2148, 2150, 2152, 2154, 2156, 2160,
2162, 2163, 2168, 2169, 2170, 2172, 2175, 2176, 2178, 2180, 2184, 2185, 2188, 2190, 2192,
2193, 2196, 2200, 2204, 2205, 2208, 2210, 2212, 2214, 2216, 2220, 2222, 2223, 2224, 2226,
2230, 2232, 2233, 2235, 2236, 2240, 2242, 2244, 2250, 2252, 2254, 2255, 2256, 2259, 2260,
2261, 2262, 2264, 2266, 2268, 2270, 2272, 2275, 2277, 2278, 2280, 2282, 2284, 2288, 2290,
2292, 2294, 2295, 2296, 2298, 2300, 2303, 2304, 2310, 2314, 2316, 2318, 2320, 2322, 2324,
2325, 2328, 2330, 2331, 2332, 2334, 2336, 2337, 2338, 2340, 2344, 2345, 2346, 2348, 2349,
2350, 2352, 2354, 2356, 2358, 2360, 2364, 2365, 2366, 2368, 2370, 2376, 2378, 2379, 2380,
2384, 2385, 2387, 2388, 2390, 2392, 2394, 2396, 2397, 2398, 2400, 2403, 2405, 2406, 2408,
2412, 2415, 2416, 2418, 2420, 2421, 2422, 2424, 2425, 2428, 2430, 2431, 2432, 2436, 2438,
2440, 2442, 2444, 2445, 2448, 2450, 2451, 2456, 2457, 2460, 2464, 2466, 2470, 2472, 2475,
2476, 2478, 2480, 2484, 2488, 2490, 2492, 2493, 2494, 2496, 2499, 2500, 2502, 2506, 2508,
2511, 2512, 2514, 2516, 2520, 2522, 2524, 2528, 2530, 2532, 2534, 2535, 2536, 2538, 2540,
2541, 2542, 2544, 2546, 2548, 2550, 2552, 2553, 2556, 2560, 2562, 2565, 2568, 2570, 2572,
2574, 2576, 2580, 2583, 2584, 2585, 2586, 2588, 2590, 2592, 2596, 2597, 2600, 2601, 2604,
2608, 2610, 2613, 2616, 2618, 2619, 2620, 2622, 2624, 2625, 2626, 2628, 2630, 2632, 2635,
2636, 2639, 2640, 2646, 2648, 2650, 2652, 2655, 2656, 2658, 2660, 2664, 2666, 2668, 2670,
2672, 2673, 2674, 2676, 2678, 2679, 2680, 2682, 2684, 2685, 2686, 2688, 2690, 2691, 2694,
2695, 2697, 2698, 2700, 2702, 2703, 2704, 2706, 2709, 2712, 2714, 2716, 2717, 2718, 2720,
2724, 2726, 2727, 2728, 2730, 2732, 2736, 2740, 2744, 2745, 2748, 2750, 2751, 2752, 2754,
2756, 2760, 2764, 2766, 2768, 2769, 2770, 2772, 2774, 2775, 2776, 2780, 2781, 2784, 2788,
2790, 2792, 2793, 2794, 2796, 2800, 2802, 2805, 2806, 2808, 2812, 2814, 2816, 2817, 2820,
2821, 2822, 2826, 2828, 2829, 2830, 2832, 2835, 2838, 2840, 2842, 2844, 2847, 2848, 2850,
2852, 2856, 2860, 2862, 2864, 2865, 2868, 2870, 2871, 2872, 2874, 2875, 2876, 2880, 2882,
2884, 2886, 2888, 2889, 2890, 2892, 2895, 2896, 2898, 2900, 2904, 2905, 2907, 2908, 2910,
2912, 2914, 2915, 2916, 2919, 2920, 2924, 2925, 2926, 2928, 2930, 2936, 2937, 2938, 2940,
2944, 2945, 2946, 2948, 2950, 2952, 2956, 2958, 2960, 2961, 2964, 2967, 2968, 2970, 2972,

2975, 2976, 2979, 2980, 2982, 2984, 2985, 2988, 2990, 2992, 2996, 2997, 3000, 3003, 3004,
3006, 3008, 3009, 3010, 3012, 3016, 3018, 3020, 3021, 3024, 3025, 3026, 3030, 3032, 3034,
3036, 3038, 3040, 3042, 3045, 3048, 3050, 3052, 3054, 3056, 3058, 3059, 3060, 3064, 3066,
3068, 3069, 3070, 3072, 3074, 3075, 3078, 3080, 3082, 3084, 3087, 3090, 3094, 3096, 3100,
3102, 3105, 3108, 3111, 3112, 3114, 3116, 3120, 3122, 3124, 3126, 3128, 3130, 3132, 3135,
3136, 3140, 3141, 3144, 3146, 3148, 3150, 3152, 3156, 3159, 3160, 3162, 3164, 3168, 3170,
3172, 3174, 3176, 3180, 3182, 3184, 3185, 3186, 3190, 3192, 3195, 3196, 3198, 3200, 3201,
3204, 3206, 3210, 3212, 3213, 3216, 3219, 3220, 3222, 3224, 3225, 3228, 3230, 3231, 3232,
3234, 3240, 3243, 3244, 3245, 3248, 3250, 3252, 3255, 3256, 3260, 3262, 3264, 3266, 3267,
3268, 3270, 3276, 3278, 3280, 3285, 3286, 3288, 3289, 3290, 3292, 3294, 3296, 3298, 3300,
3302, 3303, 3304, 3306, 3308, 3311, 3312, 3315, 3318, 3320, 3322, 3324, 3325, 3328, 3330,
3332, 3336, 3339, 3340, 3342, 3344, 3345, 3348, 3350, 3352, 3354, 3355, 3356, 3358, 3360,
3363, 3366, 3367, 3368, 3370, 3372, 3375, 3376, 3378, 3380, 3381, 3382, 3384, 3388, 3390,
3392, 3393, 3395, 3396, 3400, 3402, 3404, 3408, 3410, 3414, 3416, 3417, 3420, 3422, 3423,
3424, 3429, 3430, 3432, 3434, 3435, 3436, 3438, 3440, 3441, 3444, 3445, 3448, 3450, 3451,
3452, 3456, 3458, 3460, 3465, 3468, 3470, 3471, 3472, 3474, 3476, 3477, 3478, 3480, 3483,
3484, 3486, 3488, 3490, 3492, 3496, 3498, 3500, 3504, 3507, 3510, 3512, 3514, 3515, 3516,
3519, 3520, 3522, 3525, 3526, 3528, 3530, 3531, 3532, 3534, 3535, 3536, 3537, 3538, 3540,
3542, 3544, 3546, 3548, 3549, 3550, 3552, 3553, 3555, 3556, 3558, 3560, 3562, 3564, 3565,
3567, 3568, 3570, 3572, 3575, 3576, 3580, 3584, 3585, 3586, 3588, 3590, 3591, 3594, 3596,
3597, 3598, 3600, 3604, 3605, 3608, 3610, 3612, 3614, 3616, 3618, 3620, 3624, 3625, 3626,
3627, 3628, 3630, 3632, 3633, 3634, 3636, 3640, 3644, 3645, 3648, 3650, 3652, 3654, 3657,
3658, 3660, 3663, 3664, 3666, 3668, 3670, 3672, 3674, 3675, 3676, 3680, 3682, 3684, 3686,
3688, 3689, 3690, 3692, 3696, 3700, 3702, 3704, 3705, 3708, 3710, 3712, 3717, 3718, 3720,
3724, 3725, 3726, 3728, 3730, 3732, 3735, 3736, 3738, 3740, 3741, 3744, 3750, 3752, 3756,
3760, 3762, 3765, 3768, 3770, 3772, 3774, 3776, 3780, 3782, 3784, 3788, 3789, 3790, 3792,
3794, 3795, 3796, 3798, 3800, 3801, 3804, 3806, 3807, 3808, 3810, 3813, 3816, 3818, 3819,
3820, 3822, 3824, 3825, 3828, 3830, 3832, 3834, 3835, 3836, 3838, 3840, 3842, 3843, 3844,
3846, 3848, 3850, 3852, 3854, 3857, 3860, 3861, 3864, 3868, 3870, 3872, 3874, 3875, 3876,
3878, 3879, 3880, 3882, 3884, 3885, 3886, 3888, 3890, 3892, 3894, 3895, 3896, 3900, 3904,
3906, 3910, 3912, 3913, 3914, 3915, 3916, 3918, 3920, 3922, 3924, 3926, 3927, 3928, 3930,
3932, 3933, 3936, 3939, 3940, 3942, 3944, 3948, 3950, 3951, 3952, 3954, 3956, 3960, 3964,
3965, 3968, 3969, 3970, 3972, 3975, 3976, 3978, 3980, 3984, 3990, 3992, 3996, 3999, 4000,
4002, 4004, 4005, 4008, 4012, 4014, 4016, 4017, 4018, 4020, 4024, 4025, 4026, 4028, 4030,
4032, 4035, 4040, 4041, 4042, 4044, 4046, 4047, 4048, 4050, 4056, 4059, 4060, 4062, 4064,
4066, 4068, 4070, 4071, 4072, 4074, 4076, 4077, 4080, 4081, 4086, 4088, 4090, 4092, 4094,
4095, 4098, 4100, 4102, 4104, 4108, 4110, 4113, 4114, 4116, 4118, 4120, 4122, 4123, 4124,
4125, 4128, 4130, 4131, 4134, 4136, 4140, 4142, 4144, 4147, 4148, 4150, 4152, 4154, 4156,
4158, 4160, 4161, 4164, 4165, 4170, 4172, 4173, 4176, 4180, 4182, 4184, 4185, 4186, 4188,
4190, 4191, 4192, 4194, 4199, 4200, 4202, 4203, 4204, 4206, 4208, 4209, 4212, 4214, 4215,
4216, 4218, 4220, 4221, 4224, 4228, 4230, 4232, 4233, 4234, 4235, 4236, 4238, 4239, 4240,
4242, 4245, 4246, 4248, 4250, 4251, 4252, 4256, 4257, 4260, 4263, 4264, 4266, 4268, 4270,
4272, 4275, 4278, 4280, 4284, 4288, 4290, 4292, 4293, 4294, 4296, 4298, 4300, 4301, 4302,
4304, 4305, 4308, 4312, 4314, 4316, 4318, 4320, 4323, 4324, 4325, 4326, 4328, 4329, 4330,
4332, 4334, 4335, 4336, 4338, 4340, 4342, 4344, 4346, 4347, 4348, 4350, 4352, 4355, 4356,

4360, 4364, 4365, 4366, 4368, 4370, 4375, 4376, 4380, 4382, 4384, 4386, 4389, 4390, 4392, 4396, 4400, 4402, 4404, 4407, 4408, 4410, 4412, 4416, 4420, 4422, 4424, 4425, 4428, 4430, 4432, 4433, 4437, 4438, 4440, 4444, 4446, 4448, 4450, 4452, 4454, 4455, 4456, 4458, 4460, 4464, 4465, 4466, 4470, 4472, 4473, 4476, 4480, 4482, 4484, 4485, 4488, 4490, 4492, 4494, 4495, 4496, 4498, 4500, 4504, 4508, 4510, 4512, 4514, 4515, 4518, 4520, 4522, 4524, 4526, 4527, 4528, 4530, 4532, 4536, 4539, 4540, 4544, 4545, 4548, 4550, 4551, 4554, 4556, 4557, 4558, 4560, 4563, 4564, 4566, 4568, 4570, 4572, 4575, 4576, 4578, 4580, 4582, 4584, 4585, 4588, 4590, 4592, 4596, 4598, 4599, 4600, 4602, 4604, 4606, 4608, 4611, 4615, 4617, 4620, 4623, 4624, 4626, 4628, 4630, 4632, 4634, 4635, 4636, 4638, 4640, 4641, 4642, 4644, 4648, 4650, 4652, 4653, 4654, 4655, 4656, 4660, 4662, 4664, 4665, 4668, 4670, 4672, 4674, 4675, 4676, 4680, 4683, 4684, 4686, 4688, 4689, 4690, 4692, 4695, 4696, 4698, 4700, 4704, 4706, 4708, 4710, 4712, 4715, 4716, 4719, 4720, 4725, 4728, 4730, 4732, 4734, 4736, 4740, 4743, 4745, 4746, 4748, 4750, 4752, 4753, 4756, 4758, 4760, 4761, 4764, 4767, 4768, 4770, 4773, 4774, 4776, 4779, 4780, 4782, 4784, 4785, 4788, 4790, 4792, 4794, 4795, 4796, 4797, 4800, 4806, 4807, 4809, 4810, 4812, 4814, 4815, 4816, 4818, 4820, 4824, 4826, 4828, 4830, 4832, 4833, 4836, 4838, 4840, 4842, 4844, 4845, 4848, 4850, 4851, 4854, 4856, 4858, 4860, 4862, 4864, 4865, 4870, 4872, 4875, 4876, 4880, 4884, 4886, 4887, 4888, 4890, 4892, 4896, 4898, 4899, 4900, 4902, 4904, 4905, 4908, 4912, 4914, 4920, 4921, 4922, 4923, 4924, 4925, 4926, 4928, 4929, 4930, 4932, 4935, 4940, 4941, 4944, 4945, 4947, 4950, 4952, 4953, 4956, 4958, 4959, 4960, 4962, 4964, 4966, 4968, 4970, 4972, 4975, 4976, 4977, 4978, 4980, 4982, 4983, 4984, 4986, 4988, 4990, 4992, 4994, 4995, 4998}

Appendix C
Backtracking Program

The following program is a backtracking algorithm written in Python for finding or solving a partial factor pair latin square of order n.

```
import math
import sys
import time

#Size of Factor Pair Latin Square
n = 9

#Define the empty Array of size n
grid = [[0 for x in xrange(n)] for y in xrange(n)]

#construct the Factor Pair Latin Square
#name the first row 1 ... n
for i in xrange(0,n):
  grid[0][i] = i+1
#Note:  You can enter a partially filled in FPLS by putting
#grid = [[#,#,#,...,#],[#,#,#,...,#],...,[#,#,#,...,#]]
#Any unfilled cell, make 0

#Find Factor pairs of n
def FactorPairs(value):
    if value < 1:
        return []
    #Append 1 x n factor pair
    factors = [[1, value]]
    #Append n x 1 factor pair
    factors.append([value,1])
    for i in xrange(2, int(math.sqrt(value))+1):
        if value % i == 0:
            #Append a x b factor pair
            factors.append([i, value / i])
            #If a != b, append b x a factor pair
            if i != (value/i):
              factors.append([value/i, i])
    return factors
```

```
#Declare Variables

#General Factor Pair Check (including Rows and columns)
#Requires a list to be passed to it.  i.e. array[]
def Checker(list):
  for i in xrange(0,n-1):
    for j in xrange(i+1,n):
      if list[i] != 0 and list[i] == list[j]:
        return False
  return True

#General Factor Pair Checker
def GeneralChecker(position):
  #Check the a × b squares
  #It should be noted that we only need to check the a × b squares,
  #since the b × a squares will be checked as the next (or previous)
  #factor pair in FactorPairs(n)
  number1 = FactorPairs(n)[position][0]
  number2 = FactorPairs(n)[position][1]
  row = []
  for level in xrange(0,n,number1):
    for section in xrange(0,n,number2):
      for i in xrange(level,level+number1):
        for j in xrange(section,section+number2):
          row.append(grid[i][j])
          if len(row) == n:
            if Checker(row) == False:
              return False
            row = []
  return True

def OverallCheck(list):
  for i in xrange(0,len(FactorPairs(n))):
    if GeneralChecker(i) == False:
      return False
  return True

#Define Positions that are unchangeable (i.e. Fixed.)
Forbidden = [[0 for x in xrange(n)] for y in xrange(n)]
for row in xrange(0,n):
  for column in xrange(0,n):
    if grid[row][column] != 0:
      Forbidden[row][column] = 1

#Solver
```

```
def Solve(row, column):
  #In case the cell passed to function is fixed, go to next non-fixed cell
  while (Forbidden[row][column] == 1):
    #Try next cell
    column = column + 1
    #Advance row if necessary
    if (column > (n-1)):
      column = 0
      row = row + 1
    if (row > (n-1)):
      return True

  #Once we have our cell coordinates, substitute in a number and check it
  for intGuess in xrange(1,n+1):
    intTryRow = 0
    intTryColumn = 0
    grid[row][column] = intGuess
    #Print each guess to see what's going on.
    for i in grid:
      print i
    print '------------'
    #If good, Solve the next one
    if(OverallCheck(grid)):
      #Try the next square, preserving current values
      intTryColumn = column + 1
      intTryRow = row
      #Check if the tow needs to be advanced
      if (intTryColumn > (n-1)):
        intTryColumn = 0
        intTryRow = intTryRow + 1
      if (intTryRow > (n-1)):
        return True
      #check if we're done
      if(Solve(intTryRow, intTryColumn)):
        return True
  #If none of the numbers we've checked are right, put this cell back to 0
  #and return False
  grid[row][column] = 0
  return False

t = time.clock()
Solve(0,0)
#Double Check
#print grid
for i in grid:
```

```
  print i
#Double check
print 'Double Check:'
if OverallCheck(grid) == False:
    print 'This IS NOT a Factor Pair Latin Square'
    sys.exit()
for i in xrange(0,n):
  for j in xrange(0,n):
    if grid[i][j] == 0:
      print 'This IS NOT a Factor Pair Latin Square'
      sys.exit()
print 'This IS INDEED a Factor Pair Latin Square!'
print 'Took %.3f seconds.' % (time.clock()-t)
```

Appendix D
DLX Code

The following program is a C++ implementation of Donald Knuth's DLX algorithm. It is a modified version of bbi5291's code posted on Computer Science Canada public forum. Expressed and written consent to use and edit this file has been given by the author.[2]

D.1 Header File

```
//The following code is based on the paper "Dancing Links" by D. E. Knuth.
//See http://www-cs-faculty.stanford.edu/~uno/papers/dancing-color.ps.gz
#ifndef DLX_H
#define DLX_H
#include <cstring>
#include <climits>
struct data_object //A module in the sparse matrix data structure.
{
  data_object* L;          //Link to next object left.
  data_object* R;         //          "          right.
  data_object* U;         //          "          up.
  data_object* D;         //          "          down.
  data_object* C;         //Link to column header.
  int x;                  //In a column header: number of ones
                          //in the column. Otherwise: row index.
  void cover()            //Covers a column.
  {
    data_object* i=D;
    data_object* j;
    R->L=L;
    L->R=R;
    while (i!=this)
    {
      j=i->R;
      while (j!=i)
      {
        j->D->U=j->U;
        j->U->D=j->D;
        j->C->x--;
        j=j->R;
      }
      i=i->D;
```

```
    }
  }
  void uncover()                  //Uncovers a column.
  {
    data_object* i=U;
    data_object* j;
    while (i!=this)
    {
      j=i->L;
      while (j!=i)
      {
        j->C->x++;
        j->D->U=j;
        j->U->D=j;
        j=j->L;
      }
      i=i->U;
    }
    R->L=this;
    L->R=this;
  }
};
//Standard S-heuristic suggested in Knuth's paper: pick the column with
//the fewest ones. Takes the root of the sparse matrix structure as an
//argument; returns a pointer to the column header with the fewest ones.
data_object* DLX_Knuth_S_heuristic(data_object* root)
{
  data_object* P=root->R;
  data_object* res;
  int best=INT_MAX/2;
  while (P!=root)
  {
    if (P->x<best)
    {
      best=P->x;
      res=P;
    }
    P=P->R;
  }
  return res;
}
template <typename Func1,typename Func2>
/*
Actual recursive function implementing Knuth's Dancing Links method.
h is the root of the sparse matrix structure.
```

```
O is the stack that will contain a list of rows used.
*/
void DLX_search(data_object* h,int k,int* O,Func1 send_row,
    Func2 choose_column)
{
  int i;
  data_object *r,*c,*j;
  if (h->R==h) //done - solution found
  {
    //send rows used in solution back...
    for (i=0; i<k; i++)
      send_row(O[i]);
    //-1 signifies end of solution
    send_row(-1);
    return;
  }
  //otherwise
  c=choose_column(h); //choose a column to cover
  c->cover();         //cover it
  r=c->D;
  while (r!=c)
  {
    O[k]=r->x;
    j=r->R;
    while (j!=r)
    {
      j->C->cover();
      j=j->R;
    }
    DLX_search(h,k+1,O,send_row,choose_column);
    //set r <- O[k], and c<- C[r], this is unnecessary
    j=r->L;
    while (j!=r)
    {
      j->C->uncover();
      j=j->L;
    }
    r=r->D;
  }
  c->uncover();
}
template <typename random_access_iterator,typename Func1,typename Func2>
/*
Meta-implementation of Knuth's Dancing Links method for finding
solutions tothe exact cover problem.
```

```
PARAMETERS:
int rows: Number of rows in the matrix.
int cols: Number of columns in the matrix.
random_access_iterator buf: A random access iterator to ints (either 0 or
1), the entries of the matrix, in row major order.
Func1 send_row: A function object with return type void which takes as a
parameter the index of a row in a solution to the problem. (e.g. store
it in a buffer or print it out) -1 signifies the end of a solution.
Func2 choose_column: A deterministic function object taking as a parameter
a data_object* (the root) and returning a data_object* (the header of the
column to choose.)
*/
void DLX_dancing_links(int rows,int cols,random_access_iterator buf,
      Func1 send_row,Func2 choose_column)
{
  //step 1: construct the linked-list structure.
  //We can do this by iterating through the rows and columns. Time is
  //linear in the number of entries (optimal).
  //Space used is linear in the number of columns + the number of rows
  // + the number of ones.
  int i,j;
  data_object* root=new data_object; //root
  data_object* P=root;               //left-right walker
  data_object* Q;                    //top-down walker
  //array of pointers to column headers
  data_object** walkers=new data_object*[cols];
  //auxiliary stack for recursion
  int* st=new int[rows];
  for (i=0; i<cols; i++)
  {
    //create a column header and L/R links
    (P->R=new data_object)->L=P;
    //store a pointer to the column header
    walkers[i]=Q=P=P->R;
    P->x=0; //reset popcount
    for (j=0; j<rows; j++)
      if (buf[i+cols*j]) //a 1 in the current location?
      {
        //create a data object and U/D links
        (Q->D=new data_object)->U=Q;
        Q=Q->D;  //advance pointer
        Q->C=P;  //link to the column header
        P->x++;  //increment popcount for this column
        Q->x=j;  //note the row number of this entry
```

```
    }
  Q->D=P; //complete the column
  P->U=Q;
}
P->R=root; //complete the column list
root->L=P;
//eliminate empty columns
P=root;
for (i=0; i<cols; i++)
{
  P=P->R;
  if (!P->x)
  {
    P->L->R=P->R;
    P->R->L=P->L;
  }
}
//now construct the L/R links for the data objects.
P=new data_object;
for (i=0; i<rows; i++)
{
  Q=P;
  for (j=0; j<cols; j++)
    if (buf[j+cols*i]) //a one
    {
      //in _this_ row...
      walkers[j]=walkers[j]->D;
      //create L/R links
      (Q->R=walkers[j])->L=Q;
      //advance pointer
      Q=Q->R;
    }
  if (Q==P) continue;
  Q->R=P->R;        //link it to the first one in this row.
  P->R->L=Q;        //link the first one to the last one.
}
delete P;                 //P is no longer needed
delete walkers;           //walkers are no longer needed
//step 2: recursive algorithm
DLX_search(root,0,st,send_row,choose_column);
delete st;
P=root->R;
while (P!=root)           //deallocate sparse matrix structure
{
  Q=P->D;
```

```
    while (Q!=P)
    {
      Q=Q->D;
      delete Q->U;
    }
    P=P->R;
    delete P->L;
  }
  delete root;
}

//If no heuristic is specified, Knuth's S heuristic is used - select the
//column with the fewest ones to minimize the breadth of the search tree.
template <typename random_access_iterator,typename Func1>
void DLX_dancing_links(int rows,int cols,random_access_iterator buf,
                       Func1 send_row)
{
  DLX_dancing_links(rows,cols,buf,send_row,DLX_Knuth_S_heuristic);
}

#endif
```

D.2 CPP File

```
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <cstdlib>
#include <stdlib.h>
#include "dlx.h"
using namespace std;

#define block(r,c,i) (FParray[i][0]*((r)/FParray[i][0])+\
                      ((c)/FParray[i][1]))

// Define the order of the FPLS(n)
#define N 23
const int intCount = 2; //Number of Factors (Factor Pairs)
const int Columns = 1587; //N*N*(intCount+1)
const int Rows = 12167; //N*N*N

int matrix[Rows][Columns]; //DLX Matrix

int grid[N][N]; //End Product
```

```
// Number of Factor Pairs = Number of factors
int FactorCount(int n)
{
  int factorCount;
  factorCount = 0;

  for (int i = 1; i < n+1; i++)
  {
    if (n % i == 0)
      factorCount++;
  }

  return factorCount;
}

void f(int x)
{
  int i;
  int c;
  int r;
  if (x+1)
  {
    i=x%N; x/=N;
    c=x%N; r=x/N;
    grid[r][c]=i+1;
  }
  else
  {
    for (r=0; r<N; r++,putchar('\n'))
      for (c=0; c<N; c++)
        {
          cout << grid[r][c];
          if (c != N-1)
            cout << " & ";
          if (c == N-1)
            cout << " \\nl";
        }
    printf("\n");
    exit(0);
  }
}

void cover_col(int col)
{
  for (int row=0; row<(N*N)*N; row++)
```

```
    if (matrix[row][col])
    {
      matrix[row][col]=0;
      memset(matrix[row],0,sizeof(matrix[row]));
    }
}

void cover_row(int row)
{
  for (int col=0; col<(N*N)*(intCount+1); col++)
    if (matrix[row][col])
    {
      matrix[row][col]=0;
      cover_col(col);
    }
}
int main()
{
  // Calculates the factor Pairs.
  int FParray[intCount][2];
  int whichPair;
  whichPair = 1;

  FParray[0][0] = 1;
  FParray[0][1] = N;
  FParray[1][0] = N;
  FParray[1][1] = 1;

  for (int i = 2; i < (floor(sqrt(N))+1); i++)
  {
    if (N % i == 0)
    {
      whichPair++;
      FParray[whichPair][0] = i;
      FParray[whichPair][1] = N/i;
      if (i != (N/i))
      {
        whichPair++;
        FParray[whichPair][0] = N/i;
        FParray[whichPair][1] = i;
      }
    }
  }

  for (int row=0,r=0; r<N; r++)
```

```
    for (int c=0; c<N; c++)
      for (int i=0; i<N; i++,row++)
      {
        //uniqueness constraint
        matrix[row][r+N*c]=1;
        //Factor Pair constraint (including Row & column)
        for (int pair = 0; pair < intCount; pair++)
        {
          matrix[row][(pair+1)*(N*N)+i+N*(block(r,c,pair))] = 1;
        }
      }
  putchar('\n');
  DLX_dancing_links((N*N)*N,(N*N)*(intCount+1),(int*)&matrix,f);
  return 0;
}
```