Erasure-Correcting Codes Derived From Sudoku & Related Combinatorial Structures

Linzy A. Phillips

Mathematics and Statistics

University of Glamorgan

A submission presented in
partial fulfilment of the requirements
of the University of Glamorgan/Prifysgol Morgannwg
for the degree of Doctor of Philosophy.

February 21, 2013

Acknowledgments

My thanks must first be expressed to my Director of Studies, Dr. Stephanie Perkins, and Supervisors, Dr. Paul Roach and Prof. Derek Smith for their continued encouragement and enthusiasm throughout the research process, along with the personal and professional support offered by each of them throughout. A combination of great support, teaching, ideas and crucially great company made this research team a pleasure to be a part of - my gratitude to my supervisory team cannot be overstated.

My gratitude must also be extended to the Division of Mathematics and Statistics for supporting and encouraging my development, not only as a researcher, but as a lecturer also. The opportunity to lecture across a range of subjects is one I have particularly enjoyed, and will remember fondly.

On a personal note, my thanks to my lovely family - my parents Karan and Paul, grandparents Annette, Ron, Rita and Tegwyn and brother Richard have been a continuous source of advice, support and encouragement, throughout the writing of this thesis and always. My final thanks to my fellow post-graduates for their continuous understanding and moral support, and to one in particular, who kept me smiling throughout.

Abstract

This thesis presents the results of an investigation into the use of puzzle-based combinatorial structures for erasure correction purposes. The research encompasses two main combinatorial structures: the well-known number placement puzzle Sudoku and a novel three component construction designed specifically with puzzle-based erasure correction in mind. The thesis describes the construction of outline erasure correction schemes incorporating each of the two structures.

The research identifies that both of the structures contain a number of smaller sub-structures, the removal of which results in a grid with more than one potential solution - a detrimental property for erasure correction purposes. Extensive investigation into the properties of these sub-structures is carried out for each of the two outline erasure correction schemes, and results are determined that indicate that, although the schemes are theoretically feasible, the prevalence of sub-structures results in practically infeasible schemes.

The thesis presents detailed classifications for the different cases of sub-structures observed in each of the outline erasure correction schemes. The anticipated similarities in the sub-structures of Sudoku and sub-structures of Latin Squares, an established area of combinatorial research, are observed and investigated, the proportion of Sudoku puzzles free of small sub-structures is calculated and a simulation comparing the recovery rates of small sub-structure free Sudoku and standard Sudoku is carried out. The analysis of sub-structures for the second erasure correction scheme involves detailed classification of a variety of small sub-structures; the thesis also derives probabilistic lower bounds for the expected numbers of case-specific sub-structures within the puzzle structure, indicating that specific types of sub-structure hinder recovery to such an extent that the scheme is infeasible for practical erasure correction.

The consequences of complex cell inter-relationships and wider issues with puzzle-based erasure correction, beyond the structures investigated in the thesis are also discussed, concluding that while there are suggestions in the literature that Sudoku and other puzzle-based combinatorial structures may be useful for erasure correction, the work of this thesis suggests that this is not the case.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Erasure-Correcting Codes

An *erasure-correcting code* is a code that can be used to correct erasure errors that occur in a message during transmission. Generally, an erasure code transforms an encoded message of k symbols into an encoded message of n symbols, where $n > k$, by the addition of $n - k$ redundant symbols to the original encoded message. The addition of the redundant symbols is such that it allows the original message to be recovered from a corrupted message provided a sufficiently large subset of the n transmitted symbols are received [1].

Erasure-correcting codes have many practical applications within the field of computer science, as they eliminate the need for continuous two-way communication across a communication channel. Many schemes rely on the receiver making explicit requests for missing data to be transmitted again if the received message cannot be decoded; this process then continues until all message blocks have been decoded successfully. Employing an erasure code, and transmitting its $n-k$ symbols of redundant information in this instance, presents the receiver with adequate information to reconstruct the original message without the need for interaction with the transmitter. Erasure codes are already employed in many areas of computer science and communication theory, such as satellite communication [30] and within internet transport protocols [21], and recent research [26, 30] has identified further areas in which there exists scope to incorporate such codes as opposed to more traditional methods. Many new erasure codes are being designed to fit these purposes. Further applications of these codes are discussed in more detail in Section 2.4 of the thesis.

Erasure-correcting codes are generally designed for use over an *erasure channel.* Erasure channels are channels within which transmitted symbols that are corrupted or erased by noise present as 'error' symbols, as opposed to presenting as different symbols of the input alphabet. In a sense, erasure channels are error-free as all input symbols that are received are known to be correct; it is only the erased symbols, that are replaced by 'error' symbols, which require recovery to determine the original transmitted message [37]. The simplest example of an erasure channel is the ***binary erasure channel*** (BEC), an erasure channel with an input alphabet A of only two symbols, $A = \{0, 1\}$. A string of symbols $a_i \in A$ is transmitted over the BEC; the output of this string is from an output alphabet $B = \{0, 1, e\}$ where e represents an erased symbol (either 0 or 1). Each input symbol is correctly transmitted over the channel with probability p and is erased with probability $\bar{p} = 1 - p$ [12]. This is demonstrated in Figure 1.1.

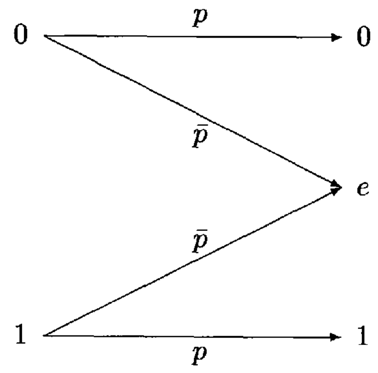


Figure 1.1: *The binary erasure channel (BEC) [12].*

Other non-binary erasure channels act very similarly to the BEC, the most notable difference being the variation in the size of the input (and hence output) alphabet.

The rate of a code is a measure that states what portion of the transmitted information is non-redundant, that is what portion of the transmitted information corresponds to the original message. Generally, the rate R of any non-empty code is given by;

$$R = \frac{\log_r |C|}{n},$$

where r represents the number of symbols of the input alphabet A, $|C|$ represents the total number of codewords of the code and n represents the length of the code [12].

Named after Claude Shannon – the mathematician credited with founding the field of information theory – the *Shannon capacity* of a channel is a fundamental concept in coding theory. The *Shannon Capacity* of a noisy channel is defined to be the theoretical maximum amount of information that can be reliably transmitted over the channel. Shannon also showed that for any rate R less than the channel capacity S there exists a coding and decoding scheme with arbitrarily low error rates for sufficiently large n. Conversely, for $R \geq S$, error rates are increased and information cannot reliably be recovered. Hence when constructing a coding scheme, it is desirable for the rate of the code, R, to be as close to the Shannon capacity as possible [12]. The Shannon capacity can be formally defined in terms of mutual information [12], this precise definition is given later in Section 3.3.1.1 of the thesis.

1.2 Combinatorial Structures

1.2.1 Introduction to Latin Squares

A square array of order n is referred to as a Latin square if each row and each column of the array contains each of the symbols $1, 2, \ldots, n$ exactly once. An example of such a structure is shown in Figure 1.2.

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

Figure 1.2: *A Latin square of order 7.*

Latin squares were first systematically studied by the Swiss mathematician Leonhard Euler in 1783, who also introduced the concept of mutual-orthogonality between the structures [10]. Two Latin squares, L_1 and L_2, are said to be orthogonal if each ordered pair of corresponding elements occurs exactly once, where pairs are obtained by combining the elements of the same cell for each of the Latin squares L_1 and L_2 [16]. Euler attempted to utilise mutually-orthogonal Latin squares (MOLS) to solve the famous 'Thirty-Six Officers Problem', the problem of arranging six regiments each consisting of six officers of different rank in a 6×6 square so that no rank or regiment is repeated in any row or column. This problem is equivalent to identifying two mutually-orthogonal Latin squares. A possible approach to solving the famous problem is to label each of the ranks with the values $\{1, 2, \ldots, 6\}$ and each of the regiments with the symbols $\{A, B, \ldots, F\}$ and to construct two order six Latin squares, one with each set of values such that when the two structures are superimposed, each number-letter pairing occurs exactly once [10]. Euler conjectured, and it has since been proved by systematic enumeration (by Gaston Tarry, in 1901 [36]), that no solution exists for the problem and that therefore no pair of order six mutually-orthogonal Latin squares exist.

Latin squares have been extensively studied since Euler's introduction and remain an active area of mathematical research, with many open problems to resolve. The structures have important applications, and appear in a variety of mathematical disciplines including combinatorics and experimental design in statistics. Further applications and important results are discussed in

Section 2.3.1 of the literature review.

Since its introduction many centuries ago, the Latin square structure has also been developed into a number-placement puzzle [9], the object of which is to complete a partially filled Latin square. The popular Sudoku puzzle is an example of a further constrained Latin square puzzle, and many Sudoku-variant puzzles have been developed in recent years.

1.2.2 Introduction to Sudoku

The first published Sudoku puzzle appeared in May 1979 in Dell Magazine's issue of "Dell Pencil Puzzles & Word Games" [19]. A freelance puzzle creator, Howard Garns is attributed with its creation [3]. The puzzle was not named Sudoku however until published by the Japanese puzzle company "Nikoli" in 1986, the English translation of Sudoku being *single number* [35]. Sudoku has become increasingly popular, on a global scale, in recent years with more and more newspapers and puzzle books publishing the puzzle. Its popularity in the United Kingdom was established in 2005 following its first UK publication in The Daily Telegraph newspaper [35].

A Sudoku grid is a square grid of order n. The grid is a Latin square with one further property; the grid is subdivided into n non-overlapping minigrids and each of the values 1 to n must appear exactly once within each of these minigrids, in addition to occurring exactly once in each row and each column. Typical published puzzles use $n = 9$ whereby the grid is subdivided into 9 non-overlapping minigrids of order 3; hence the values $1, 2, \ldots, 9$ may occur exactly once in each row, each column and each 3×3 minigrid. A Sudoku puzzle is created by removing values from some cells of a valid Sudoku grid, such that the grid can be completed uniquely and the object of the puzzle is for the player to deduce the unique completion by using the structure's properties and logical reasoning. An example of a Sudoku puzzle with its unique solution is shown in Figure 1.3.

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

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

Figure 1.3: *An example of a Sudoku puzzle and its unique solution [27].*

Although originally introduced as a number-placement puzzle, Sudoku is also now an established combinatorial object and the study of the structure is a popular area of combinatorial research; research varies from more pure questions of existence and enumeration for various orders [6], to the investigation of the potential applications of the structure within other disciplines [32] – some of which shall be discussed in subsequent chapters of this thesis.

1.3 Research Aims

The aim of the work described in this thesis is to examine and evaluate the potential of puzzle-based combinatorial structures to provide features useful for the construction of erasure correction schemes. Following the suggestion of Sudoku as a useful structure for this purpose [32], this puzzle is considered first. A further structure is subsequently examined following the establishment of the poor performance of Sudoku-derived codes. Evaluation of the issues arising from both puzzle-based erasure correction schemes is presented throughout the thesis.

The notation for Sudoku is presented here, and other puzzle-specific notation is introduced where it becomes relevant in subsequent chapters.

1.4 Sudoku Notation

A 9×9 Sudoku grid is made up of a 3×3 array of 3×3 *minigrids.* A row of this 3×3 array shall be referred to as a *band* and a column of this array is referred to as a *stack.* A minigrid is labelled M_{ij} if it appears in band i and stack j of the grid, where $i, j \in \{1, 2, 3\}$, and a single row of the grid shall be referred to simply as row, likewise for a column. This is shown diagrammatically in Figure 1.4.

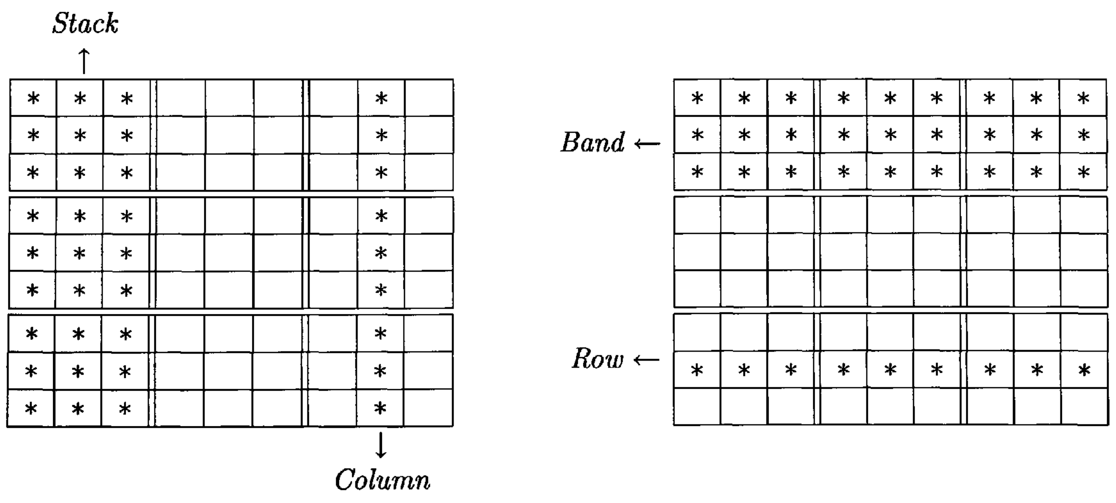


Figure 1.4: *Sudoku grid notation used in subsequent sections.*

Furthermore, a row of a *single minigrid* within a row shall be referred to as a *tier*, and a column of a single minigrid shall be referred to as a *pillar*. A specific cell of a Sudoku grid shall be referenced by firstly referring to the cell's minigrid by the notation given above, followed by a reference to its position within the minigrid's tiers and pillars; for example cell $[M_{22}]_{22}$ refers to the central cell of the grid. Further cell examples are provided in Figure 1.5, which also illustrates the entire minigrid notation.

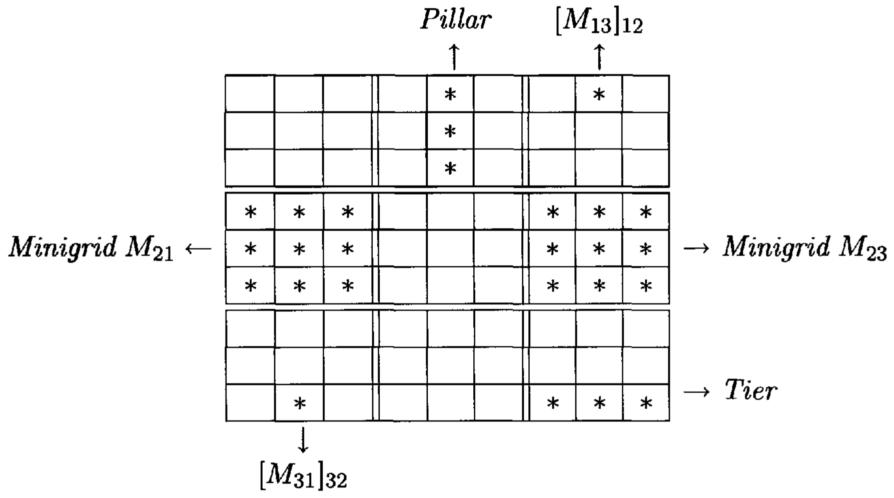


Figure 1.5: *Minigrid notation used in subsequent sections.*

1.5 Thesis Structure

The thesis is structured into six chapters, enclosed by introductory and summarising chapters. Puzzle-based combinatorial structures are explored as potential erasure correction tools within these chapters, as outlined below.

Chapter 2 reviews literature available in areas relevant to this research project; namely combinatorial objects, erasure-correcting codes and powerline communication – a potential application of erasure schemes incorporating permutation-based puzzles. The chapter initially discusses a variety of popular logic puzzles and progresses to discuss Sudoku-variant type puzzles, some of which may be useful to incorporate into a coding scheme. The chapter then provides a more in-depth discussion of relevant literature available on the two combinatorial objects of most importance to the research; Latin squares and Sudoku grids. Details of erasure-correcting codes are also covered in the literature review, primarily detailing applications of erasure-correcting codes and also discussing specific established erasure correction schemes. The final section of the chapter progresses to discuss powerline communication, the notion of utilising the current powerline infrastructure for the use of communication through the internet, and the specific type of code that is likely to be most suited to this application.

Chapter 3 discusses all relevant aspects for devising an outline erasure correction scheme that incorporates Sudoku, and concludes by detailing the scheme. The chapter opens by detailing the requirements of an erasure-correcting code and identifies the ways in which Sudoku is a useful combinatorial object that may be used to this end. The main focus of the chapter is to address the encoding of information within an empty Sudoku grid, identifying a suitable choice of cells to be used for the placement of the encoded message. The chapter proves that for the smaller Rodoku puzzle, a specific choice of cells is suitable for every possible encoded message that could be placed within the chosen cells, and details of a computation identifying a similar result for the larger 9×9 Sudoku puzzle are also provided.

Chapter 4 investigates the existence of a sub-structure of Rodoku and Sudoku grids known as *intercalates*. Intercalates are simply 2×2 sub-squares of a valid Sudoku grid that are themselves Latin squares, and their existence within some valid Sudoku or Rodoku grids has a significant effect on the erasure correction capabilities of the scheme outlined in Chapter 3. Hence, this chapter investigates the existence of these sub-structures and allows for conclusions to be made about the feasibility and efficiency of the scheme. Additionally, the chapter remarks on the limitations of the Sudoku scheme identified in Chapters 3 and 4 and discusses potential improvements that could be made to enable an erasure correction scheme based on Sudoku to be more desirable for erasure correction. The chapter concludes that Sudoku is a poor structure to use to this end.

Chapter 5 introduces a new structure GLUV (Grid with Length and Up-down Vectors) that could reasonably be expected to be more favourable for erasure correction purposes than Sudoku. Initial details including the size, alphabet, rate of the erasure scheme and details of the structure's individual components are discussed in this chapter.

Chapter 6 describes the solution strategies involved in a solver designed for recovering GLUVs, it also provides results of a simulation investigating the GLUV structure that incorporates this solver. The analysis of grids that the solver is unable to recover follows the statement of simulation results, and sub-structures within the unrecovered grids are identified that hamper its recovery; the sub-structures are referred to as *halting sets.*

Chapter 7 classifies different categories of the halting sets discovered in Chapter 6 and also derives lower bounds that indicate the probability of specific classes of halting sets occurring in a given GLUV. The evaluation of the established lower bounds at moderate erasure probabilities revealed that one particular class of halting set is particularly prevalent within GLUV puzzles. Identifying this property indicates that the recovery of GLUVs within the current scheme will always be hindered by the prevalence of halting sets, at a percentage that is higher than traditional schemes would tolerate. Consequently the scheme, though theoretically feasible, does not yield a practically viable erasure correction tool.

Finally, Chapter 8 concludes the thesis providing a detailed summary of the findings of the previous chapter, and outlining aims for the direction of possible future work.

Chapter 2

Literature Review

This chapter reviews current literature available on the topics relating to this research, namely: (i) a review of a representative selection of structurally different logic puzzles, with emphasis on Sudoku and Sudoku-variant puzzles; (ii) a review of literature available in the area of erasure correction, examining both a variety of recently established erasure-correcting codes and potential areas of application of these codes; and also (iii) a proposed application of Sudoku or Sudoku derived erasure correction schemes.

2.1 Logic Puzzles

Logic puzzles are a specific type of puzzle that require no particular prior knowledge to solve. Given some *clues* and a set of *rules*, it is possible to solve the puzzle by logic alone. There are many different categories of logic puzzle; this section is concerned with mathematical-logic and number-placement type puzzles discussing both recently popularised, and more historical, logic puzzles. Due to a lack of standard academic literature on the topics of logic puzzles and Sudoku-variant puzzles, much of the detail on these topics within this review is attributed to less academic sources including puzzle-creators' forums and webpages.

2.1.1 Kakuro

Kakuro is a popular number-placement puzzle, which is often likened to a mathematical equivalent of a crossword. The layout of the puzzle is similar to a crossword whereby some cells which occur in horizontal or vertical *runs* are white and are to be completed by the player. Others are shaded black and some of these cells contain clues to aid the player in completing the puzzle. Each run of white cells has a 'black cell clue' associated with it (above or to the left of the run); this clue tells the player the required sum of values to be placed in the particular run. The values $1, 2, \ldots, 9$ may each occur at most once in each run. An example of a simple Kakuro puzzle is shown in Figure 2.1.

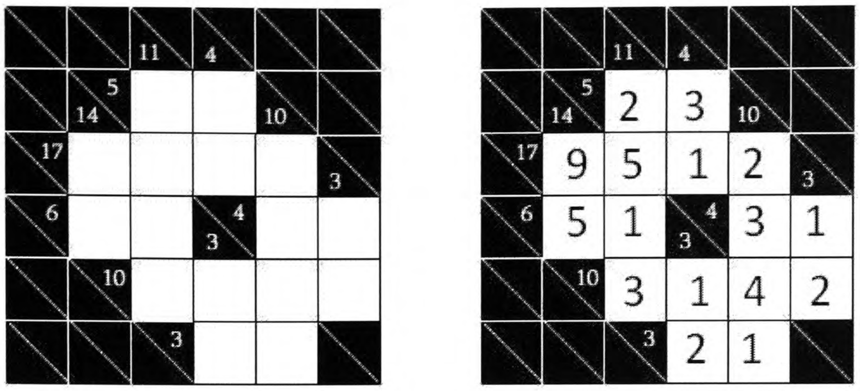
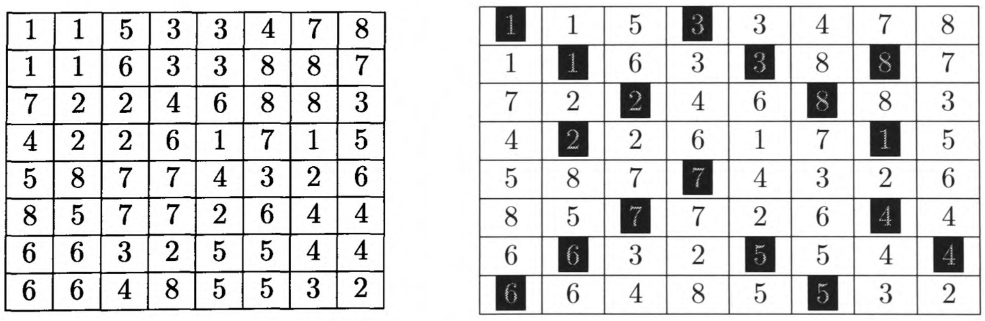


Figure 2.1: *An example of a small Kakuro puzzle, and its associated solution [4].*

2.1.2 Hitori

Hitori puzzles involve erasing certain values from a grid so that the remaining grid contains at most one of each value within each row and each column; the whole range of values need not all appear in each row and column. Two deleted cells cannot be placed horizontally or vertically adjacent to one-another. Diagonal-adjacency between deleted cells is permitted however, provided the diagonally-adjacent deleted cells do not disconnect any non-erased cells from one-another; the non-erased cells must form a single connected run or chain. For example, in the Hitori puzzle given in Figure 2.2 along with its associated solution, the cell occurring in row three and column eight of the grid containing the value '3' could not be erased as this would disconnect the horizontally-adjacent cell containing the value '8'.



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

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

Figure 2.2: *An example of an order 8 Hitori puzzle, and its associated solution [28].*

2.1.3 Magic Squares

Magic square puzzles provide another category of number-placement puzzle. The object of the puzzle is, given some initial clues, to complete the $n \times n$ grid with each of the values $1, 2, \ldots, n^2$

to form a 'magic square', a grid in which each row, each column and each non-broken diagonal contains values that sum to the same constant, sometimes referred to as the *magic constant*. The value of the magic constant, M, depends on the value of n and can be calculated by $M(n) = \frac{1}{2}n(n^2+1)$ [5]. For example, the magic constant of the order 3 puzzle in Figure 2.3 is $M(3) = \frac{3}{2}(3^2+1) = 15$, the puzzle's solution is also given.

	7	
4		8

2	7	6
9	5	1
4	3	8

Figure 2.3: *An example of an order 3 magic square puzzle, and its associated solution.*

2.2 Sudoku-Variant Puzzles

The popularity of Sudoku has motivated the creation of many variants of the puzzle, which shall be discussed in this section. The puzzles vary from standard Sudoku in a range of ways, and have been categorised in this thesis into three different types of variants: (i) size and layout variants; (ii) further-constrained variants; and (iii) clue variants.

2.2.1 Size and Layout Variants

Variants within this category exactly follow the rules of 'standard' 9×9 Sudoku, with no further constraints imposed on the puzzle. These variants differ only by their size, or the layout or shape of minigrids within the puzzle. Grids of order less than nine are generally referred to as *sub-doku* puzzles and puzzles of any order larger than nine are referred to generally as *super-doku* puzzles [8], although the majority of puzzles of specific orders have specific names.

2.2.1.1 Rodoku (6×6 Sudoku)

Rodoku is an example of a sub-doku puzzle and is set out on a 6×6 grid, and therefore contains rectangular 2×3 minigrids as opposed to square minigrids within standard Sudoku. 3×2 minigrids are also permitted, and result in an equivalent puzzle as a grid of this type can be achieved by transposing a puzzle containing 2×3 minigrids. The values $1, 2, \ldots, 6$ may occur exactly once within each row, column and each 2×3 minigrid as shown in Figure 2.4. Puzzles containing rectangular minigrids can be created for the majority of non-prime orders, the exception being when the order of the grid is a value that has a single prime factor. An example of this is the standard 9×9 Sudoku puzzle, the only factors of nine being one, three and itself; hence the grid may only be divided into square minigrids of order three.

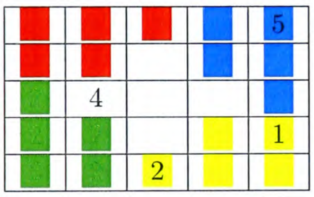
		4			
			2	3	
3				6	
	6				2
	2	1			
			5		

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

Figure 2.4: *An example of a Rodoku puzzle, and its associated solution [41].*

2.2.1.2 Prime-Sudoku

These types of puzzle permit grids of prime order, and deal with the inability to divide the minigrids into square or rectangular regions by allowing the minigrids to be irregular and varied in shape. The example in Figure 2.5 shows a Pento-Sudoku grid or a grid of order 5, but grids can be created for any prime (or equally non-prime) order.



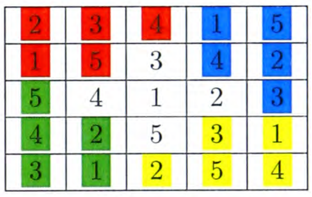


Figure 2.5: *An example of a Pento-Sudoku puzzle, and its associated solution [38].*

The completed Pento-Sudoku example shown above is an example of a Gerechte design. This is a specialisation of Latin squares that imposes a further constraint on the grid by subdividing the $n \times n$ grid into n regions which must also contain the symbols $1, 2, \ldots, n$. This is similar to the minigrid constraint imposed by Sudoku, however regions may vary in shape within Gerechte designs. Gerechte designs were introduced by W. U. Behrens in 1956 and have since been employed to ensure fair testing within various aspects of experimental design, including agricultural experiments [2].

2.2.1.3 Hexadecimal Sudoku (16 × 16 Sudoku)

16×16 Sudoku is simply a larger Sudoku puzzle, with 4×4 minigrids as opposed to 3×3 in standard Sudoku. Hexadecimal Sudoku varies only by the use of hexadecimal numbering as opposed to using the values $1, 2, \ldots, 16$. An example is shown in Figure 2.6.

4	7		F	5	9	C			A						
		E							0		C		9	7	6
B	6									D				A	
						1		5	8		6	4	E		
				F							B		4	8	A
3	C		1			0				E		D			2
8			A	2		E			D		0		3		
			7		C	3				9	2	E		F	
	9		6	4	D									1	
		D		6				0				9		4	C
1			5							3	7	0			
	F	0		1	A				2		D	8		B	E
	5		C							A					
					2			B							5
				C	6			D		1	4	F	B	E	8
		1	D					6	9	2	8	C	A		

4	7	3	F	5	9	C	6	2	A	B	E	1	8	D	0
D	1	E	8	3	B	A	2	F	0	4	C	5	9	7	6
B	6	5	0	7	E	4	8	3	1	D	9	2	C	A	F
9	A	C	2	D	0	1	F	5	8	7	6	4	E	3	B
E	0	2	9	F	1	6	D	C	3	5	B	7	4	8	A
3	C	F	1	B	4	0	9	8	7	E	A	D	6	5	2
8	4	6	A	2	7	E	5	1	D	F	0	B	3	C	9
5	D	B	7	8	C	3	A	4	6	9	2	E	0	F	1
2	9	7	6	4	D	8	0	E	B	C	5	A	F	1	3
A	3	D	B	6	5	2	E	0	F	8	1	9	7	4	C
1	E	8	5	9	F	B	C	A	4	3	7	0	2	6	D
C	F	0	4	1	A	7	3	9	2	6	D	8	5	B	E
6	5	9	C	0	8	D	B	7	E	A	F	3	1	2	4
7	8	4	E	A	2	F	1	B	C	0	3	6	D	9	5
0	2	A	3	C	6	9	7	D	5	1	4	F	B	E	8
F	B	1	D	E	3	5	4	6	9	2	8	C	A	0	7

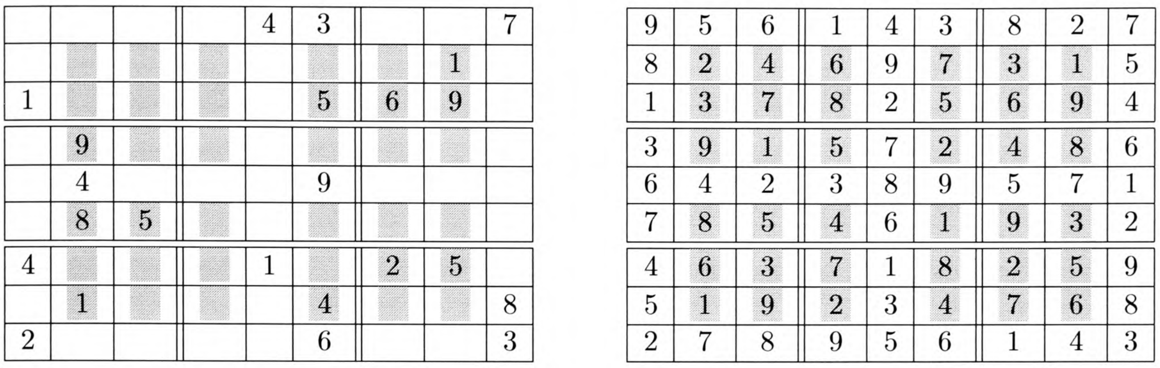
Figure 2.6: *An example of a Hexadecimal Sudoku puzzle and its associated solution [25].*

2.2.2 Further-Constrained Variants

These types of variants impose assorted further constraints on standard Sudoku puzzles. The addition of the further constraints allows the removal of more values from the grid such that a unique solution still exists and also permits a wider variety of solution strategies to be employed when attempting to solve the puzzle.

2.2.2.1 Hyper-Sudoku (4-Box Sudoku)

The further constraint on this type of variant is provided by the addition of four minigrids that overlap with the current nine minigrids; the values $1, 2, \ldots, 9$ must also occur exactly once within each of these minigrids. An example of a Hyper-Sudoku puzzle is shown in Figure 2.7, the additional minigrids are shown by the grey shaded regions within the grids.



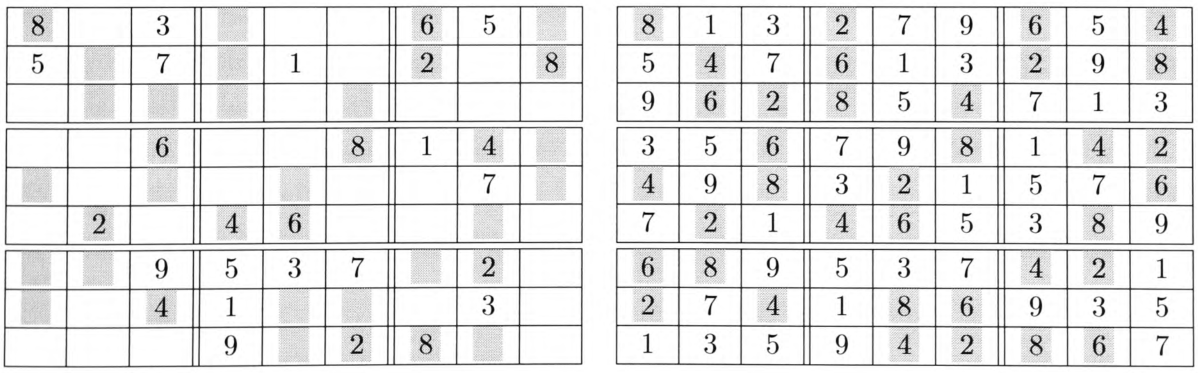
				4	3			7
							1	
1					5	6	9	
	9							
	4				9			
	8	5						
4				1		2	5	
	1				4			8
2					6			3

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

Figure 2.7: *An example of a Hyper-Sudoku puzzle, and its associated solution [34].*

2.2.2.2 Even-Odd Sudoku

In even-odd Sudoku, each cell of the grid is coloured with one of two colours. Even values may only be placed within cells of one of these colours, and odd values may only be placed in cells of the opposite colour.



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

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

Figure 2.8: *An example of an Even-Odd Sudoku puzzle, and its associated solution [33].*

2.2.2.3 Skyscraper Sudoku

The further constraint in skyscraper Sudoku is provided by values, varying from $1, 2, \ldots, 9$, given around the edge of the Sudoku grid. Each cell contained within the 9×9 grid can now be considered to represent a 'skyscraper' and the values contained within each cell to represent the height of the particular 'skyscraper'. The additional values around the outside of the grid represent the number of 'skyscrapers' that can be viewed from the vantage point indicated by the position of the value – either along a row or column, hence allowing the player to deduce the positioning of values within the grid. An example of this puzzle is given in Figure 2.9.

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

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

Figure 2.9: *An example of a Skyscraper Sudoku puzzle, and its associated solution.*

2.2.2.4 Quasi-Magic Sudoku

Quasi-magic Sudoku attempts to combine standard Sudoku and magic square puzzles, whereby each 3×3 minigrid is a magic square. However, every 3×3 magic square must contain a '5' in its central cell, and this would violate the constraints of Sudoku [7]. The extra constraint in quasi-magic Sudoku is therefore that each of the nine minigrids is a *quasi-magic square*; that is each tier, pillar and unbroken diagonal of each minigrid sums to a value in the range $[15 - \Delta, 15 + \Delta]$ where Δ is a fixed value ranging between two and nine (the case $\Delta = 9$ corresponds to a standard Sudoku puzzle). The value 15 is included in the range as it is the magic constant for 3×3 magic squares, as seen in Section 2.1.3. An example of such a puzzle for the case where $\Delta = 2$, and its solution, are given in Figure 2.10.

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

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

Figure 2.10: *An example of a Quasi-Magic Sudoku puzzle, and its associated solution [7].*

2.2.3 Clue Variants

These puzzles vary from standard Sudoku by the types of clues given to the player to enable them to deduce the unique valid Sudoku grid.

2.2.3.1 Outside Sudoku

The clues in Outside Sudoku are given outside of the Sudoku grid, and the grid is initially blank. The clues represent the values that must be placed in the tier or pillar of the minigrid immediately adjacent to the particular clue. The resultant grid must be a standard valid Sudoku. An example of an Outside Sudoku puzzle, with its solution, is given in Figure 2.11.

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

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

Figure 2.11: *An example of an Outside Sudoku puzzle, and its associated solution [39].*

2.2.3.2 Dice-Rodoku

Dice-Rodoku uses the values $1, 2, \ldots, 6$ as represented on a die as opposed to the numbers $1, \ldots, 6$. The puzzles follows the rules of 6×6 Rodoku, but the given dice within the grid are only partially completed and hence could represent more than one die value. An example of this is shown in Figure 2.12, also shown are the die-faces representative of the values $1, 2, \ldots, 6$.

○ ○ ○	○ ○ ●	○ ○ ●	● ○ ●	● ○ ●	● ● ●
○ ● ○	○ ○ ○	○ ● ○	○ ○ ○	○ ● ○	○ ○ ○
○ ○ ○	● ○ ○	● ○ ○	● ○ ●	● ○ ●	● ● ●
1	2	3	4	5	6

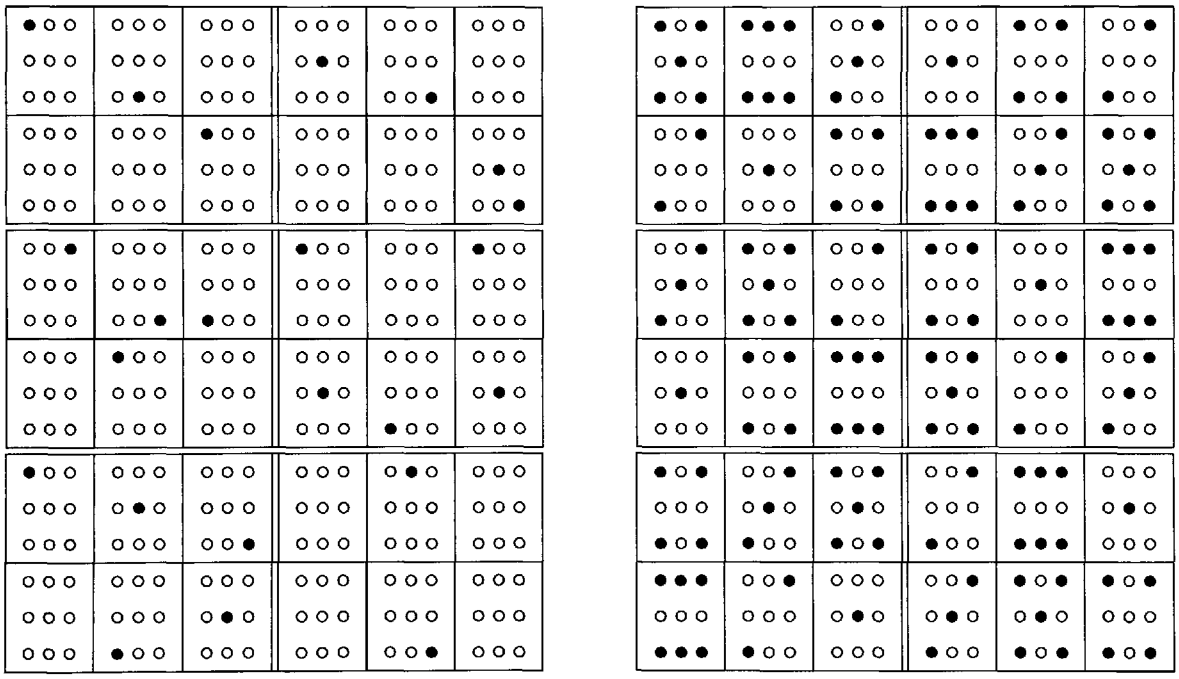
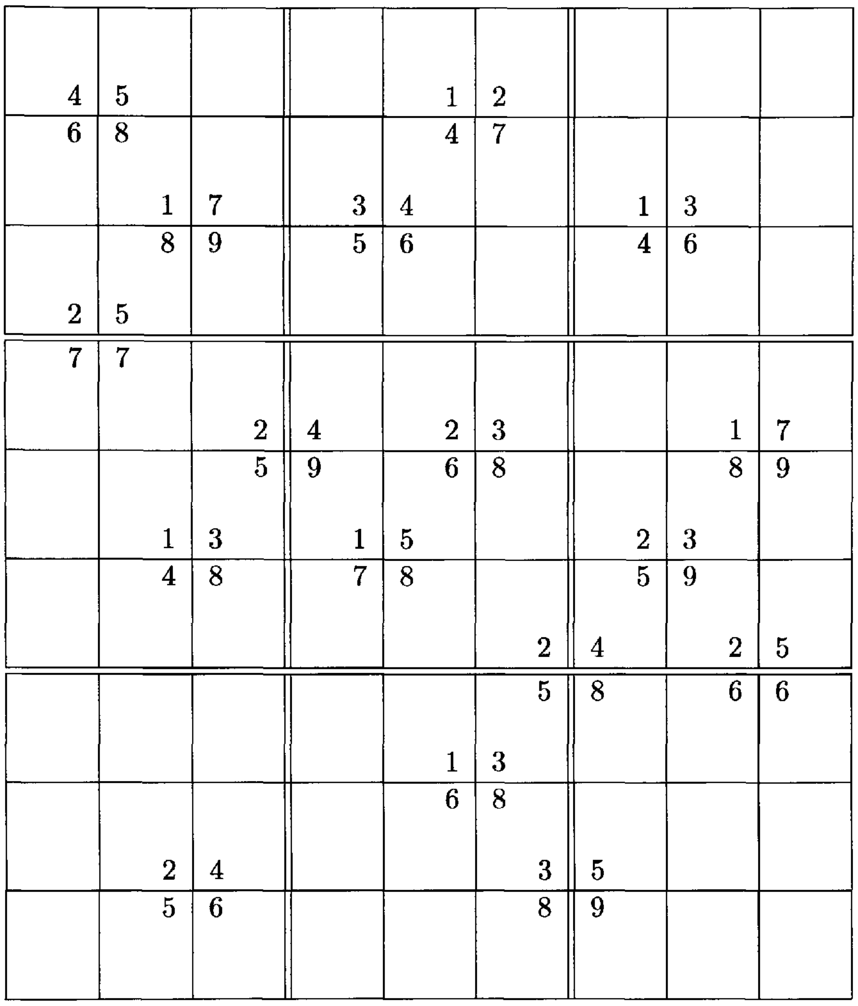


Figure 2.12: *An example of a Dice-Rodoku puzzle, and its associated solution [40].*

2.2.3.3 Quadruple-Clue Sudoku

Quadruple-Clue Sudoku provides clues at the intersection of four cells; each clue contains four values and denotes the four values to be placed in some way within the cells adjacent to the intersection at which the clue is given. The resultant grid must be a standard valid Sudoku grid. An example, and its solution are given in Figure 2.13.



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

Figure 2.13: *An example of a Quadruple-Clue Sudoku puzzle, and its associated solution [15].*

It is hoped that the various puzzles presented in this review will provide a variety of constraints that could potentially be incorporated into an erasure-correcting code derived from Sudoku or a related combinatorial structure.

2.3 Combinatorial Structures

2.3.1 Latin Squares

As stated in the introductory chapter, a Latin square is a square array that contains the values $1, 2, \ldots, n$, where n is the order of the array, exactly once in each row and once in each column. Latin squares have been extensively studied as combinatorial objects since their introduction by Euler in the late eighteenth century. The research carried out on the structures includes solving existence problems such as enumerating, by proof or computation, the number of order n Latin squares [24] for various values of n, or calculating the number of mutually-orthogonal Latin squares of various orders.

One area of importance to this research concerns the existence of Latin sub-squares within larger structures; these are of importance as they are likely to have a significant effect on the reliability of a coding scheme that incorporates Sudoku. For example, a Latin sub-square of order two, also referred to as an *intercalate*, within a Sudoku grid or Latin square consists of four cells that form a Latin square over two symbols. Erasure of all four cells of an intercalate from a valid Latin square or Sudoku grid, would mean that the grid could not be uniquely completed – an issue when recovery of an encoded message relies on the unique recovery or deduction of the original grid.

In 1998, McKay and Wanless [23] showed that most Latin squares contain many intercalates and that intercalate-free Latin squares are rare, with less than 1% of Latin Squares of order six or above being intercalate-free. Establishing similar results for Sudoku will be an important aspect of this research, and Chapter 4 discusses intercalates within Sudoku grids in more detail.

2.3.2 Sudoku

As stated previously, a Sudoku grid is a square grid of order 9. The grid is a Latin square with one further property; the grid is subdivided into 9 non-overlapping minigrids of order 3 and the values 1 to 9 appear exactly once within each of these minigrids, in addition to occurring exactly once in each row and each column. Sudoku grids, despite originating as a puzzle, are now considered as combinatorial objects and some research has been conducted on the structure; current literature concerns enumeration calculations for grids of various orders [6] and computing solutions to the Sudoku puzzle [14].

In 2007, Soedarmadji and McEliece [32] introduced a new class of erasure-correcting codes based on Latin squares and Sudoku, primarily introducing an iterative decoding algorithm for recovering Sudoku grids with erasures. This paper has motivated further research into the potential of Sudoku codes – including this research. Chapter 3 extends the work presented in [32], investigating the potential of Sudoku codes and detailing issues not explored by these authors, in particular the encoding of information in Sudoku grids.

2.4 Erasure-Correcting Codes

2.4.1 Applications of Erasure-Correcting Codes

This section is a continuation of the introductory coding theory section, Section 1.1, and will discuss both established and more recent applications of erasure-correcting codes, or simply *erasure codes* for brevity. As stated in the introduction, an *erasure-correcting code* is a code that can be used to correct erasure errors that occur in a message during transmission. Erasure-correcting codes differ from error-correcting codes in the way that errors present post-transmission; all received symbols that have been transmitted over an erasure channel are known to be correct and errors present simply as erasures, whereas the error-correction involves corrupted symbols where symbols affected by noise may present post-transmission as different symbols of the input alphabet. Error-correcting codes hence require the ability to detect and then correct errors, whereas erasure-correcting codes are concerned only with the correction of erased symbols. Generally, an erasure code transforms a message subdivided into a string of blocks of k symbols into a message which is a string of blocks of n symbols, where $n > k$, such that the original message may be recovered from a subset of the n symbols in each block [1]. Each message block consists of a string of input symbols of a specified fixed length.

By eliminating the requirement for continuous two-way communication over an erasure channel, by presenting the receiver with adequate information to reconstruct original messages without interaction with the transmitter, erasure-correcting codes can be applied to many areas of computer science and communication theory. A representative selection of these applications are detailed below.

A transport protocol is a protocol in place within various forms of two-way communication that is responsible for establishing a connection and ensuring that all data arrives safely. One of the most commonly used transport protocols for requesting repeat transmissions of corrupted data is the ARQ (Automatic Repeat reQuest) protocol. ARQ transmits only enough parity bits to allow the receiver to detect errors in the received message, but not to correct them. On detecting an error in the message, ARQ automatically issues a repeat request to the transmitter. However, there is no guaranteed maximum time delay when employing ARQ, as the number of retransmissions depends on the error distribution of the specific channel in use. As a result, a protocol that employs only a repeat request scheme such as ARQ is seldom suitable for real-time applications that require a guaranteed maximum time delay for correctly receiving data [21]. Examples of such applications include the recently popularised 'TV-on-demand' websites [30], such as BBC iPlayer, that allow viewers access to recently broadcast shows online. When transmitting video, it is important that particular blocks of information are received successfully from the server in time for the data to be used (played). Although video is a medium that can tolerate some loss of information, it is still preferable to employ a protocol that does not rely solely on ARQ and hence has no guaranteed maximum delay [21]. Often, the acceptable delay for information to be received is above the time required for a small number of transmissions or even a single retransmission. Hence, erasure correction can be employed to contribute to a more reliable protocol for video transmission.

Erasure codes must be employed when a feedback channel is unavailable, such as deep space communication, as the receiver is then unable to issue requests for repeat transmissions of data. Time-wise, it is also more economical to employ erasure codes when the return channel has too long a delay, such as satellite communication or in transcontinental internet/broadband networks; exploiting an erasure code in such instances reduces the time required to acquire all of the transmitted data and successfully decode it [21].

Many internet and broadband transport protocols still utilise more traditional transport protocols that were originally developed for use in telephone networks. It has been suggested that these protocols are not ideally suited for correcting errors that occur in broadband networks. As a result, recently published literature introduces new or altered codes that are designed specifically for use in broadband networks [26]. Errors occurring in broadband networks are predominantly due to congestion; this occurs when information is routed through a busy channel and there is hence a risk of losing some, or all, of the information. Loss of information due to congestion typically presents as a burst of erasures [21]. On occasions, the burst is too long to allow the errors to be corrected and a repeat request must be issued to the transmitter, however employing a transport protocol that combines an erasure-correcting code with the possibility of repeat requests (if required) efficiently covers all eventualities. Consequently, reliable broadband communication is best achieved by a hybrid transport protocol that incorporates both an erasure-correcting code and repeat requests, which can be used if the receiver encounters too many errors to correct by the employed erasure correction code.

The sending of redundant information in the form of an erasure code also has obvious benefits in mobile communication. Mobile communication devices, such as wireless-enabled laptop computers, cannot afford to waste power unnecessarily when more efficient methods exist for receiving and successfully decoding transmitted data. The use of erasure codes reduces the number of required repeat requests and transmissions and consequently could provide time, and hence power, saving benefits [30].

Commonly in communication, one central transmitter is responsible for the transmission of encoded data to multiple receivers. These multiple receivers are likely to experience information loss at different points within the transmitted data, and consequently the central transmitter has to cope with feedback from many of the receivers and retransmit the corrupted data to the receivers; repairing individual losses by this method can often incur significant expense. Erasure coding is a particularly useful feature in scenarios such as that outlined above as it is a means by which the amount of feedback can be reduced. As well as the economic benefits of reducing feedback, reduced feedback allows protocols to scale well to a large number of receivers [30]. A common example of a communication system which uses a centralised transmitter to convey information to multiple receivers is a classroom of students downloading files from a Virtual Learning Environment (VLE); the students download the file simultaneously from a centralised server (transmitter). If the transport protocol by which the file is sent to multiple receivers incorporates an erasure code, the process of the students downloading the file could be achieved more rapidly particularly when the number of students within the classroom (number of receivers to single transmitter) is large [30].

The problem of multiple receivers is also an issue in wider networks than the above local network example. Wider networks such as websites that transfer information to a large number of users/subscribers also need to scale well to a significantly larger number of receivers, but doing so becomes more difficult than within a localised network. The problem caused with downloads from websites is that receivers do not necessarily connect to the server simultaneously, and hence transmission of information to the multiple receivers cannot commence simultaneously; other issues include a significant variation in error distribution and loss rates for each receiver, and obviously congestion issues exist within the feedback channel. Erasure codes with a large amount of redundant information are of use in such circumstances as they dramatically reduce the amount of feedback, by ensuring that the receiver need only successfully receive a relatively small amount of information to reconstruct the original file or message [30].

2.4.2 Current Erasure-Correction Schemes

This section discusses recent relevant developments in erasure correction coding, particularly a recently developed class of codes known as *fountain* codes, or *rateless* erasure codes.

Fountain codes are so named due to the encoder being likened to a metaphorical fountain – capable of generating a limitless supply of encoded data [20]. Hence, the codes are often referred to as 'rateless' due to their potential to generate a limitless supply of encoded strings of symbols from the original message, all of which could be transmitted if necessary. The main advantage of fountain codes is that they are near-optimal for every erasure channel, regardless of the error rates on the channel. This near-optimality is a result of the rateless property, because as many encoded strings of symbols as needed for the decoder to recover the original data can always be transmitted. As a result of this, fountain codes provide a flexibility that cannot be offered by more traditional erasure correction schemes such as Reed-Solomon codes, or other block codes. With any block code it is necessary to estimate the erasure probability of the channel, and hence the required rate of the code before transmission [20]. Fountain codes are adaptable during transmission as the number of encoded strings of symbols to be transmitted is realised during transmission and does not require prior estimations which could possibly compromise the transmission if the erasure probability is higher than the expected value. Fountain codes also have a further advantage of minimising, if not fully alleviating, feedback in comparison to more traditional techniques such as the ARQ protocol described previously.

A fountain code is referred to as optimal if the original k strings of symbols can be recovered from any k encoded strings of symbols. Currently, fountain codes have been developed that are capable of recovering the original message from k' encoded strings of symbols, where k' is slightly larger than k. Typically, these codes are capable of operating reliably when k' is around 5% larger than k and are hence near-optimal.

The first practical fountain code, the Luby Transform code (LT code), was introduced by Luby in 2002 [17]. The code offers a relatively simple encoding and decoding algorithm that employs the exclusive or operator, $\oplus$. A more developed code that incorporates the LT code, known as the Raptor code introduced by Shokrollahi [31], was the first fountain code to achieve linear encoding and decoding.

The simplest fountain codes are random linear fountain codes. The encoder for a random linear fountain utilises a random number generator of some description (most likely a pseudo-random process or a deterministic random-number generator) to construct a generator matrix with a potentially infinite number of columns. Suppose a transmission of a file of size k packets was required; the encoder generates multiples of k random bits which will each form a column of the generator matrix and facilitate the transmission of a redundant packet obtained by taking a bitwise sum, modulo 2, of the k source packets. The next redundant packet to be transmitted is created in the same way using a different string of k random bits to form the packet – each transmitted packet can be said to be redundant as the source packets themselves are not transmitted [20].

Suppose a number of the transmitted redundant packets are erased, the likelihood of recovering the entire source file depends on the value n – the number of received packets. Assuming the receiver knows the portion of the generator matrix that relates to the n received redundant packets, by assuming that either a key for a pseudo-random number generator is transmitted with each transmitted packet or by the assumption that both the transmitter and receiver are using the same random-number generator. If $n < k$, the receiver will not be able to recover the file; in order to recover the file the portion of the generator matrix relating to the received packets must be invertible. When $n = k$, it is possible that the $k \times k$ matrix will be invertible, provided all columns of the generator matrix portion are linearly independent. MacKay [20] states this probability as being 0.289 for any $k > 10$. When $n = k + E$ where E represents a small number of excess packets, the question of recovery relates to being able to identify a $k \times k$ invertible matrix from the $k \times n$ generator matrix portion – drastically improving the probability of recovery to $1 - 2^{-E}$ [20]. Although random linear fountain codes are computationally expensive, LT codes reduce the encoding and decoding complexities, and fountain codes such as Raptor codes achieve linear encoding and decoding.

Fountain codes offer flexibility, low computational complexity and are near-optimal and as a result of this, any developments in erasure correction must offer codes with similar properties to compete with this class of codes.

2.5 Powerline Communication

A field in which there exists scope to incorporate Sudoku and similar structures is within *powerline communication*. Powerline communication is a technology, currently in its infancy, which allows data to be transmitted over the same lines used for the transmission of electricity, as opposed to the current technologies where data is transmitted along dedicated channels. The development of such a technology would be extremely useful as the technology would exploit current powerline infrastructures that already span vast parts of the developed world, allowing the internet to become far more accessible without causing disruption to provide new areas with internet access. Powerline communication would also be useful in alleviating the current 'digital divide' between urban and rural areas and is also likely to be of use in creating local area networks within offices and homes [10]. However, despite the elegance and vast potential of this technology, the area is in its infancy and many issues in both information theory and engineering need to be resolved before widespread use of powerline communication is feasible.

Current data transmission is achieved by sending encoded messages along dedicated channels; information theorists have a good understanding of the problems presented by these traditional channels and are able to effectively deal with the types of noise encountered. Powerlines, however, were not principally designed for the transmission of data, but for the transmission of electricity, and hence the types of noise encountered are more varied and present challenges to information theorists that present data transmission techniques do not address. Noise along traditional data channels typically affects data by randomly corrupting individual symbols of codewords; methods of overcoming this type of noise, typically *white Gaussian noise*, are well developed and effective in recovering corrupted data. The noise encountered within powerlines however, is of three main types; the first is the white Gaussian noise experienced within dedicated channels. The second is *permanent narrow-band noise*, this type of noise affects a specific frequency over a period of time, and the third; *impulse-noise*, a type of noise that affects many if not all frequencies for a short duration.

Recent research reveals effectively overcoming noise within powerlines can be achieved by increasing the range of frequencies used to transmit the data; that is increasing the *bandwidth* – this is useful in dealing with narrow-band noise – and also by increasing the number of timeslots used to transmit the data as a means of overcoming impulse noise [10]. The direction of current research as a consequence of these results leans toward *permutation codes* as being one of the more-robust ways of dealing with the varied noise within powerlines. These codes ensure that the modulated frequencies are used equally.

2.5.1 Permutation Codes

Permutation codes are codes in which each codeword is a permutation of the symbols $\{1, 2, \ldots, n\}$ and each codeword is hence of length n. The *minimum distance* or *Hamming distance* of a code is the minimum number of positions in which codewords differ. Codewords within a permutation code C must have at least minimum distance $d = 2$, as changing only a single value within one permutation does not yield a new permutation; for example given the identity permutation 1 2 3 4 5, altering only the first position to say a four yields 4 2 3 4 5 which is not a permutation, the fourth position must also be altered to form a valid permutation 4 2 3 1 5.

The maximum possible minimum distance for two codewords is n; that is they may differ in all positions. For example, it is possible to form an array known as a *permutation array,* which contains within each row, a codeword of the permutation code C; the array is denoted by PA(n, d) where n denotes the number of symbols of the input alphabet or equivalently the length of the codewords and d is the minimum distance of C. If we let $n = 7$, then in the case where $d = n$, we have PA(7,7) which is;

$$\text{PA(7,7)} = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & 4 & 5 & 6 & 7 & 1 \\ 3 & 4 & 5 & 6 & 7 & 1 & 2 \\ 4 & 5 & 6 & 7 & 1 & 2 & 3 \\ 5 & 6 & 7 & 1 & 2 & 3 & 4 \\ 6 & 7 & 1 & 2 & 3 & 4 & 5 \\ 7 & 1 & 2 & 3 & 4 & 5 & 6 \end{matrix}$$

In any case where the minimum distance equals n, the list of all possible codewords forms a Latin square, as the codewords must differ in all positions. A Sudoku grid can be considered as a Latin square with further constraints, and employing Sudoku in these cases could increase the reliability of these schemes as the further minigrid constraint improves the error-correction capabilities. Due to the extra constraint, a row-by-row construction with a view to forming a concatenation of codewords that obeys the constraints of Sudoku may not be feasible; the construction of a Sudoku grid would require a more complex encoding process. Thus it is likely that Sudoku would be most suited to powerline communication if each grid is representative of a single codeword, as opposed to each line of the grid representing a codeword as with typical permutation arrays.

Chapter 3

Sudoku Codes: An Outline Scheme

Following the proposals discussed in previous chapters for the use of Sudoku as an erasure correction tool, this chapter is concerned with the development of an initial coding scheme that incorporates Sudoku and deals with some of the preliminary combinatorial issues that need to be resolved to allow the development of the coding scheme.

3.1 Requirements of Erasure-Correcting Codes

Erasure-correcting codes are constructed by adding $n-k$ redundant symbols to each of the original message blocks of k symbols during the encoding process, creating an encoded message of strings of n symbols. These strings are then transmitted to the receiver via an erasure channel, where the channel has a certain probability of erasure. Upon receipt of the transmitted data, provided a sufficiently large number of symbols are received, the receiver is capable of correcting the erasures that have occurred, therefore reconstructing the original message.

Sudoku supports all of the afore-mentioned requirements of an erasure-correcting code as both are concerned with the recovery of original data from an incomplete subset of this data. The strong underlying mathematical structure of Sudoku is such that a unique valid grid can often be recovered from a relatively small subset of data (a small number of *clues*), and it is this property that provides the most significant justification for attempting to create a coding scheme that incorporates Sudoku, or a similar structure.

A successful initial coding scheme must satisfy the requirements given below.

- There must exist some method of encoding an original message into an empty Sudoku grid.
- It must be possible to add a fixed amount of redundant information to the encoded message.
- The addition of redundant information must be such that it aids the recovery of the original message post-transmission.

The Sudoku structure is able to support all of the above requirements. Firstly, the original message can be encoded into specific cells of an empty Sudoku grid; a simple and fairly efficient method is desired for doing so. Secondly, a fixed amount of redundant information can be added by completing the remaining empty cells of the grid to form a valid Sudoku grid. Finally, completing the empty cells of the grid certainly aids the recovery of the original encoded message; when erasures are likely to have occurred in certain cells of the grid, the added cell values can be used to deduce the values to be placed in cells whose values have been erased – this can be likened to the completion of a standard Sudoku puzzle. It must be noted, however, that a scheme satisfying the above criteria does not guarantee practical functionality as these criteria have no direct effects on the scheme's efficiency. The efficiency of any coding scheme developed from Sudoku will depend on the specifics of the methods developed for each of the above criteria.

There are however, issues that must be resolved before even an initial scheme is feasible. A crucial, initial issue is the choice of cells that should be used for the placement of the encoded message; this issue will affect many other aspects of the coding scheme, including the type of mapping that is required to map messages to empty grids. Also, the number of cells chosen for the placement of the encoded message will have an immediate effect on the efficiency of the code. This issue is therefore a sensible starting point and is discussed in the following section.

3.2 Encoding Information in a Sudoku Grid

The choice of cells to be used for the placement of encoded information (or messages) must be such that it attempts to maximise the number and the length of messages that can be encoded in the grid; yet, it must also be such that the placement of all possible different messages in an empty grid always permits an extension to a valid Sudoku grid. Extension to a valid Sudoku grid is necessary to allow redundant information to be added, allowing the original message to be recovered post-transmission.

In order to maximise the number of different messages that can be encoded and to avoid creating a complicated encoding method, it is desirable to be able to place the values $1, 2, \ldots, 9$ into each of the chosen cells; however this scenario is dramatically limiting as the chosen cells could then not appear in the same row, column or minigrid as one another, allowing only $9^9 = 387420489$ different messages to be encoded within a single grid. The use of the Sudoku structure is already limiting the number of messages that can be encoded as without requiring the addition of redundant information, it is possible to encode 9^{81} different messages in a 9×9 grid using the symbols $\{1, 2, \ldots, 9\}$; a more efficient choice of cells is needed.

This can be improved upon by turning attention away from individual cells and considering the permutations of the values $1, 2, \ldots, 9$ that may appear within each row, column and minigrid in Sudoku. Although row and column properties are not particularly useful, as cell inter-relationships always exist between two completed rows or columns, the use of entire minigrids for the placement of encoded messages is feasible as two completed minigrids do not necessarily have a direct influence on one-another's potential values. Provided the chosen minigrids are not horizontally or vertically adjacent to one another (such as the minigrids on the *leading diagonal*) they do not affect the possible arrangements of values that can be placed within each of the minigrids. As the grid can be considered as a 3×3 array of minigrids, we can use three minigrids for the placement of the encoded message and may place any permutation of $1, 2, \dots, 9$ within each minigrid. This choice of cells improves the number of different messages that can be encoded to $9!^3 = 47784725839872000$ (as there are 9! different permutations of the values $1, 2, \dots, 9$ and we can use three minigrids to each contain one of these possible permutations).

It is now necessary to check that this initial candidate is suitable, by ensuring that all *encoded grids*; that is grids that contain some permutation of the values $1, 2, \dots, 9$ in each minigrid on the leading diagonal, can always be extended to at least one valid Sudoku grid. This involves showing that in each of the $9!^3$ different instances of encoded grids, an extension to a valid grid exists. To begin to investigate this issue, a smaller structure Rodoku (discussed in Section 2.2.1) can be investigated.

3.2.1 Rodoku

Rodoku is a structure related to Sudoku, that is set out on a grid of order 6 and contains 6 rectangular 2×3 minigrids. The values $1, 2, \dots, 6$ occur exactly once in each row, each column and each minigrid.

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

Figure 3.1: *An example of a valid Rodoku grid.*

As Rodoku is made up of a rectangular 3×2 array of minigrids, it is only possible to use two of the minigrids for the placement of the encoded message – for our purposes, the minigrids on the leading diagonal. Due to this rectangular layout, some minigrids are more restricted by the placement of the encoded message than others and this property plus the reduction in the order of the grid allows the structure to be investigated more easily than Sudoku. A case-by-case investigation of encoded Rodoku grids results in the theorem and proof detailed below.

Theorem 1. *If the cells within the Rodoku minigrids M_{11} and M_{22}, are completed with any arbitrary arrangement of the values $1, 2, \dots, 6$; there always exist multiple valid completions of the entire 6×6 grid, satisfying the rules of Rodoku.*

Proof. The proof falls into three parts: the completion of M_{12}, the completion of M_{21} and then the simultaneous completion of the lower band of minigrids, M_{31} and M_{32}.

To deal with the completion of M_{12}, it is necessary to consider the different cases that exist for the relationship between the tiers of M_{11} and the pillars of M_{22}. Two cases can be identified;

1. One pillar of M_{22} contains two values (of a possible three) from the first tier of M_{11} and another pillar of M_{22} contains two values from the second tier of M_{11}.

2. Each pillar of M_{22} contains one value from each tier of M_{11}.

There are only two cases as there exist only two different ways of partitioning two sets of size three (the two rows of M_{11}) into three sets of size two (the three columns of M_{22}), with regard only to the element's original set, not its specific value. Examination of each of the above cases shows that, in the first case, two values can always be deduced uniquely in M_{12}; one in each tier. As a result, there remain two cells in each tier of M_{12} for which a choice of two values exist, therefore there remain precisely four ways of completing the remaining cells of M_{12}. Examination of the second case shows that no values within M_{12} can be deduced uniquely, there is a choice of precisely two values for each cell of M_{12}. However, in each individual tier of M_{12}, choosing and fixing one value (from a choice of two values) fixes that entire tier; therefore there are also precisely four ways of completing M_{12} in the second case.

The completion of M_{21} is independent of the completion of M_{12}. The completion of M_{21} can be achieved in a similar way, by reversing the roles of M_{11} and M_{22}; that is considering the tiers of M_{22} and the pillars of M_{11}, in contrast to the opposite above.

It remains to show that M_{31} and M_{32} can be simultaneously completed to form a valid Rodoku grid, given the completions for M_{11}, M_{12}, M_{21} and M_{22}. Each pillar of M_{31} and each pillar of M_{32} has two values remaining to fill the minigrids' empty cells. The potential placement of any value in $\{1, 2, \ldots, 6\}$ is restricted to a single pillar of M_{31} and similarly for M_{32}, due to the completion of the four minigrids above. As the two remaining empty minigrids are adjacent, the placement of values within one significantly restricts the placement of values in the other; therefore, both minigrids should be completed simultaneously.

A method for completing these grids simultaneously is to consider the relationship between the potential values for M_{31} and M_{32} as a graph G, with vertex set $\{1, 2, \ldots, 6\}$. Create a matching of edges E_1 such that $(mn) \in E_1$ if m and n can appear in the same pillar of M_{31}, where m, n are each vertices of the vertex set $\{1, 2, \ldots, 6\}$ and $m \neq n$. Then, create a second matching of edges E_2 where $(mn) \in E_2$ if m and n can appear in the same pillar of M_{32}. Let the edge set of G be $E = E_1 \cup E_2$, therefore G consists of either: (i) a single cycle of length 6, (ii) a cycle of length 4 and a cycle of length 2, or (iii) three cycles of length 2. Choosing one of the values within a cycle will fix all other values contained within the same cycle and as there are two choices of value for any cell within M_{31} or M_{32} there will therefore exist either: (i) 2, (ii) 4, or (iii) 8 valid completions of minigrids M_{31} and M_{32} for (i) a single cycle of length 6, (ii) a cycle of length 4 and a cycle of length 2, or (iii) three cycles of length 2 respectively. $\square$

Theorem 1 refers to the completion of specific minigrids, M_{11} and M_{22}. However, this can be extended to deal with the completion of any two minigrids that fall within different bands and stacks of the grid. This is because the movement from a main diagonal partial completion to any other valid start can be achieved by a series of band and stack permutations being applied to the 6×6 grid. Theorem 2, the generalisation of the above, is given below.

Theorem 2. *Suppose the cells within any two Rodoku minigrids M_{a1} and M_{b2}, where $a, b \in \{1, 2, 3\}$ and $a \neq b$, are completed with any arbitrary arrangement of the values $1, 2, \ldots, 6$; that is any two minigrids that occur in different bands and stacks of the grid. Then, there always exist multiple valid completions of the entire 6×6 grid, satisfying the rules of Rodoku.*

Therefore, it has been shown for 6×6 Rodoku that a grid in which data has suitably been encoded into the diagonal blocks can always be extended to a valid Rodoku grid, in fact multiple valid completions exist for every grid partially completed in this manner. Now it is necessary to test if the same property holds for 9×9 Sudoku.

3.2.2 Sudoku

Proving an equivalent result for Sudoku, that is showing that an extension to a valid 9×9 grid can always be achieved from a grid in which three non-adjacent minigrids have been suitably completed with values, is more difficult. The increased difficulty can be attributed to the square layout of minigrids within the grid which results in an increased number of cell inter-relationships when compared with Rodoku. As a result of this, no single minigrid can be completed without limiting the values that could be placed within the other minigrids, hence to follow a similar method as used for Rodoku would require simultaneous completion of all minigrids and this type of method is not desirable due to a dramatic increase in different cases of relationships between the three minigrids.

To check that this choice of cells holds for Sudoku, computational techniques were employed. The computation revealed that the result does indeed hold for Sudoku and the result is stated more formally below and is followed by details of the computation.

Computational Result 1. *Suppose the cells within any three Sudoku minigrids M_{a1}, M_{b2} and M_{c3}, where $a, b, c \in \{1, 2, 3\}$ and a, b, c are distinct, are completed with any arbitrary arrangement of the values $1, 2, \ldots, 9$; that is any three minigrids that occur in different bands and stacks of the grid. Then, there always exists at least one valid completion of the entire 9×9 grid, satisfying the rules of Sudoku.*

Details of Computation. The computation employed the frequency assignment software, FASoft [11], and hence required the problem to be mapped to an equivalent frequency assignment problem, by creating constraints on a graph that mimicked the rules of 9×9 Sudoku. This was convenient as FASoft already contains some useful features for this application. A frequency assignment problem is a problem that involves assigning frequencies to a number of transmitters such that the assigned frequencies minimise the interference between transmissions from different transmitters; interference is generally experienced between two geographically close transmitters assigned the same or similar frequencies, and the use of frequency assignment is intended to avoid such interference. A frequency assignment problem can be considered as a graph, where each vertex represents a transmitter and each edge between two vertices means that these transmitters should not be assigned the same frequency as interference will result. To perform a frequency assignment computation for Sudoku it was necessary to identify the constraints that mimicked the rules of the puzzle by numbering each of the 81 Sudoku cells and creating constraints so that no cell was assigned the same frequency (the frequencies to be assigned were $1, 2, \dots, 9$) as a cell that appeared in the same row, column or minigrid.

This particular computation employed one of the exhaustive search methods available in FASoft. Exhaustive searches [29] are impractical for large frequency assignment problems, but their incorporation into a predominantly metaheuristic [29, 18] system is important as it enables verification of the metaheuristics on fairly small problems. For problems in which the number of transmitters and/or the density of constraints are small, exhaustive searches can be more practical than heuristic algorithms [11]. FASoft includes two different exhaustive search techniques, *backtracking* [29] and *forward-checking* [11]; the simplest of the two techniques is backtracking. The backtracking algorithm uses a domain, D which lists all of the values to be assigned, in the case of Sudoku $D = \{1, 2, \dots, 9\}$, the first value of the domain is assigned to the first transmitter (or Sudoku cell) and the algorithm then checks that the assignment has not violated any of the constraints. If no violations occur then the algorithm progresses to assign a value to the next cell; however if a violation occurs, the algorithm backtracks and alters the assigned frequency to the next frequency listed in D and continues in this way [11]. In contrast, forward-checking assigns a value to cell only when it is certain that the assignment will not violate any constraints or disagree with any of the currently assigned values. In addition to the domain D used in backtracking, the forward checking technique also uses a current domain D^c which takes such constraints into account. Values are selected from the current domain and the algorithm then examines all constraints with future cells; violating frequencies are then erased from the current domain of these cells. This ensures that when a value is assigned to a cell, it does not violate any constraints with cells that have already been assigned values [11].

The computation was simplified without loss of generality by utilising many symmetry and permutation properties of the structure, detailed below; but essentially involved using the frequency assignment software to find a single extension to a valid grid for each of the partially-completed starting grids. The details of the simplifications applied are given below.

1. Firstly, without loss of generality, the values a, b and c, relating to minigrids M_{a1}, M_{b2} and M_{c3}, may be fixed such that $a, b, c \in \{1, 2, 3\}$ and a, b and c are distinct. This generalisation is permitted as the various other assignments of a, b and c to the values $1, 2, 3$ are permitted by the permutations of bands and stacks of the grid, as discussed previously. The chosen assignment for the computation was $a = 1, b = 2$ and $c = 3$.

2. Secondly, without loss of generality, it was possible to fix minigrid M_{11} to be any of the $9!$ permutations of the values $1, 2, \ldots, 9$, the chosen permutation being the values placed in natural order (row-by-row). The remaining $9! - 1$ starting grids ignored due to this generalisation can be generated by applying permutations of the values $1, 2, \ldots, 9$. Therefore, if a grid can be completed for one arrangement of M_{11}, extensions must always exist for the remaining $9! - 1$ arrangements.

3. Also, a single cell within each of the minigrids M_{22} and M_{33} could be fixed to one of the values in the range $1, 2, \ldots, 9$, without loss of generality. The computation fixed the top-left cell of each of the minigrids to the value 1. Other positionings of this value within the minigrid could be generated by applying row and/or column permutations to the grid.

4. It could also be shown, without loss of generality, that if a valid extension existed for some arrangement of values within M_{22} and another within M_{33} then a valid extension would always exist for a grid in which the arrangement of values within the two minigrids were swapped – again due to permutation of bands and stacks.

5. Finally, although no further cells or arrangements could be fixed for the entire computation, it was not necessary to check every remaining arrangement; generalisations due to row/column permutation properties were still possible. A second value within each of the minigrids M_{22} and M_{33} needed only to be checked in three other positions without loss of generality: (i) a cell within the same pillar; (ii) a cell within the same tier; (iii) a cell that was neither in the same tier or pillar. Besides the three required placements of this value, other placements could be reached by row and/or column permutations being applied to the grid.

Following these simplifications, the computation revealed that at least one extension to a valid grid existed for each of the $9!^3$ starting grids.

3.3 Initial Coding Scheme

Following the above investigation, an initial coding scheme is possible; an outline of which is given below.

1. An encoded message can be mapped to an empty Sudoku grid such that minigrids M_{11}, M_{22} and M_{33} (or equally M_{a1}, M_{b2} and M_{c3}, where $a, b, c \in \{1, 2, 3\}$ and a, b, c are distinct) are each completed with any of the 9! permutations of the values $1, 2, \ldots, 9$.
2. Redundant information is added by completing the remaining empty cells of the grid to form a valid Sudoku grid (this is always possible in both Rodoku and Sudoku as shown previously).
3. The valid grid can then be transmitted over an erasure channel.
4. The received grid is likely to contain some erasures, which may be overcome by solving the puzzle.
5. The encoded message can be extracted from the appropriate cells and decoded.

3.3.1 Efficiency of Initial Coding Scheme

The initial coding scheme uses 27 of the 81 cells for the placement of the encoded message, hence only one third of the cells are used. However, the rate of the coding scheme is less than a third as the scheme does not permit 9^{27} different encoded messages to be created, as a permutation of the values $1, 2, \ldots, 9$ must be placed in each of the minigrids on the leading diagonal. As a result, the rate of the scheme is;

$$\frac{\log_9(9!^3)}{81} = 0.21579.$$

Informally, it can be seen that $1 - p$ is an upper bound on the channel capacity for an erasure channel, where p is the probability of symbol erasure. This can be derived by assuming the presence of a 'genie' that informs the source each time a symbol is erased. By being informed of these erasures, the source is able to correct the erasures by transmitting the required symbols for a second time. If the probability of a single erasure is p, and we wish to transmit s symbols then we would expect ps symbols to be erased. The source, knowing of the erasures because of our 'genie', would transmit the ps symbols for a second time where there is once again a probability p of the symbols being erased, hence we would expect p^2s symbols to be erased on this occasion. Therefore, the total number of symbols we are required to transmit, S_t is given by;

$$S_t = s + ps + p^2s + p^3s + \ldots = s(1 + p + p^2 + p^3 + \ldots).$$

This can be simplified as the expression is a geometric progression with common ratio p. As $|p| < 1$ (as p is a probability) the sum to infinity can be calculated, hence;

$$S_t = s\left(\frac{1}{1-p}\right) = \frac{s}{1-p}.$$

The rate, R, of such a code is given by $\frac{k}{n}$ where k is the number of information symbols sent and n is the total of the information symbols and redundant symbols transmitted. Hence,

$$R = \frac{s}{\frac{s}{1-p}} = 1 - p.$$

As the existence of a genie is not possible, $1 - p$ is an upper bound for the capacity of the erasure channel over s symbols.

The capacity of the r-ary erasure channel can be derived more formally using *mutual information.*

3.3.1.1 Deriving the Capacity of the r-ary Erasure Channel using Mutual Information

Mutual information is a quantity that measures the relationship between two random variables that are simultaneously sampled; it is a measure of how much information is communicated in one variable about another. Hence, the mutual information $I(\mathcal{A}, \mathcal{B})$ for a channel represents the uncertainty about the input $\mathcal{A}$ resolved by knowing the output $\mathcal{B}$. For any given channel, we wish to maximise the mutual information by choosing the source $\mathcal{A}$ suitably. The capacity C of a channel Γ is defined to be the maximum value of $I(\mathcal{A}, \mathcal{B})$ where $\mathcal{A}$ ranges over all possible inputs for the channel; that is varying the input probabilities whilst fixing the probabilities of erasure for each symbol [12]. Thus, C depends on the channel alone, and represents the maximum amount of information which the given channel can transmit.

The mutual information, $I(\mathcal{A}, \mathcal{B})$, is given by:

$$I(\mathcal{A}, \mathcal{B}) = H(\mathcal{B}) - H(\mathcal{B}|\mathcal{A})$$

where H communicates the *uncertainty* of a given variable, specifically $H(\mathcal{B})$ is the *entropy* of $\mathcal{B}$; which can be thought of as the uncertainty about $\mathcal{B}$ when $\mathcal{A}$ is unknown. The equivocation $H(\mathcal{B}|\mathcal{A})$ is the uncertainty about $\mathcal{B}$ when $\mathcal{A}$ is known. Thus, the capacity C of a noisy channel is given by:

$$C = \max_{p(a)}(I(\mathcal{A}, \mathcal{B})) = \max\left(H(\mathcal{B}) - H(\mathcal{B}|\mathcal{A})\right).$$

where $p(a)$ represents all possible inputs for the channel. If we take the input of the r-ary erasure channel to be the source $\mathcal{A}$, as previously, with a finite alphabet of symbols $A = \{a_1, a_2, \dots, a_r\}$, where each symbol a_i is transmitted with probability p_i and the likelihood of symbol erasure is p_e. The output of the channel is $\mathcal{B}$ with a finite alphabet $B = \{a_1, a_2, \dots, a_r, e\}$, thus $|B| = |A| + 1$. Therefore, the channel matrix is:

$$\begin{array}{c|cccccc}
 & b=a_1 & b=a_2 & b=a_3 & \dots & b=a_r & b=e \\
\hline
a=a_1 & 1-p_e & 0 & 0 & \dots & 0 & p_e \\
a=a_2 & 0 & 1-p_e & 0 & \dots & 0 & p_e \\
a=a_3 & 0 & 0 & 1-p_e & \dots & 0 & p_e \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
a=a_r & 0 & 0 & 0 & \dots & 1-p_e & p_e
\end{array}$$

The capacity of a noisy channel is given by maximising the mutual information over all possible inputs for the channel, hence $C = \max\left(H(\mathcal{B}) - H(\mathcal{B}|\mathcal{A})\right)$. The channel matrix is uniform, therefore each symbol a_i is erased with equal probability, thus:

$$\begin{aligned}
H(\mathcal{B}|\mathcal{A}) &= \sum_{b\in B} P(b|a)\log_2\left(\frac{1}{P(b|a)}\right) \\
&= (1-p_e)\log_2\left(\frac{1}{1-p_e}\right) + (r-1)(0)\log_2\left(\frac{1}{0}\right) + p_e\log_2\left(\frac{1}{p_e}\right) \\
&= (1-p_e)\log_2\left(\frac{1}{1-p_e}\right) + p_e\log_2\left(\frac{1}{p_e}\right)
\end{aligned}$$

where $0\log_2\left(\frac{1}{0}\right)$ is taken to be zero, as there is no uncertainty relating to an event with zero probability – it is certain not to occur and thus has zero entropy. To find $H(\mathcal{B})$:

$$p(b=a_1) = p_1(1-p_e), p(b=a_2) = p_2(1-p_e), \dots, p(b=a_r) = p_r(1-p_e),$$

$$p(b=e) = (p_1 + p_2 + \dots + p_r)p_e = p_e$$

Thus, the probability that $b = a_s$ where $s \in \{1, 2, \dots, r\}$ is given by $p_s(1-p_e)$ and the probability that $b \neq a_s$ is $(1-p_s)(1-p_e)$, the probability the symbol is erased $(b = e)$ remains p_e, so:

$$\begin{aligned}
H(\mathcal{B}) &= p_s(1-p_e)\log_2\left(\frac{1}{p_s(1-p_e)}\right) + (1-p_s)(1-p_e)\log_2\left(\frac{1}{(1-p_s)(1-p_e)}\right) + p_e\log_2\left(\frac{1}{p_e}\right) \\
&= (1-p_e)\left[p_s\log_2\left(\frac{1}{p_s(1-p_e)}\right) + (1-p_s)\log_2\left(\frac{1}{(1-p_s)(1-p_e)}\right)\right] + p_e\log_2\left(\frac{1}{p_e}\right) \\
&= (1-p_e)\left[p_s\log_2\left(\frac{1}{p_s}\right) + (1-p_s)\log_2\left(\frac{1}{1-p_s}\right) + \log_2\left(\frac{1}{1-p_e}\right)\right] + p_e\log_2\left(\frac{1}{p_e}\right)
\end{aligned}$$

Therefore:

$$\begin{aligned}
I(\mathcal{A},\mathcal{B}) &= H(\mathcal{B}) - H(\mathcal{B}|\mathcal{A}) \\
&= (1-p_e)\left[p_s\log_2\left(\frac{1}{p_s}\right) + (1-p_s)\log_2\left(\frac{1}{1-p_s}\right) + \log_2\left(\frac{1}{1-p_e}\right)\right] + p_e\log_2\left(\frac{1}{p_e}\right) \\
&\quad - \left[(1-p_e)\log_2\left(\frac{1}{1-p_e}\right) + p_e\log_2\left(\frac{1}{p_e}\right)\right] \\
&= (1-p_e)\left[p_s\log_2\left(\frac{1}{p_s}\right) + (1-p_s)\log_2\left(\frac{1}{1-p_s}\right)\right] \\
&= (1-p_e)H(P)
\end{aligned}$$

where $H(P)$ is the *binary entropy function.* The binary entropy function describes the entropy of a variable that may only take two values; 0 or 1, where the probability of occurrence of one of the values is p and the other is $1-p$. $H(P)$ is maximal when $p = 1-p = \frac{1}{2}$, thus $H(P) = \frac{1}{2}\log_2\left(\frac{1}{\frac{1}{2}}\right) + \frac{1}{2}\log_2\left(\frac{1}{\frac{1}{2}}\right) = 1$. From this, the capacity of the r-ary erasure channel can be calculated:

$$C = \max_{p(a)}(I(\mathcal{A},\mathcal{B})) = \max[(1-p_e)H(P)] = (1-p_e)\max[H(P)] = 1-p_e$$

Therefore, the capacity C of an r-ary erasure channel is $C = 1-p_e$ [12].

3.3.2 Issues with Initial Coding Scheme

Although a basic outline scheme has been established, issues still exist with the scheme; the most important of these relates to the efficiency of the scheme.

As the capacity of the erasure channel is $1-p$ and the calculated rate of the current scheme is 0.21579, this indicates that to operate competitively with other established erasure correction techniques, such as digital fountain codes (discussed in Section 2.4.2), the coding scheme must be able to deal with erasure probabilities approaching $p = 1 - 0.21579 = 0.78421$ to operate closely to the channel capacity.

A simulation was employed to test the efficiency of the scheme for various erasure probabilities, and the graph shown in Figure 3.2 details the results of the simulation (shown by the red series). The simulation tested a sample of 1681 grids in which the minigrids M_{11}, M_{22} and M_{33} had been completed with values. For each of these grids, an extension to a valid grid was chosen from a large variety of completions. Following this, random cells were erased in accordance with the particular erasure probability (for example, if $p = 0.1$ then 10% of cells were randomly erased). FASoft [11] was again employed to attempt to solve the grid and recover the message, employing the same computation as discussed previously. If all cells occurring within minigrids M_{11}, M_{22} and M_{33} were solved then all of the information symbols were recovered and the message could hence be recovered; the simulation did not take into account the recovery of redundant symbols outside of the minigrids on the leading diagonal. This simulation was carried out on all 1681 grids for a variety of probabilities and the graph shown in Figure 3.2 shows the percentage of the 1681 grids that could be recovered for a given probability. The value 1681 is achieved by cyclical selection of grids using the set of grids generated in the simplifications described in Computational Result 1 in section 3.2.2.

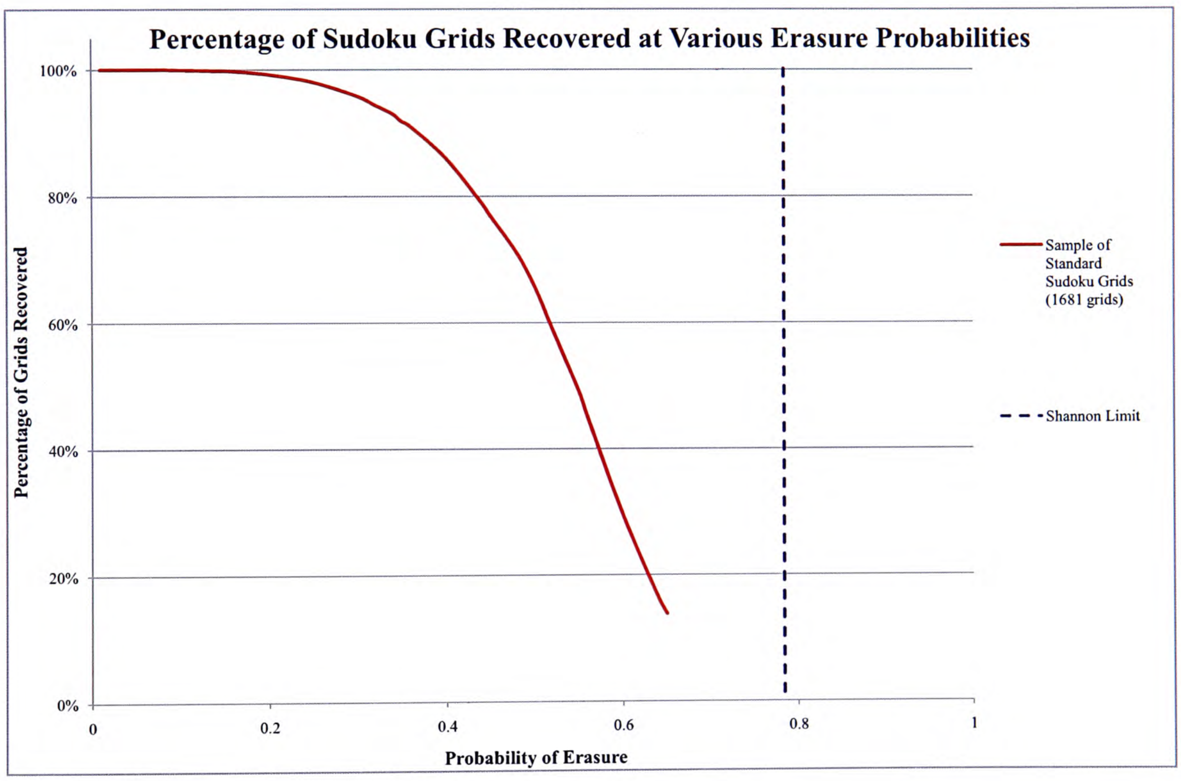


Figure 3.2: *Graph detailing percentage of errors against number of recoveries for various Sudoku grids.*

The blue dashed line that can be seen on the graph shows the target probability for the scheme as established above. Operating closely to this probability would mean that the code is operating efficiently and at these probabilities could rival the erasure correction capabilities of established erasure correction schemes. However, it can be seen in Figure 3.2 that when $p = 0.6$, that is when around only 32-33 cells contain values, less than 30% of grids are being recovered and hence the scheme thus far is not particularly efficient.

The failure of the scheme to operate efficiently is likely to be due to the patterns of erasures; in the cases where erasure patterns coincide with intercalates or Latin sub-squares, the grid is unlikely to be recovered. Intercalates and Latin sub-squares were described briefly in Section 2.3.1, the next chapter will further investigate the existence of intercalates within Rodoku and Sudoku, as it is these small types of Latin sub-squares that occur most frequently within valid grids and are most likely to result in issues with recovery. The chapter will concentrate on probability calculations for recovery for grids with various numbers of intercalates and also an investigation of the number of intercalates within different classes of grids within Rodoku.

It is reasonable at this stage to assume that investigating the number of intercalates within specific grids will suggest particular extensions to valid grids that contain fewer intercalates than others and as a result, specific extensions to valid grids can be used instead of random extensions, hopefully increasing the number of grids that can be recovered at higher probabilities.

Chapter 4

Intercalates and Efficiency

Within an order n Latin square, a Latin sub-square of order s is a sub-square of the $n \times n$ array that also preserves Latin square properties – that is, the sub-square is also a Latin square over s of the n possible symbols. The cells of the array containing the Latin sub-square need not be contiguous however, and, the largest Latin sub-square that can be contained within an order n array is $\lfloor \frac{n}{2} \rfloor$, else the $n \times n$ array could not form a Latin square as symbols would be repeated in rows and columns of the array [23].

An *intercalate* is an order two Latin sub-square over two symbols that is contained within a larger Latin structure. As well as occurring within Latin squares, Latin sub-squares may also occur in Sudoku grids. Within standard Sudoku, that is Sudoku with a square layout of square minigrids, the largest Latin sub-squares that can be contained within an order n grid are sub-squares of order $\sqrt{n}$ as the grid is constructed of $\sqrt{n}$ bands or stacks of minigrids of order $\sqrt{n}$ and larger subsquares would violate the minigrid constraint. Hence when compared with Latin squares, Sudoku grids are likely to contain fewer Latin sub-squares as the additional minigrid constraint prohibits sub-squares in which $\sqrt{(n)} < s < \lfloor \frac{n}{2} \rfloor$, where s is the order of the sub-square. In cases where the minigrids have a rectangular layout however, the largest Latin sub-square that can be contained with the grid are of order $max(m, n)$ where minigrids are of size $m \times n$ or $n \times m$. This is due to the grid consisting of, without loss of generality, m bands and n stacks of $n \times m$ minigrids – a subsquare of $max(m, n)$ is possible as the number of bands and stacks permits this without violating the minigrid constraint.

From an erasure correction viewpoint, intercalates pose the greatest threat to the recovery of a valid Sudoku grid, as the erasure of all four cells of an intercalate results in a grid with two potential solutions. Larger Latin sub-squares are also potentially problematic, however the probability of all nine cells of an order 3 Latin sub-square being erased is far lower than the probability of four intercalate cells being erased. As a result, this chapter concentrates on Latin sub-squares of order two and the effects on the rate of recovery of Rodoku and Sudoku grids.

4.1 Existence of Intercalates in Rodoku

By the outline scheme introduced in the previous chapter, there are $6!^2$ different partially completed or *encoded* Rodoku grids, all of which can be extended to valid Rodoku grids in a number of different ways. From an erasure correction perspective, it is necessary to analyse the $6!^2$ encoded grids to ascertain which of the multiple completions of each encoded grid contains the smallest number of intercalates, as it is this completion that is likely to be most robust against cell erasures. In order to achieve this, it is necessary to divide the $6!^2$ encoded grids into cases and analyse these cases to determine the minimum number of intercalates that exist in valid completions for each case. The cases are given in Section 4.1.2.

4.1.1 Notation and Terminology

The *tier set* of a minigrid M_{ij}, denoted by T_{ij}, is a set containing sets of all elements in each tier of minigrid M_{ij}. For example, consider the following 2×3 minigrid.

a	b	c
d	e	f

The tier set of the above minigrid is $\{\{a, b, c\}, \{d, e, f\}\}$. Similarly, the *pillar set* of a minigrid M_{ij}, denoted by P_{ij}, is a set containing sets of all elements of each pillar of minigrid M_{ij}. For example, the pillar set of the minigrid above is $\{\{a, d\}, \{b, e\}, \{c, f\}\}$.

4.1.2 Deriving The 'Encoded Rodoku Grid Cases'

There are $6!^2$ different encoded Rodoku grids; that is the number of grids that can be created by completing minigrids M_{11} and M_{22} with any arrangement of the values $\{1, 2, \ldots, 6\}$. These $6!^2$ partially completed grids can be divided into only 9 cases by categorising them according to three defining properties, detailed below.

1. The relationship between the tiers of minigrids M_{11} and M_{22}:
   (a) The tier set of M_{11} equals the tier set of M_{22}, that is $T_{11} = T_{22}$;
   (b) The tier sets of M_{11} and M_{22} are not equal, that is $T_{11} \neq T_{22}$.
2. The relationship between the pillars of minigrids M_{11} and M_{22}:
   (a) The intersection of the pillar sets P_{11} and P_{22} results in a set of size three, $|P_{11} \cap P_{22}| = 3$;
   (b) The intersection of the pillar sets P_{11} and P_{22} results in a set of size one, $|P_{11} \cap P_{22}| = 1$;
   (c) The intersection of the pillar sets P_{11} and P_{22} results in the empty set, $|P_{11} \cap P_{22}| = 0$.
3. The number of values fixed as a result of the arrangement of values within minigrids M_{11} and M_{22}:

(a) No values are fixed as a result of the arrangement of values in M_{11} and M_{22};

(b) Two values are fixed as a result of the arrangement of values in M_{11} and M_{22}, either in M_{12} or M_{21};

(c) Four values are fixed as a result of the arrangement of values in M_{11} and M_{22}, two in each of M_{12} and M_{21}.

These properties potentially create $2 \times 3 \times 3 = 18$ different cases of Rodoku grids, however obeying the rules of Rodoku it is not possible to combine some of the mentioned properties, leaving 9 cases. For example, if the pillar sets of M_{11} and M_{22} are equal, there is no possibility of fixed values outside of M_{11} and M_{22}. The nine resultant cases are outlined in Table 4.1.

Case No	Tier Relationship	Pillar Relationship	No of Fixed Values	% of $6!^2$ Grids
1	$T_{11} \neq T_{22}$	$\|P_{11} \cap P_{22}\| = 0$	2	20%
2	$T_{11} \neq T_{22}$	$\|P_{11} \cap P_{22}\| = 1$	2	20%
3	$T_{11} \neq T_{22}$	$\|P_{11} \cap P_{22}\| = 1$	4	10%
4	$T_{11} \neq T_{22}$	$\|P_{11} \cap P_{22}\| = 1$	0	5%
5*	$T_{11} \neq T_{22}$	$\|P_{11} \cap P_{22}\| = 0$	4	30%
6	$T_{11} \neq T_{22}$	$\|P_{11} \cap P_{22}\| = 3$	0	5%
7	$T_{11} = T_{22}$	$\|P_{11} \cap P_{22}\| = 0$	0	$3\frac{1}{3}$%
8	$T_{11} = T_{22}$	$\|P_{11} \cap P_{22}\| = 1$	0	5%
9	$T_{11} = T_{22}$	$\|P_{11} \cap P_{22}\| = 3$	0	$1\frac{2}{3}$%

Table 4.1: *The nine different cases of encoded Rodoku grids.*

As previously discussed whilst outlining the defining properties of encoded Rodoku grids, the relationship between the arrangement of values within M_{11} and M_{22} can fix values within M_{12} and M_{21}. When values are fixed within M_{12}, M_{21} or both, a further related property can be identified which is useful for further investigation of intercalates within the Rodoku structure, and also permits a tenth case of encoded minigrid to be acknowledged by splitting the largest case in Table 4.1, case 5*, into two, more detailed cases. The property and the resultant splitting of case 5* is outlined subsequently.

Consider a single minigrid M_{ab}, where $a, b \in \{1, 2\}$ and $a \neq b$, where the arrangement of values in the two adjacent encoded minigrids M_{aa} and M_{bb} is such that M_{ab} contains two fixed values. The relationship between M_{aa} and M_{bb} is such that the values from each of the tiers of M_{aa} are arranged in the pillars of M_{bb} so that one pillar contains two values from the first tier of M_{aa}, the second pillar of M_{bb} contains two values that appear together in the second tier of M_{aa}, and hence the final pillar must contain two values, one from each tier of M_{aa}. Hence, the pillars of M_{bb} may be categorised by their relationship with the tiers of M_{aa} as such: a pillar that consists of two values from the same tier of M_{aa} shall be referred to as a *like pillar*. Whereas, a pillar consisting of two values; one from each of the tiers of M_{aa} shall be referred to as a *mixed pillar*.

It is the mixed pillar that is of most interest as the values contained within the mixed pillar of M_{bb} are the two values fixed to specific cells within minigrid M_{ab}.

Mixed pillars also occur in the encoded minigrids when no values are fixed outside of M_{11} and M_{22}; in these cases, every pillar of both encoded minigrids is mixed, and there is no obvious relationship between these mixed pillars and fixed values outside of M_{11} and M_{22}. However, when one or both of M_{11} and M_{22} contain a *single* mixed pillar combined with two like pillars, the single mixed pillar is useful in making further observations on the structure. The first of these is allowing case 5*, shown in Table 4.1, to be divided into two, more specific cases. Case 5* encoded grids contain two fixed values in each of M_{12} and M_{21}, indicating that both M_{11} and M_{22} contain two like pillars and, more importantly, a single mixed pillar each. If we consider the two values of M_{11}'s mixed pillar as a set and do likewise for the mixed pillar M_{22}, a distinction can be made between the various grids of case 5*. Taking the intersection of the two sets can result in either a set of size one (hence, the single mixed pillars of M_{11} and M_{22} share a common value), or the empty set (the mixed pillars have no common values).

On further investigation it transpires that the grids of case 5* whose single mixed pillars share a common value have 58 completions and those with no common value within the mixed pillars have 66 completions. Separating the case by the distinction described results in ten encoded Rodoku grid cases, which are more appropriate as the cases are now separated not only by their shared properties but also by the number of ways they can be extended to a valid grid also. Table 4.2 shows the new cases, where the case numbers have been altered to correspond to the number of extensions to a valid grid which exist for a given case (that is, case 46 contains grids with 46 completions and likewise for subsequent case numbers).

4.1.3 Observations of Intercalate Occurrence in Encoded Rodoku Grid Cases

The initial investigation of the existence of intercalates within each of the ten encoded Rodoku grid cases will begin by examining the minigrids in the upper two bands of the grid, observations on the lower band will follow. Some basic terminology is required to make the distinction between the types of intercalates; an intercalate occurring within a single stack of the Rodoku grid shall be called a *same-stack* intercalate and an intercalate that occurs within a single band shall be referred to as a *same-band* intercalate. The remainder of this subsection outlines some initial observations on intercalates within encoded grids of a certain case.

Lemma 1. *Same-stack intercalates do not occur in the top two bands of the grid if the grid contains no fixed values outside of* M_{11} *and* M_{22}.

Proof. Consider the minigrid M_{ab} where $a, b \in \{1, 2\}$ and $a \neq b$. If no values are fixed in M_{ab} as a result of the arrangement of values in the encoded minigrids, then each value from a given tier of the encoded minigrid horizontally-adjacent to M_{ab}, M_{aa}, must occur within different pillars of the encoded minigrid M_{bb}. An attempt to form a same-stack intercalate within a tier of M_{ab}

Case No	Tier Relationship	Pillar Relationship	No of Fixed Values	% of $6!^2$ Grids
46	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 0$	2	20%
48	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 1$	2	20%
52	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 1$	4	10%
56	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 1$	0	5%
58	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 0$ (the single mixed pillars of M_{11} and M_{22} share a common value)	4	20%
60	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 3$	0	5%
62	$T_{11} = T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 0$	0	$3\frac{1}{3}$%
64	$T_{11} = T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 1$	0	5%
66	$T_{11} \neq T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 0$ (the single mixed pillars of M_{11} and M_{22} have no common value)	4	10%
68	$T_{11} = T_{22}$	$\lvert P_{11} \cap P_{22}\rvert = 3$	0	$1\frac{2}{3}$%

Table 4.2: *The ten different cases of encoded Rodoku grids, accounting for single mixed pillars distinction.*

would hence result in a grid that does not satisfy the column constraint of Rodoku as the same value would have to occur in some pillar of M_{ab} and the vertically-adjacent pillar of M_{bb}. □

Therefore, encoded grids that do not force any fixed values contain no same-stack intercalates in the top two bands of the grid. So, valid Rodoku grids can be constructed from cases 56, 60, 62, 64 and 68 such that they contain no same-stack intercalates in the upper two bands. An example of such a case, as discussed in Lemma 1, is given below in Figure 4.1.

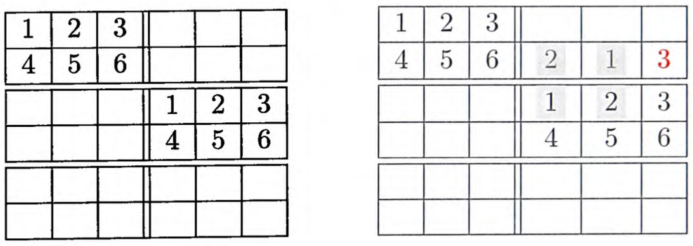


Figure 4.1: *An example of the type of Rodoku grid discussed in Lemma 1.*

Figure 4.1 shows the original encoded grid on the left, on the right the grid shown demonstrates an attempt to create an intercalate over the symbols $\{1, 2\}$ in the second tier of M_{12} and the first tier of M_{22}, the symbol 3, highlighted in red, shows the violation of the column constraint of Rodoku.

Lemma 2. *If two values are fixed in precisely one of the minigrids M_{12} or M_{21} as a result of the arrangement of values in M_{11} and M_{22} then a quarter of completions of the upper two bands contain no same-stack intercalates.*

Proof. Consider a single stack of the Rodoku grid in which M_{ab} contains two fixed values, where $a, b \in \{1, 2\}$ and $a \neq b$. The two fixed values housed within M_{ab} are those that occur in the single mixed pillar of the vertically-adjacent encoded minigrid M_{bb}. The fixed values occur in vertical adjacency to the each of the like pillars of the minigrid M_{bb}, one in each tier of M_{ab}. Hence, each tier of M_{ab} can be completed in 2! ways, and there are hence $2!^2 = 4$ ways of completing M_{ab}.

Consider a completion of a single tier; one value is already fixed and the remaining pair of values may be ordered in one of two different ways. The pillar of the M_{bb} that is vertically-adjacent to a fixed value of M_{ab} corresponds to the two values that must be placed in the same tier as the fixed value in M_{ab}. As a result of this, one of the two completions of a particular tier of M_{ab} will always result in a same-stack intercalate. Hence, when considering both tiers of M_{ab}, an intercalate is formed in half of the arrangements of each tier. Therefore $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$ of arrangements have two same-stack intercalates, $\frac{2}{4} = \frac{1}{2}$ have one same-stack intercalate and $\frac{1}{4}$ have no same stack intercalates (in top-two bands). □

An example of such an encoded grid, as discussed in Lemma 2, is given below in Figure 4.2.

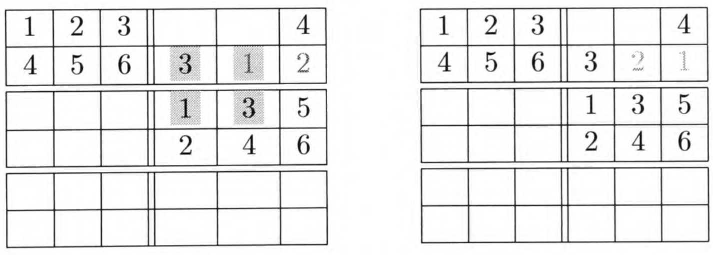


Figure 4.2: *An example of the type of Rodoku grid discussed in Lemma 2.*

Figure 4.2 shows the two possible completions of the second tier of M_{12}. The single mixed pillar of encoded minigrid M_{22} is the central pillar of M_{22}, the values of which correspond to the two fixed values in M_{12}. The grid shown on the left hand side results in an intercalate over the symbols $\{1, 3\}$, whereas the right-hand grid does not contain an intercalate.

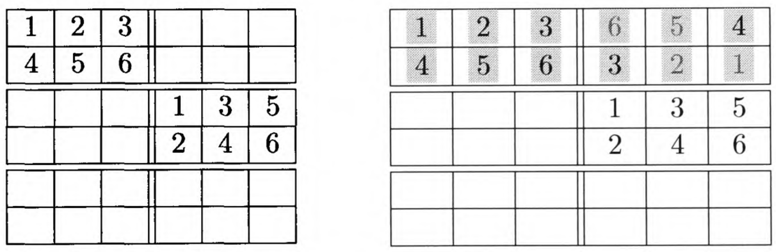
Lemma 3. *If two values are fixed in both minigrid M_{12} and in minigrid M_{21} as a result of the arrangement of values in M_{11} and M_{22} then $\frac{1}{16}$ of completions of the upper two bands contain no same-stack intercalates.*

Proof. As a consequence of the results on fixed values in a single stack, discussed in Lemma 2, grids with fixed values in both stacks (four fixed values) can be considered. From Lemma 2, if $\frac{1}{4}$ of completions of the first stack contain no intercalates and likewise for the second stack then $\frac{1}{4} \times \frac{1}{4} = \frac{1}{16}$ of completions of the top two bands of the Rodoku grid contain no same-stack intercalates. $\square$

It now remains to deduce if the grids identified above as being same-stack intercalate free in the top two bands contain either same-band intercalates or intercalates in the lower band of the grids.

Lemma 4. *Grids with two fixed values in precisely one of minigrids M_{12} and M_{21} satisfying the conditions $T_{11} \neq T_{22}$ and $|P_{11} \cap P_{22}| = 0$ always contain intercalate(s).*

Proof. As $|P_{11} \cap P_{22}| = 0$, the encoded minigrids M_{aa} and M_{bb} do not have any pillars in common. As M_{ab} (where $a, b \in \{1, 2\}$ and $a \neq b$) contains two fixed values, the encoded minigrid contained in the same stack as M_{ab}, minigrid M_{bb}, consists of two like pillars and a single mixed pillar in relation to the arrangement of values comparative to the tiers of M_{aa}. Hence, each of the four values to be arranged within M_{ab} can be placed in one of two cells – resulting in four different potential completions. As a consequence of Lemma 2, it is known that $\frac{3}{4}$ of the completions of M_{ab} result in same-stack intercalates, therefore only one of the four completions avoid same-stack intercalates. Placing values into M_{ab} such that no same-stack intercalates occur over M_{ab} and M_{bb} results in $|P_{aa} \cap P_{ab}| = 3$, hence three same-band intercalates occur over band a where $a \in \{1, 2\}$. $\square$



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

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

Figure 4.3: *An example of the type of Rodoku grid discussed in Lemma 4.*

Figure 4.3 gives an example of the type of Rodoku grid discussed in Lemma 4, the left-hand grid shows the initial encoded grid and the right-hand grid shows the single completion of M_{12} that contains no same-stack intercalates – it can be seen that this completion results in three same-band intercalates across the first band of the grid.

Lemma 5. *Grids with two fixed values in precisely one of minigrids M_{12} and M_{21} satisfying the conditions $T_{11} \neq T_{22}$ and $|P_{11} \cap P_{22}| = 1$ always contain intercalate(s).*

Proof. Without loss of generality, the two fixed values can be assumed to be in M_{12}, as permutations of bands and stacks could alter the positioning of encoded minigrids M_{11} and M_{22}. Two fixed values occurring in M_{12} imply two like pillars and one mixed pillar in M_{22} in relation to

M_{11} – the set of fixed values in M_{12} equals the mixed pillar of M_{22}. Of the remaining four cells of M_{12}, there are four possible completions of the minigrid, involving the values occurring in the like pillars of M_{22}; two independent completions of each tier of M_{12}. As $T_{11} \neq T_{22}$ the tiers of M_{22} will each always contain two values matching those contained in the tiers of M_{11} and hence M_{12}, creating the possibility of same-stack intercalates. One of the tier completions for each tier of M_{12} results in a same-stack intercalate between the values of M_{12} and M_{22}. The remaining completion for each tier, that avoids such a same-stack intercalate, results in a single same-band intercalate between M_{11} and M_{12}, and hence Rodoku grids of this type always contain intercalate(s). □

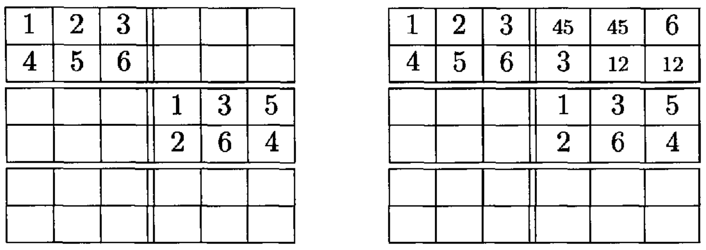


Figure 4.4: *The possible completions of the type of Rodoku grid discussed in Lemma 5; intercalate(s) must occur either over a band or over a stack.*

Figure 4.4 shows an example of a grid of the type discussed in Lemma 5; the left-hand grid shows the original encoded grid, and the grid on the right shows the possible completions of M_{12}. The figure shows that either a same-stack intercalate(s) or a same-band intercalate must be formed when extending a grid of this type; this is true for all grids satisfying the conditions of Lemma 5, as proved previously.

Lemma 6. *Grids with two fixed values in each of the minigrids M_{12} and M_{21} satisfying the conditions $T_{11} \neq T_{22}$ and $|P_{11} \cap P_{22}| = 1$ always contain intercalate(s).*

Proof. Given that M_{12} and M_{21} each contain two fixed values, M_{11} and M_{22} must each consist of two like pillars and a single mixed pillar in relation to one another's tier sets. By definition $|P_{11} \cap P_{22}| = 1$, thus a pillar of M_{11} equals a pillar of M_{22}. This pillar must relate to the single mixed pillar in each encoded minigrid as each like pillar of M_{11} is composed of two values that occur together in the same tier of minigrid M_{22} (as is also the case for the tiers of M_{11} and the like pillars of M_{22}). Thus, the mixed pillar of each of minigrids M_{11} and M_{22} is equal.

As a result of Lemma 3, it is known that only one completion of the top two bands of the grid contains no same-stack intercalates; this completion involves the same two values being placed in M_{12} directly above the single mixed pillar of M_{22} as are placed in M_{21} below the single mixed pillar of M_{11}. As a result, the concatenation of each of the single mixed pillars with the pillars directly above/below yield *partial* column completions that contain the same four values – forcing a same-band intercalate across the lower band of minigrids. Therefore Rodoku grids of this type always contain intercalate(s).

□

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

1	2	3	5	4	6
4	5	6	3	2	1
5	6	2	1	3	4
3	1	4	2	6	5
		15		15	
		15		15	

Figure 4.5: *An example of the type of Rodoku grid discussed in Lemma 6.*

Lemma 7. *Grids with two fixed values in each of the minigrids M_{12} and M_{21} satisfying the conditions $T_{11} \neq T_{22}$ and $|P_{11} \cap P_{22}| = 0$ such that the single mixed pillar of M_{11} and the single mixed pillar of M_{22} share a single common value always contain intercalate(s).*

Proof. Given that M_{12} and M_{21} each contain two fixed values, M_{11} and M_{22} must each consist of two like pillars and a single mixed pillar in relation to one another's tier sets. By definition, the encoded minigrids share no equal pillars but the single mixed pillars of each minigrid share a single common value. As the single mixed pillars of M_{11} and M_{22} have a common value, the fixed values within minigrids M_{12} and M_{21} have a common value.

Of the remaining vacant cells of M_{12} and M_{21}, there exist four possible completions of each minigrid – two independent completions of each tier of M_{12} and two independent completions of each tier of M_{21}. As $T_{11} \neq T_{22}$, a given tier of M_{11} will always contain two values that reside within a single tier of M_{22}. Resultantly, a tier of M_{12} must then contain the two matching values within a single tier – creating the possibility of a same-stack intercalate in minigrids M_{12} and M_{22}; this is also the case for same-stack intercalates between M_{11} and M_{21}. For the completion of each individual tier of M_{12} or M_{21}, it can be seen that one of the two independent tier completions results in a same-stack intercalate.

The remaining completion for each tier, that avoids such a same-stack intercalate, results in a single same-band intercalate by forming equal pillars within a band; hence, Rodoku grids of this type always contain intercalate(s).

□

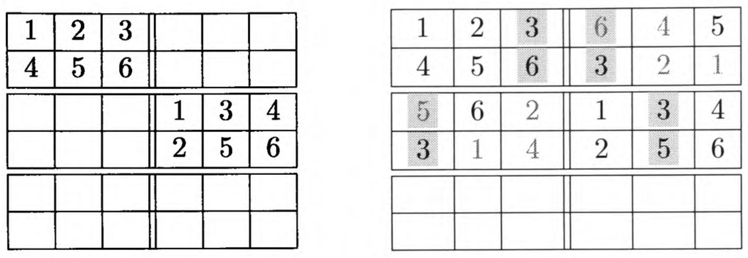


Figure 4.6: *An example of the type of Rodoku grid discussed in Lemma 7.*

Lemma 8. *Grids with two fixed values in each of the minigrids M_{12} and M_{21} satisfying the conditions $T_{11} \neq T_{22}$ and $|P_{11} \cap P_{22}| = 0$ such that the single mixed pillar of M_{11} and the single mixed pillar of M_{22} share no common values always contain intercalate(s).*

Proof. Given that M_{12} and M_{21} each contain two fixed values, M_{11} and M_{22} must each consist of two like pillars and a single mixed pillar in relation to one another's tier sets. By definition, the single mixed pillars of M_{11} and M_{22} contain no common values and resultantly, the fixed values in M_{12} and M_{21} are distinct. There hence exists two independent tier completions for each tier of M_{12} and each tier of M_{21}.

As $T_{11} \neq T_{22}$, a given tier of M_{11} will always contain two values that reside within a single tier of M_{22}. Resultantly, a tier of M_{12} must then contain the two matching values within a single tier – creating the possibility of a same-stack intercalate in minigrids M_{12} and M_{22}; this is also the case for same-stack intercalates between M_{11} and M_{21}. For the completion of each individual tier of M_{12} or M_{21}, it can be seen that one of the two independent tier completions results in a same-stack intercalate. As a result, there exists only a single completion of the upper two bands that does not contain same-stack intercalates; due to the distinct fixed values, this completion of the upper two bands does not contain same-band intercalates either. However, up to permutation of rows there exists one completion of the lower band and this arrangement forces four same-stack intercalates intersecting the lower band of the grid.

□

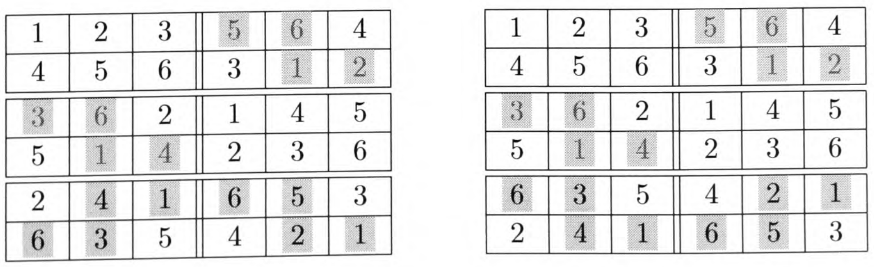


Figure 4.7: *An example of the type of Rodoku grid discussed in Lemma 8.*

Lemma 9. *If no values are fixed as a result of the arrangement of values in M_{11} and M_{22}, and the encoded minigrids contain a 'common pillar', that is $|P_{11} \cap P_{22}| = 1$, then every completion of the upper two bands results in a single same-band intercalate in each of the two bands.*

Proof. As a result of Lemma 1, it is known that same-stack intercalates do not occur in the top two bands of the grid if the grid contains no fixed values outside of M_{11} and M_{22}. By definition, the grids under consideration contain a common pillar within M_{11} and M_{22}. It may be assumed, without loss of generality, that M_{12} is to be completed first. There are four possible ways of completing M_{12}, two of which would result in the symbols of the common pillar being placed within a single pillar of M_{12}, hence resulting in a same-band intercalate over the symbols contained within the common pillar across the top band. The remaining two arrangements within M_{12} result in the symbols of the common pillar being placed in different pillars; one in vertical adjacency to each of the remaining ('uncommon') pillars of the minigrid M_{22}. The pillar of M_{12} that is that vertically-adjacent to the common pillar of M_{22} must hence contain one value from each of the remaining (uncommon) pillars; either of these combinations would result in a pillar that equals a pillar of M_{11} and a same-band intercalate therefore occurs. This also occurs in minigrid M_{21} causing a same-band intercalate across the middle band also. As a result, intercalate(s) always occur in this type of Rodoku grid.

□

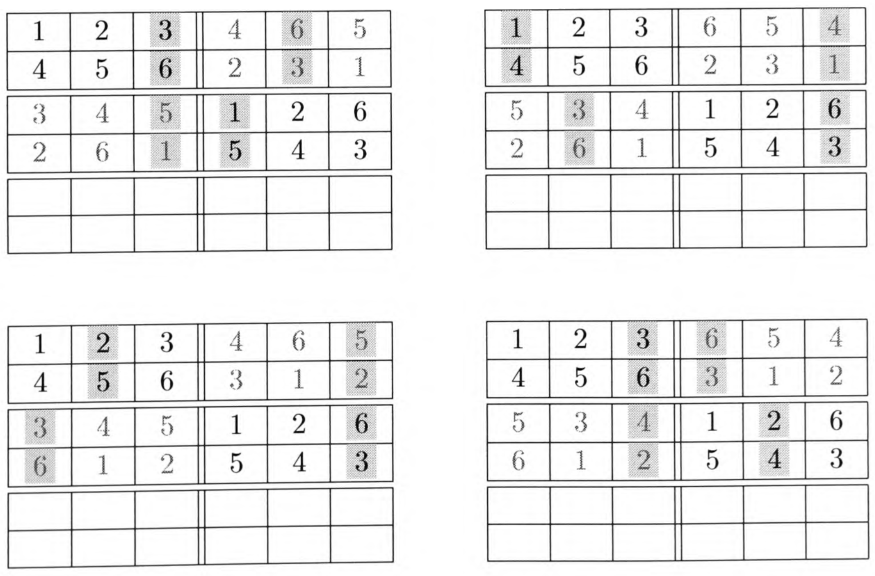


Figure 4.8: *An example of one of the type of Rodoku grid discussed in Lemma 9.*

Lemma 10. *If no values are fixed as a result of the arrangement of values in M_{11} and M_{22}, and the pillar sets of the encoded minigrids are equal, that is $|P_{11} \cap P_{22}| = 3$, then intercalates can be avoided in a quarter of completions of the upper two bands.*

Proof. As a result of Lemma 1, it is known that same-stack intercalates do not occur in the top two bands of the grid if the grid contains no fixed values outside of M_{11} and M_{22}. By definition the grids under consideration share three common pillars; there are four different potential placements for the two values contained within any one of these common pillars that hence hence fix the placement of the remaining values (the choice of which pillar is unimportant, the choice of any pillar will yield the same results). If the symbols of a common pillar are placed together in a pillar of M_{12} or M_{21} then a same-band intercalate is formed and as all pillars of the encoded minigrids are common, this forces a further two same-band intercalates. This type of completion occurs in half of the arrangements. If the symbols of a common pillar are placed one in each of the two permitted pillars of M_{12} or M_{21}, then this also forces the remaining symbols of the two pillars of the encoded minigrids to be placed in different pillars of M_{12} or M_{21}, hence avoiding any same-band intercalates in the upper two bands.

If arrangements avoiding same-band intercalates occur half of the time in M_{12} and half of the time in M_{21}, then no same-stack or same-band intercalates occur in $(\frac{1}{2})^2 = \frac{1}{4}$ of arrangements in the top two bands of cases satisfying $|P_{11} \cap P_{22}| = 3$ (cases 60 and 68).

□

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

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

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

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

Figure 4.9: *An example of one of the type of Rodoku grid discussed in Lemma 10.*

Lemma 11. *If no values are fixed as a result of the arrangement of values in M_{11} and M_{22}, and the pillar sets of the encoded minigrids share no common pillars, that is $|P_{11} \cap P_{22}| = 0$, then intercalates can be avoided in a $(\frac{3}{4})^2 = \frac{9}{16}$ of completions of the upper two bands.*

Proof. As a result of Lemma 1, it is known that same-stack intercalates do not occur in the top two bands of the grid if the grid contains no fixed values outside of M_{11} and M_{22}. As there are no fixed values and $|P_{11} \cap P_{22}| = 0$ there are again four different ways of completing either M_{12} or M_{21}; two of these placements preserve the pillar set of the vertically-adjacent encoded minigrid – hence, as no pillars are common to both encoded minigrids same-band intercalates are avoided. The remaining two placements of values do not preserve the pillar structure of the vertically-adjacent encoded minigrid and hence new 'mixed pillars' are generated. One of the two mixed pillar arrangements creates a pillar set that has no pillars in common with either of the encoded minigrids, again avoiding any same-band intercalates. However, the second arrangement of mixed pillars must equal the pillar set of the horizontally-adjacent minigrid and creates three same-stack intercalates.

Therefore, intercalates can be avoided in $\frac{3}{4}$ of completions of M_{12}, and equally for M_{21}. Hence, $(\frac{3}{4})^2 = \frac{9}{16}$ of arrangements within the upper two minigrids of case 62 avoid any intercalates. □

Thus, by Lemma 10 and Lemma 11, it can be seen that cases 60, 62 and 68 are able to avoid intercalates in some completions of the upper two bands of the grid. It will next be necessary to examine the lower band of the grid for such cases to identify any intercalate-free grids that exist for 6×6 Rodoku.

Lemma 12. *In the case where $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 3$, that is case 68, there are four completions of the top two bands that avoid intercalates. Of these four, two of the grids can be completed such that they avoid intercalates in the lower band also, hence generating four intercalate-free valid Rodoku grids.*

Proof. There are two completions for M_{12} that avoid intercalates in the top band, likewise for M_{21} in the middle band; thus, generating four intercalate free completions of the top two bands. However, as $P_{11} = P_{22}$, two of these four completions will result in $P_{12} = P_{21}$ and would hence generate three same-band intercalates in the lower band. The remaining two completions of the upper four minigrids are such that $|P_{12} \cap P_{21}| = 0$ and as a result, no intercalates result in the lower band.

Therefore, two completions of the upper two bands for case 68 result in intercalate free solutions. There are two ways of arranging the values in the lower band for each of these completions, hence four intercalate-free valid Rodoku grids can be created from encoded grids with the properties $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 3$. □

Lemma 13. *In the case where $T_{11} \neq T_{22}$ and $|P_{11} \cap P_{22}| = 3$, that is case 60, there are four completions of the top two bands that avoid intercalates. Of these four, none of the grids can be completed such that they avoid same-band intercalates in the lower band or avoid creating same-stack intercalates.*

Proof. There are two completions for M_{12} that avoid intercalates in the top band, likewise for M_{21} in the middle band; thus, generating four intercalate free completions of the top two bands. Each of the two completions of M_{12} has a pillar in common with one of the completions of M_{21}

and are also arranged above/below like pillars in the encoded minigrids. As a result of this, the two values to be placed below these pillars in the lower band must contain the same value and hence a same-band intercalate occurs in the lower band in two of the four cases. Also, regardless of the arrangement of values in the lower band (which can be done in four ways in this case), a further two same-stack intercalates are forced due to this layout – one in each stack.

In the remaining two cases, $|P_{12} \cap P_{21}| = 0$ and hence no two pillars in the lower band result in a same-band intercalate. However, when no two pillars of M_{31} and M_{32} are equal, the arrangement of values within tiers are fixed and always force 4 same-stack intercalates. □

Lemma 14. *In the case where $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 0$, that is case 62, there are nine completions of the upper two bands that avoid intercalates. Of these nine, five of the grids can be completed such that they avoid intercalates as a result of the completion of the lower band, thus generating ten intercalate-free valid Rodoku grids.*

Proof. As a consequence of Lemma 11, nine completions of the upper two bands avoid intercalates. Of these nine completions, four of these grids cannot avoid intercalates in the lower band and each contain three same-band intercalates. This is due to remaining possible layout of values in the upper two-bands of the grid, the completions of M_{12} and M_{21} are restricted as completions satisfying $|P_{11} \cap P_{12}| > 0$ and $|P_{21} \cap P_{22}| > 0$ have already been ruled out in Lemma 4 as these would cause intercalates. M_{11} and M_{22} can hence only be constructed such that they have no pillar in common with the horizontally-adjacent minigrid. This forces $|P_{31} \cap P_{32}| = 3$ in four of the five remaining grids. The remaining five each have two potential completions of the lower band, and both completions for each of these five grids avoid intercalates in the lower band. There are hence ten grids generated by each instance of grids of case 62 that are intercalate-free. □

Theorem 3. *The only instances of intercalate-free Rodoku grids are observed when minigrids M_{11} and M_{22} satisfy the conditions $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| \in \{0, 3\}$. Hence, valid intercalate-free completed Rodoku grids can only be created for 5% of encodings created with the Rodoku structure.*

Proof. Lemma 12 shows that every grid satisfying $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 3$ can generate four intercalate-free completions and Lemma 14 shows that grids satisfying $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 0$ can each generate ten intercalate-free completions. As $1\frac{2}{3}\%$ of encoded grids satisfy $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 3$ and $3\frac{1}{3}\%$ of encoded grids satisfy $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 0$, only $3\frac{1}{3}\% + 1\frac{2}{3}\% = 5\%$ of encoded grids can be extended to intercalate-free Rodoku grids. □

Theorem 4. *The total number of intercalate-free Rodoku puzzles is 207360. Enumeration calculations for Rodoku reveal that there are 28,200,960 Rodoku grids [13]. Hence, the percentage of intercalate-free grids is 0.735%.*

Proof. These values can be calculated by extension of Theorem 3, as such:

$$\frac{(4 \times 6!^2 \times 1\frac{2}{3}) + (10 \times 6!^2 \times 3\frac{2}{3})}{100} = 207,360.$$

The values 4 and $1\frac{2}{3}$ relate to the number of intercalate-free extension to a valid grid for encoded grids satisfying $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 3$ and the percentage of encoded grids of this type respectively. Likewise for 10 and $3\frac{1}{3}$, 10 is the number of intercalate free extensions and $3\frac{1}{3}$ is the percentage of encoded grids satisfying $T_{11} = T_{22}$ and $|P_{11} \cap P_{22}| = 3$. Therefore, the percentage of intercalate-free grids relative to the total number of grids is 0.735% □

4.1.4 Probability of Recovery Calculations from Grids with Varied Numbers of Intercalates

The calculations below detail the probabilities of not recovering the original Rodoku grid due to the existence of intercalates. The calculations consider both disjoint and intersecting intercalates.

Single Intercalate Intersecting an Encoded Minigrid

$$\begin{aligned}\text{P(cannot recover grid)} &= \text{P(all 4 cells of single intercalate erased)}\\ &= [\text{P(one cell of intercalate being erased)}]^4\\ &= [\text{P(one cell being erased)}]^4\end{aligned}$$

Let P(one cell being erased) $= p$; then, P(cannot recover grid) $= p^4$; and, P(can recover grid) $= 1 - p^4$.

Two Disjoint Intercalates Intersecting one of the Encoded Minigrids

$$\begin{aligned}\text{P(cannot recover grid)} &= 1 \text{ - P(neither intercalate erased)}\\ &= 1 - (1 - p^4)^2\\ &= 2p^4 - p^8\end{aligned}$$

Two Intersecting Intercalates (one cell intersection), Intersecting an Encoded Minigrid

$$\begin{aligned}\text{P(cannot recover grid)} &= 2[\text{P(one intercalate erased AND other recovered)}]\\ &\quad + \text{P(both intercalates erased)}\\ &= 2p^4(1 - p^3) + p^7\\ &= 2p^4 - p^7\end{aligned}$$

Three Disjoint Intercalates Intersecting an Encoded Minigrid

$$\begin{aligned}\text{P(cannot recover grid)} &= 1 \text{ - P(no intercalates erased)}\\ &= 1 - (1 - p^4)^3\\ &= 3p^4 - 3p^8 + p^{12}\end{aligned}$$

Three Intersecting Intercalates Intersecting an Encoded Minigrid

$$\begin{aligned}\text{P(cannot recover grid)} &= 3[\text{P(one intercalate erased AND two recovered)}]\\ &\quad + 3[\text{P(two intercalates erased AND one recovered)}]\\ &\quad + \text{P(all three erased)}\\ &= 3p^4(1-p^3)^2 + 3p^7(1-p^3) + p^{10}\\ &= 3p^4 - 3p^7 + p^{10}\end{aligned}$$

From the results determined above, Table 4.3 was constructed. It gives the probability of not recovering the grid containing various numbers of intercalates, for a range of probabilities of single cell erasure.

Probability of Not Recovering Grid for Various Numbers of Intercalates					
P(single erasure)	Number of Intercalates (2 × 2 sub-squares)				
	1	2		3	
p	p^4	$2p^4 - p^8$	$2p^4 - p^7$	$3p^4 - 3p^8 + p^{12}$	$3p^4 - 3p^7 + p^{10}$
0.2	0.0016	0.0032	0.0032	0.0048	0.0048
0.25	0.0039	0.0078	0.0078	0.0117	0.0115
0.333	0.0123	0.0245	0.0242	0.0366	0.0357
0.4	0.0256	0.0505	0.0496	0.0749	0.0720
0.5	0.0625	0.1211	0.1760	0.1760	0.1650
0.6	0.1296	0.2424	0.2312	0.3406	0.3109
0.667	0.1975	0.3560	0.3365	0.4832	0.4344
0.75	0.3164	0.5327	0.4993	0.6806	0.6051
0.8	0.4096	0.6514	0.6095	0.7942	0.7070

Table 4.3: *Probabilities of not recovering a grid for various numbers of intercalates.*

Each probability detailed in Table 4.3 could be considered to be misleading however. The probabilities given relate to the likelihood of an intercalate being erased, but for each erased 2×2 intercalate there exist only two different completions – one of which is correct. As a consequence of this, if we were to guess the order of values within a single intercalate, we would be correct in half of the cases, therefore reducing the probabilities given in Table 4.3.

4.2 Intercalate Free Puzzles

Of the $6!^2$ encoded Rodoku grids that can be created by the outline scheme detailed in Chapter 3, only 5% of the encoded grids possess completions that are able to completely avoid intercalates. Cases 62 and 68, as outlined above, are the only cases of encoded Rodoku grid that can be extended to valid grids so as to avoid intercalates, although not all completions of these grids will ensure avoidance. The types of completions that avoid intercalates completely all possess a special property; the property being that the grids could be seen as being constructed by interleaving

3×3 Latin squares into a Rodoku grid structure, whilst avoiding same-band intercalates. An example of such a grid is shown in Figure 4.10.

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

Figure 4.10: *An example of an intercalate-free Rodoku grid.*

As a result, the intercalate-free grids are by definition constructed entirely from 3×3 Latin sub-squares, hence reducing the grids robustness against erasures occurring over these sub-squares and also over 2×3 or 3×2 Latin sub-rectangles. Further investigation, and possibly computation, would be required to ascertain whether grids of this type are more robust against cell erasures than Rodoku grids containing intercalates, but the improvements of utilising intercalate-free solutions only in a coding scheme is not likely to be of significant benefit. This is due to the number of intercalate-free solutions being very small, hence limiting the number of messages that could be encoded and also due to intercalate-free puzzles demonstrating increased numbers of Latin sub-squares and sub-rectangles.

The 5% value for encoded Rodoku grids, from Theorem 3, is a rather misleading figure. We should instead consider the proportion of intercalate-free grids relative to the total number of Rodoku grids described in Theorem 4. Theorem 4 shows there to be 207360 intercalate-free valid Rodoku grids; 0.735% of the total number of valid Rodoku grids.

Comparing this percentage to the percentage of order 6 Latin squares that are intercalate-free, only a small difference is observed. McKay and Wanless [24] calculated that the number of intercalate-free order 6 Latin squares is $40 \times 6! \times 5!$, and total number of order 6 Latin squares is $9408 \times 6! \times 5!$ [24], hence the percentage of intercalate-free order 6 Latin squares is 0.425%. These results show that extra 'minigrid' constraint in Rodoku does not dramatically increase the number of intercalate-free solutions of a structure. This indicates that the use of a further constrained Sudoku (or Rodoku) type puzzle should only be approached if the further constraint is deliberately designed to avoid the formation of intercalates. These values however, relate only to intercalate-free grids and although identification of intercalate-free grids is desirable, grids containing few intercalates could also feasibly be used.

4.3 Evaluation of Current Scheme

As a result of the above investigation it can be concluded that the Rodoku-based scheme is currently not particularly robust in dealing with erasures, and it is expected that these issues would also affect the Sudoku-based scheme, possibly to a greater extent when considering the increase in the number of cell inter-relationships. The problem can be attributed in the main part to the existence of intercalates within the Rodoku structure, where only 5% of grids with initial encodings can be completed to a valid Rodoku grid such that they contain no intercalates. For the remaining 95% of grids with initial encodings, completion to an intercalate-free Rodoku grid is not possible; all completions contain intercalates – the erasure of all of these four cells during transmission of the completed grid would result in a grid in which erasures cannot be resolved. Furthermore, in terms of completed grids intercalate-free Rodoku grids account for only 0.735% of the total number of possible completed Rodoku grids. As a result of the percentage of encodings with intercalate-free completions and percentage of intercalate-free valid grids, it is likely that the use of intercalate-free structures alone with the current scheme would be inadvisable due to the limitations imposed by using so few grids.

In addition to the scarcity of intercalate-free structures, further problems also stem from the fact that the values within the intercalate-free valid grids are arranged in such a way that the grid contains four 3×3 Latin sub-squares, the erasure of any one of the Latin sub-squares would result in a grid that again cannot be recovered. As a result of the existence of 3×3 Latin sub-squares being contained in all intercalate-free Rodoku grids, these grids may also contain more 2×3 and 3×2 Latin sub-rectangles which can also potentially result in failure to recover transmitted grids.

Construction of a simulation for Sudoku that investigated the potential improvements offered by intercalate-free grids showed some surprising results. This simulation follows on from that discussed previously in Section 3.3.2. The simulation compared the results of the original simulation, in which a sample of grids were subjected to a number of cell erasure and solving techniques were employed to attempt to recover the grid, with single intercalate-free Sudoku grids subjected to the same conditions. The intercalate-free Sudoku grids used in the simulation are shown in Figure 4.11.

Each of the four grids shown in Figure 4.11 are intercalate-free but contain varied numbers of $3 \times 3, 2 \times 3$ and 3×2 Latin sub-squares and sub-rectangles. The first three grids were created by extending the properties of an intercalate-free Rodoku puzzles, identified earlier in the chapter, to an order 9 grid and the fourth grid is created by a construction that would not eliminate intercalates within 6×6 Rodoku.

The results of this simulation show that grids created in such a way as grids 1, 2 and 3 are more likely to include 2×3 and 3×2 Latin sub-rectangles and the increased number of such sub-structures resulted in a decrease in the percentage of grids recovered when compared to the original simulation on a sample of Sudoku grids. Grid 4 contains no 3×3 Latin sub-squares and

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

Grid 1

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

Grid 2

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

Grid 3

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

Grid 4

Figure 4.11: *Grids 1, 2, 3 and 4 that were used in the simulation.*

fewer 2×3 and 3×2 Latin sub-rectangles in comparison to grids 1, 2 and 3; as a result, this grid appears more robust at low probabilities than all other grids, but at higher probabilities the original sample of Sudoku grids is more robust against erasures. The results are demonstrated in the chart given in Figure 4.12.

Combining the problems born from the existence of intercalates and Latin sub-squares in Rodoku, which are also likely in Sudoku, with the relatively low rate of 0.21579, continuation with the current scheme is advisable only with dramatic improvements. There are potentially improvements to be made by incorporating a further-constrained Sudoku variant puzzle as opposed to the standard Sudoku used in the current scheme; however the type of constraint applied is of importance. It was shown in Section 4.2 that the reduction in the number of intercalate-free puzzles by introducing a minigrid constraint to a 6×6 Latin square was small, although it must be noted that these values consider only grids containing no intercalates; grids containing a small number of intercalates are likely to also be effective for erasure correction. Continuation with the scheme incorporating standard Rodoku or Sudoku is hence inadvisable, but there is scope to consider a further constrained Rodoku or Sudoku puzzle, provided the further constraint is such

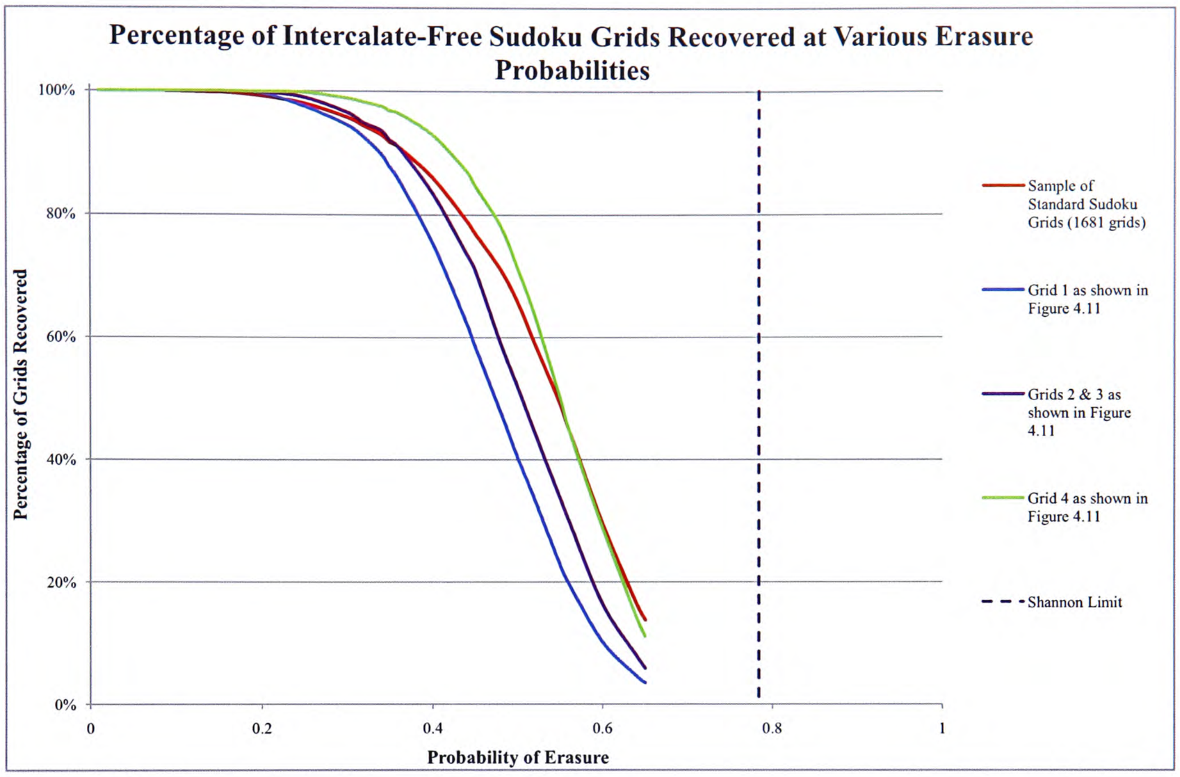


Figure 4.12: *Graph detailing probability of erasure against number of recoveries for various Sudoku grids.*

that it dramatically reduces the number of intercalates contained within a completed puzzle.

4.4 Remarks on Sudoku Scheme

4.4.1 Limitations of Sudoku Scheme

Although an outline erasure correction scheme has been constructed that incorporates Sudoku, this does not provide any guarantees that it is practically feasible. In order to be considered practically feasible, Sudoku codes would need to compete with established codes, such as Fountain codes, which are near-optimal, usually operating within 5% of optimality [20].

A code is considered optimal if its rate is very close to the Shannon capacity of the communication channel. Chapter 3 used mutual information to show that the channel capacity for any

r-ary erasure channel is $1 - p$ where p is the probability of symbol erasure, and also calculated the rate of the Sudoku scheme to be:

$$\frac{\log_9(9!^3)}{81} = 0.21579.$$

Using these results, it can be seen that Sudoku codes must perform at erasure probabilities approaching $1 - 0.21579 = 0.78421$ to be considered practically feasible. This indicates that the scheme would need to be able to recover grids from which around sixty cells have been erased; this is extremely unlikely as specifically designed Sudoku puzzles are rare with so few givens – the smallest number of givens for a uniquely completable puzzle is currently seventeen [22].

The low rate of the scheme has an irreparable consequential effect on the practical feasibility of the potential code; the Sudoku structure cannot tolerate random symbol erasure to the extent required. The low rate can predominantly be attributed to the Sudoku structure being based on permutations as this limits the number of possible encodings that can be achieved with the input alphabet. The scheme has $(9!)^3$ possible initial encodings into a 9×9 grid, where each minigrid on the leading diagonal is encoded with a permutation of the values $\{0, 1, \ldots, 9\}$ as outlined in Section 3.2. The use of permutation encoding, although necessary for the Sudoku scheme, is limiting. It would be possible to have up to 9^{27} initial encodings with the given input alphabet and erasure channel as every one of the 27 cells used for the encoding could take any value from $\{0, 1, \ldots, 9\}$ if permutations were not obligatory – this would improve the rate from 0.21579 to one third.

Chapter 4 identifies a further limitation of the Sudoku structure for erasure correction – the existence of intercalates and Latin sub-structures within the larger structure. The erasure of all cells of a Latin sub-structure renders the grid incompletable and as a result, the original information encoded into the Sudoku grid may not be recovered. The chapter also identifies that simple algorithmic constructions to reduce the number of intercalates or small Latin sub-structures typically resulted in an increased existence of larger sub-structures, and hence there is no simple means of controlling the issues engendered by Latin sub-structures.

4.4.2 Potential Improvements

The most significant limitation of the Sudoku coding scheme is that its structure is based on permutations, limiting the total number of possible encodings. Consequently, a favourable erasure correction scheme will not be based on permutations. Although the removal of the permutation basis offers a lot more freedom when encoding initial data into a grid and is likely to significantly improve the rate, the removal of the strong cell inter-relationships of Sudoku make recovery of missing data more difficult. As it is desirable to maintain a puzzle-type structure (and a puzzle-type 'feel') to any improved coding scheme, so that a variety of solving strategies may be employed to recover information, additional constraints from Section 2.2.2 are considered. The most favourable of the further constraints, to be used for a puzzle not based on permutations are those using *meta-information*; that is information outside the grid that communicates clues as to the placement of values within the grid. Meta-information (also referred to as meta-data) is seen in both Skyscraper Sudoku and Outside Sudoku in Section 2.2.2. The use of meta-data will allow an entire grid to be encoded with any symbols from the input alphabet, and meta-data will be used in the proposed new structure to aid recovery post-transmission.

In order to improve the ability of a scheme to cope with intercalates and Latin sub-structures, additional meta-data may be considered; this would allow the scheme to be more robust in its recovery of erasures as the amount of relevant meta-data for individual cells would be increased. Identifying and colouring sub-structures before transmission could also be considered; whereby a search for sub-structures would be conducted pre-transmission and different colourings applied to each value within the sub-structure, similar to the colouring employed for Even-Odd Sudoku introduced in Section 2.2.2. Although both options have an undesirable effect on the rate, the use of additional data that aid recovery is likely to be the more favourable option as it is less work-intensive than identifying sub-structures pre-transmission, and avoids any issues associated with transmitting colourings over an r-ary erasure channel. It would be expected that the number of Latin sub-structures would be fewer after the removal of the permutation property as there are far fewer restrictions on the placement of values.

Chapter 5

GLUV Codes: An Introduction

Following the identification of the various shortcomings of Sudoku as a practically feasible structure for incorporation into an erasure correction scheme, this chapter of the thesis introduces a new combinatorial structure, termed GLUV. The individual components of GLUV are outlined, as are the improvements they offer over Sudoku for use as part of an erasure correction scheme.

5.1 GLUV: Grid with Length and Up-Down Vectors

An investigation into appropriate meta-data allowed for the introduction of meta-data vectors into GLUV; these vectors communicate information about a particular row or column of a grid, where each cell of the grid may contain any value from a given input alphabet. Investigation of different potential meta-data vectors resulted in the choice of two sets of vectors that would be used to aid recovery of missing data post-transmission. Traditional erasure-correction often involves the use of block codes with additional parity check symbols being applied to each block, the meta-data vectors within GLUV provide less rigid information than traditional parity checks, however they can often be utilised in a variety of solution strategies to help recover erased data, the different meta-data vectors can each be used in different ways to endeavour to recover the grid. However, the increase in flexibility of use of the redundant information in GLUV is likely to be partnered by a more difficult decoding algorithm than with traditional erasure-correction codes. The meta-data vector components of GLUV are discussed in detail in the subsequent sections.

A GLUV puzzle is made up of three separate components; the grid itself into which information is encoded, a set of row and column *length vectors* and a set of row and column *up-down vectors*.

5.1.1 GLUV Grid

The GLUV grid is an $n \times n$ grid in which each cell may contain any value from an input alphabet $S = \{0, 1, \ldots, s-1\}$. It is into the cells of the GLUV grid that the initial data are encoded, and therefore all grid symbols must be recovered post-transmission to successfully recover the encoded message. The meta-data, made up of length vectors and up-down vectors, are then calculated from the grid encoding and are used post-transmission to attempt to recover the missing grid symbols using a variety of solution strategies.

Encoding in this way allows far greater flexibility than is the case for Sudoku – where each row, column and mini-grid is required, by definition, to contain a permutation of all symbols of the alphabet. Also, within Sudoku, the number of symbols to be used is defined by the grid size; the devised new structure allows the number of symbols to vary for the given array size, hence the number of symbols can be chosen for a specific array size to maximise the rate of the code.

5.1.2 Length Vectors

A length vector communicates the number of occurrences of each symbol of the input alphabet in a given row or column of the grid. For an $n \times n$ grid over s symbols, a length vector would contain s elements, each relating to one specific symbol of the grid input alphabet. Each of the length vector elements could take any value from the alphabet L of size $n+1$, where $L = \{0, 1, \ldots, n\}$, as there may be between zero and n occurrences of a grid symbol in a given row or column. Each position within the vector refers to a specific symbol of the alphabet and the number contained within the position communicates the number of occurrences of that symbol. That is, the i^{th} element of the vector details the number of occurrences of the symbol $i-1$. The sum of all elements of a length vector must also equal n as there are n grid symbols in a row or column.

Consider a 9×9 GLUV grid over 4 symbols; an appropriate length vector for this grid is $\begin{pmatrix} 2 & 4 & 1 & 2 \end{pmatrix}$. The length vector communicates that the number of occurrences of the symbols 0, 1, 2 and 3 are two, four, one and two respectively for the row or column of the grid associated with the length vector.

It can be said that the length vector acts in a similar way to the permutation property in Sudoku. It communicates information about the placement of values, although these now vary from row to row and column to column and individual vectors communicate the specifics for each row or column.

5.1.3 Up-Down Vectors

The length vector alone does not communicate enough information to recover significant losses, as it is difficult to differentiate the correct placement of erased values when the only information for recovery is the number of required occurrences of a given symbol. The second vector to be considered is a binary up-down vector; this vector communicates the relationship between two horizontally or vertically adjacent cells. The up-down vector communicates the relative increases and decreases between successive grid symbols; a '0' represents that the next successive element is less than or equal to the preceding element and a '1' represents a strict increase from one element to the next. That is, the i^{th} element of the up-down vector communicates the increase or decrease between elements i and $i+1$ within the grid.

The incorporation of the up-down vector allows further inferences to be made when attempting to recover erased grid symbols post-transmission. It also allows some occurrences of Latin substructures to be overcome as the vector describes the relationship between an erased value and its surrounding grid values. An example of a complete 9×9 grid over 4 symbols is given in Figure 5.1.

$$
\begin{array}{ccccccccccc}
 & \begin{bmatrix}0\\1\\0\\0\\0\\1\\0\\0\end{bmatrix} & \begin{bmatrix}1\\0\\0\\0\\1\\0\\1\\0\end{bmatrix} & \begin{bmatrix}1\\0\\0\\0\\0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\1\\1\\0\\0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\1\\0\\0\\0\\1\end{bmatrix} & \begin{bmatrix}0\\0\\0\\1\\0\\1\\0\\1\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\1\\0\\1\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\1\\1\\0\\0\\0\end{bmatrix} & \\
\begin{bmatrix}1&0&1&0&1&0&0&1\end{bmatrix} & 1 & 2 & 1 & 3 & 0 & 2 & 1 & 0 & 1 & \begin{bmatrix}2&4&2&1\end{bmatrix} \\
\begin{bmatrix}1&0&0&0&1&0&0&1\end{bmatrix} & 1 & 3 & 3 & 1 & 0 & 2 & 1 & 0 & 1 & \begin{bmatrix}2&4&1&2\end{bmatrix} \\
\begin{bmatrix}0&1&0&0&0&0&0&1\end{bmatrix} & 3 & 1 & 2 & 2 & 2 & 1 & 1 & 1 & 2 & \begin{bmatrix}0&4&4&1\end{bmatrix} \\
\begin{bmatrix}0&1&1&0&0&1&0&0\end{bmatrix} & 2 & 1 & 2 & 3 & 1 & 0 & 3 & 2 & 1 & \begin{bmatrix}1&3&3&2\end{bmatrix} \\
\begin{bmatrix}0&1&1&0&0&0&0&1\end{bmatrix} & 0 & 0 & 1 & 3 & 3 & 2 & 1 & 0 & 2 & \begin{bmatrix}3&2&2&2\end{bmatrix} \\
\begin{bmatrix}1&0&0&1&0&0&1&0\end{bmatrix} & 0 & 1 & 1 & 1 & 3 & 1 & 0 & 3 & 3 & \begin{bmatrix}2&4&0&3\end{bmatrix} \\
\begin{bmatrix}0&0&0&0&1&0&0&1\end{bmatrix} & 3 & 1 & 1 & 1 & 1 & 3 & 2 & 0 & 3 & \begin{bmatrix}1&4&1&3\end{bmatrix} \\
\begin{bmatrix}0&0&0&0&0&1&0&1\end{bmatrix} & 3 & 3 & 3 & 2 & 1 & 0 & 2 & 1 & 2 & \begin{bmatrix}1&2&3&3\end{bmatrix} \\
\begin{bmatrix}0&0&1&0&0&0&0&0\end{bmatrix} & 3 & 3 & 2 & 1 & 3 & 2 & 1 & 0 & 0 & \begin{bmatrix}2&2&2&3\end{bmatrix} \\
 & \begin{bmatrix}2\\2\\1\\4\end{bmatrix} & \begin{bmatrix}1\\4\\1\\3\end{bmatrix} & \begin{bmatrix}0\\4\\3\\2\end{bmatrix} & \begin{bmatrix}0\\4\\2\\3\end{bmatrix} & \begin{bmatrix}2\\3\\1\\3\end{bmatrix} & \begin{bmatrix}2\\2\\4\\1\end{bmatrix} & \begin{bmatrix}1\\5\\2\\1\end{bmatrix} & \begin{bmatrix}5\\2\\1\\1\end{bmatrix} & \begin{bmatrix}1\\3\\3\\2\end{bmatrix} &
\end{array}
$$

Figure 5.1: *An example of a completed order 9 GLUV structure, with its associated meta-data.*

5.2 GLUV: Efficiency, Size and Transmission

5.2.1 Rate of GLUV Scheme

The efficiency of a coding scheme that incorporates GLUV is dependent on the size of the grid and number of symbols in the grid input alphabet, as these values vary the *rate* of the coding scheme varies. The rate of a GLUV scheme in which the data to be transmitted is encoded into the entirety of the grid is:

$$R = \frac{n^2 \log_2 s}{n^2 \lceil \log_2 s \rceil + 2ns \lceil \log_2 (n+1) \rceil + 2n(n-1)}$$

where the numerator represents the number of data bits encoded in the GLUV grid and the denominator represents the number of bits required to represent the GLUV as a whole (meta-data included) in binary form. More specifically, the numerator $n^2 \log_2 s$ communicates the number of bits encoded into the grid itself as there are n^2 grid cells and an alphabet of s symbols, requiring $log_2 s$ bits in each cell. The denominator is composed of three components, one for each component of GLUV; $n^2 \lceil \log_2 s \rceil$ again relates to the grid itself, where the ceiling function is used to round up to the nearest number of bits required to represent the grid. The second component $2ns \lceil \log_2 (n+1) \rceil$ relates to the length vectors, as there are two sets of n length vectors (row and column), each of length s; each of the length vector symbols represents a symbol from the alphabet $\{0, 1, \ldots, n\}$ and as a result $\lceil \log_2 (n+1) \rceil$ bits are required to represent the length vectors. The third component $2n(n-1)$ relates to the up-down vectors which are already binary, there are two sets of n up-down vectors, each containing $n-1$ bits. When all components are combined, the rate value communicates the amount of non-redundant information in the GLUV structure.

Generally there is little practical interest in codes with very low rates, however powerful the error or erasure correction capability. The rates of all GLUV codes are fairly small. The rate of the GLUV coding scheme peaks when $s = 2^a, a \in \mathbb{Z}$ and $n = 2^b - 1, b \in \mathbb{Z}$; the optimum value of a varies with grid size and the optimum value of b varies for different alphabet sizes. The optimum choices for grid size and number of symbols are highlighted in Figures 5.2 and 5.3.

It can be seen from these plots that if the number of symbols in the input alphabet is kept constant, the rate generally improves as the grid size increases. More importantly, it can also be noted that for all grids below 50×50 in size, the optimum number of symbols is 4 (for small order grids, up to order 20 on graph) or 8 (for larger order grids, order 25 and above on graph). This indicates that the most favourable practical values would appear to be $n = 15$ with $s = 4$ (rate $= 0.3333$) or $n = 31$ with $s = 8$ (rate $= 0.3991$). Larger grid sizes or number of symbols are undesirable as recovering missing data in larger grids is likely to be computationally more difficult.

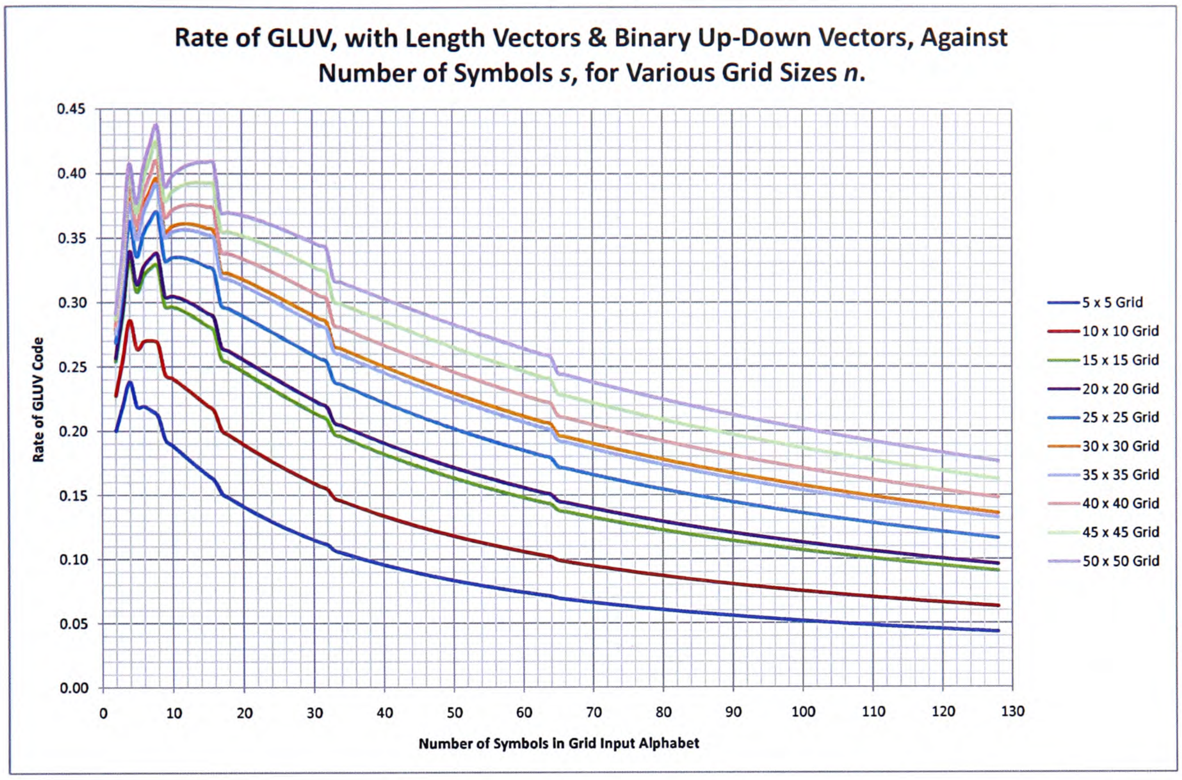


Figure 5.2: *Graph of GLUV rate plotted against input alphabet size, for a variety of grid sizes.*

5.2.2 Transmission

The current GLUV coding scheme is polyalphabetic; each component of the scheme has a different alphabet. As a result, transmission over an erasure channel is difficult as the information would need to be transmitted over an r-ary erasure channel in which r equals the size of the largest alphabet of the 3 components. This is not efficient as the remaining two alphabets are smaller and do not require a larger alphabet for transmission. It is hence desirable to transmit all components using a single alphabet, as this will allow erasures to occur more uniformly and eases the difficulties in transmitting a polyalphabetic structure.

The conversion from polyalphabetic GLUV to monoalphabetic GLUV is achieved by representing the symbols of components with larger input alphabets by a string of symbols from the new, smaller, single alphabet and likewise representing r-tuples of symbols of components with smaller input alphabets with a single symbol from the new, single alphabet.

For example, consider a polyalphabetic GLUV with $n = 15$ and $s = 4$. The grid input alphabet of this GLUV is $S = \{0, 1, 2, 3\}$, the length vector alphabet $L = \{0, 1, \ldots, 15\}$ and therefore $|L| = 16$, and the up-down vector alphabet is binary. As $4^2 = 16$ the polyalphabetic GLUV can be converted to its monoalphabetic counterpart, this is achieved by allowing every

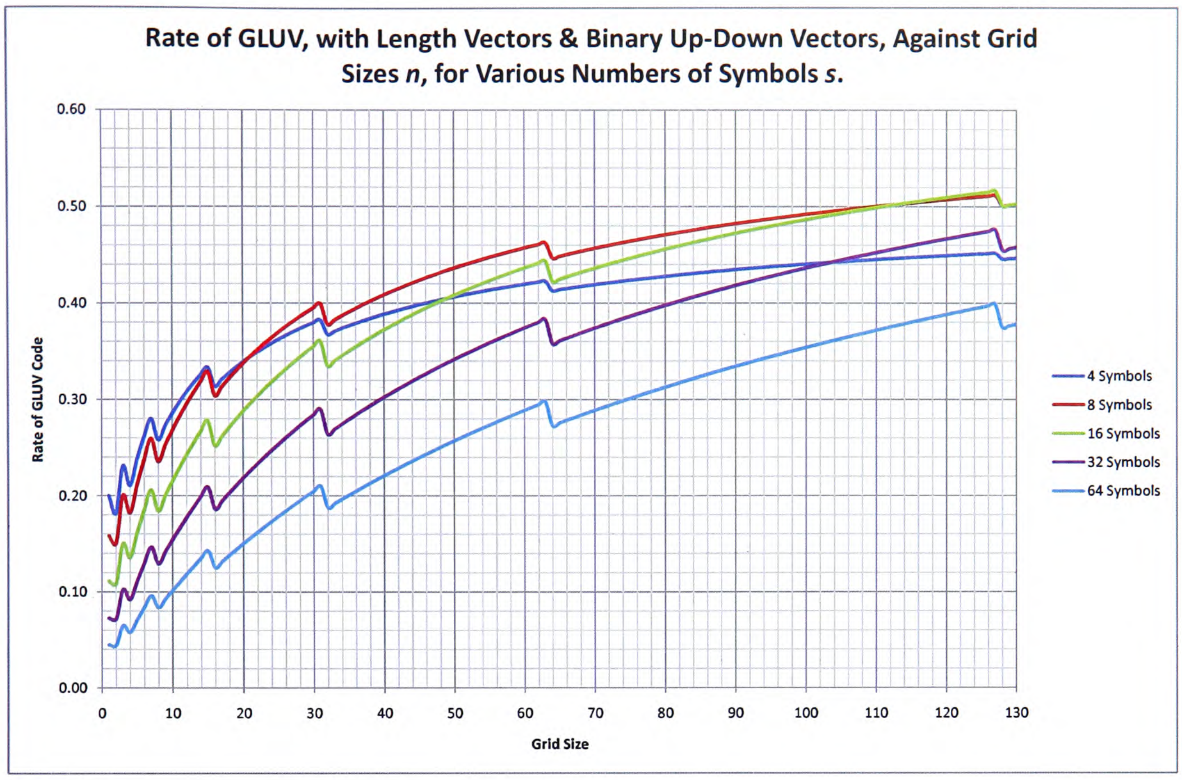


Figure 5.3: *Graph of GLUV rate plotted against grid size, for a variety of input alphabet sizes.*

symbol in the length vectors to be represented by a pair of symbols from $S = \{0, 1, 2, 3\}$, using quaternary counting to represent the values zero through to fifteen. For example, the quaternary value $21 = 2 \times 4^1 + 1 \times 4^0 = 8 + 1 = 9$ in decimal. Similarly, every pair of bits in the up-down vector can be represented by a single value from the alphabet S.

Conversion from polyalphabetic GLUV to monoalphabetic GLUV can only be achieved when $n = 2^i - 1$ and $s = 2^j$ such that $n + 1 = s^k, k \in \mathbb{Z}$, this allows the length vector symbols to each be represented by a k-tuple of symbols from the alphabet $S = \{0, 1, \ldots, s-1\}$ of size s. It is also desirable that $\frac{n-1}{j}$ is integer as this means that the division of each up-down vector into j-tuples of bits is exact and hence has no impact on the rate.

The rate of a scheme incorporating monoalphabetic GLUV as outlined has rate;

$$R = \frac{n^2}{n^2 + 2ns\log_s(n+1) + 2n\left\lceil\frac{n-1}{j}\right\rceil}$$

where $j = \log_2 s$. The only values of n and s satisfying these conditions for $n < 50$ are $n = 15$ and $s = 4$ which was also favourable for polyalphabetic GLUV. A monoalphabetic GLUV coding scheme with such parameters has a rate of one third. This rate equals that of its polyalphabetic counterpart as all conversions into the single alphabet are exact, the main incentive for conversion being to make best use of the erasure channel over which the information is transmitted as opposed to optimising the rate.

5.3 GLUV: Outline Erasure Correction Scheme

$$
\begin{array}{ccccccccccccccccc}
 & \begin{bmatrix}1\\1\\1\\1\\0\\0\\2\end{bmatrix} & \begin{bmatrix}1\\0\\2\\2\\2\\2\\0\end{bmatrix} & \begin{bmatrix}0\\3\\0\\3\\0\\2\\3\end{bmatrix} & \begin{bmatrix}0\\1\\1\\0\\2\\2\\1\end{bmatrix} & \begin{bmatrix}3\\2\\1\\2\\1\\1\\1\end{bmatrix} & \begin{bmatrix}0\\1\\1\\2\\1\\1\\0\end{bmatrix} & \begin{bmatrix}2\\1\\0\\2\\2\\1\\2\end{bmatrix} & \begin{bmatrix}2\\2\\1\\0\\1\\2\\2\end{bmatrix} & \begin{bmatrix}0\\2\\2\\1\\2\\2\\3\end{bmatrix} & \begin{bmatrix}1\\0\\1\\1\\0\\2\\2\end{bmatrix} & \begin{bmatrix}1\\0\\1\\2\\2\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\2\\3\\0\\0\\1\end{bmatrix} & \begin{bmatrix}2\\1\\1\\0\\1\\1\\2\end{bmatrix} & \begin{bmatrix}2\\3\\0\\0\\1\\1\\0\end{bmatrix} & \begin{bmatrix}0\\2\\0\\0\\2\\0\\2\end{bmatrix} & \\
\begin{bmatrix}2&2&2&1&1&0&1\end{bmatrix} & 1 & 2 & 1 & 3 & 0 & 2 & 1 & 0 & 1 & 1 & 3 & 3 & 1 & 0 & 2 & \begin{bmatrix}03&12&03&03\end{bmatrix} \\
\begin{bmatrix}1&2&2&0&0&2&1\end{bmatrix} & 1 & 0 & 1 & 3 & 1 & 2 & 2 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 2 & \begin{bmatrix}01&13&12&01\end{bmatrix} \\
\begin{bmatrix}0&2&0&1&2&0&1\end{bmatrix} & 3 & 1 & 0 & 3 & 2 & 1 & 0 & 0 & 1 & 3 & 3 & 2 & 1 & 0 & 2 & \begin{bmatrix}10&10&03&10\end{bmatrix} \\
\begin{bmatrix}2&1&0&2&0&0&1\end{bmatrix} & 0 & 1 & 1 & 1 & 3 & 1 & 0 & 3 & 3 & 3 & 1 & 1 & 1 & 1 & 3 & \begin{bmatrix}02&20&00&11\end{bmatrix} \\
\begin{bmatrix}1&0&0&0&2&3&0\end{bmatrix} & 2 & 0 & 3 & 3 & 3 & 3 & 2 & 1 & 0 & 2 & 1 & 2 & 3 & 3 & 2 & \begin{bmatrix}02&02&11&12\end{bmatrix} \\
\begin{bmatrix}2&0&1&1&0&2&2\end{bmatrix} & 1 & 3 & 2 & 1 & 0 & 0 & 1 & 1 & 2 & 2 & 0 & 3 & 2 & 3 & 0 & \begin{bmatrix}10&10&10&03\end{bmatrix} \\
\begin{bmatrix}0&2&0&2&2&1&0\end{bmatrix} & 3 & 1 & 1 & 3 & 1 & 1 & 0 & 3 & 0 & 3 & 1 & 0 & 3 & 3 & 0 & \begin{bmatrix}10&11&00&12\end{bmatrix} \\
\begin{bmatrix}2&0&1&0&1&1&0\end{bmatrix} & 1 & 3 & 2 & 2 & 2 & 2 & 3 & 1 & 0 & 0 & 2 & 2 & 3 & 3 & 0 & \begin{bmatrix}03&02&12&10\end{bmatrix} \\
\begin{bmatrix}1&1&1&1&2&2&2\end{bmatrix} & 3 & 0 & 3 & 1 & 2 & 0 & 2 & 0 & 2 & 3 & 0 & 3 & 2 & 3 & 0 & \begin{bmatrix}11&01&10&11\end{bmatrix} \\
\begin{bmatrix}2&2&1&1&0&0&0\end{bmatrix} & 2 & 3 & 0 & 2 & 0 & 0 & 3 & 0 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & \begin{bmatrix}10&00&11&12\end{bmatrix} \\
\begin{bmatrix}0&2&0&2&1&1&0\end{bmatrix} & 1 & 0 & 0 & 2 & 1 & 1 & 1 & 2 & 0 & 0 & 3 & 2 & 3 & 3 & 1 & \begin{bmatrix}10&11&03&03\end{bmatrix} \\
\begin{bmatrix}2&2&0&2&2&0&0\end{bmatrix} & 0 & 3 & 1 & 3 & 1 & 1 & 1 & 3 & 1 & 3 & 1 & 1 & 1 & 0 & 0 & \begin{bmatrix}03&20&00&10\end{bmatrix} \\
\begin{bmatrix}2&1&0&0&2&1&0\end{bmatrix} & 0 & 3 & 0 & 0 & 3 & 2 & 2 & 1 & 0 & 1 & 1 & 1 & 2 & 2 & 0 & \begin{bmatrix}11&10&10&02\end{bmatrix} \\
\begin{bmatrix}0&0&3&0&2&1&1\end{bmatrix} & 3 & 3 & 1 & 0 & 0 & 2 & 3 & 3 & 2 & 3 & 1 & 1 & 3 & 2 & 3 & \begin{bmatrix}02&03&03&13\end{bmatrix} \\
\begin{bmatrix}2&1&0&1&0&2&0\end{bmatrix} & 0 & 3 & 2 & 2 & 3 & 1 & 1 & 0 & 3 & 1 & 0 & 3 & 2 & 2 & 0 & \begin{bmatrix}10&03&10&10\end{bmatrix} \\
 & \begin{bmatrix}10\\11\\02\\10\end{bmatrix} & \begin{bmatrix}10\\03\\01\\13\end{bmatrix} & \begin{bmatrix}10\\12\\03\\02\end{bmatrix} & \begin{bmatrix}02\\03\\10\\12\end{bmatrix} & \begin{bmatrix}10\\10\\03\\10\end{bmatrix} & \begin{bmatrix}03\\12\\11\\01\end{bmatrix} & \begin{bmatrix}03\\11\\10\\03\end{bmatrix} & \begin{bmatrix}11\\10\\02\\10\end{bmatrix} & \begin{bmatrix}11\\10\\03\\03\end{bmatrix} & \begin{bmatrix}02\\10\\02\\13\end{bmatrix} & \begin{bmatrix}03\\13\\01\\10\end{bmatrix} & \begin{bmatrix}01\\10\\11\\11\end{bmatrix} & \begin{bmatrix}00\\10\\12\\11\end{bmatrix} & \begin{bmatrix}03\\02\\10\\12\end{bmatrix} & \begin{bmatrix}13\\01\\11\\02\end{bmatrix} &
\end{array}
$$

Figure 5.4: *A completed monoalphabetic GLUV with* $n = 15$ *and* $s = 4$.

The chosen GLUV scheme is monoalphabetic, with $n = 15$ and $s = 4$. Every cell of the grid may be encoded with a symbol from the input alphabet $S = \{0, 1, 2, 3\}$ and the appropriate meta-information calculated before being converted to symbols or pairs of symbols from the 4-ary alphabet in use. Every length vector value is from $S = \{0, 1, \ldots, 15\}$ and can be converted to a pair of 4-ary symbols, as outlined in Section 5.2.2; similarly every consecutive pair of bits in each up-down vector can also be converted to a value from S. An example of a monoalphabetic GLUV with $n = 15$ and $s = 4$ is shown in Figure 5.4.

Following conversion to a monoalphabetic 4-ary GLUV, the structure can be transmitted over a 4-ary erasure channel, where a number of 4-ary symbols might be lost. Post-transmission, a variety of solution strategies can be employed to attempt to recover the missing information from the grid; recovery of meta-data is not imperative as only the grid contains non-redundant (message) information. The simulation outlined in the next chapter gives details on the variety of solution strategies that can be employed to recover information from a partially completed monoalphabetic GLUV.

Chapter 6

GLUV Codes Simulation

In order to test the robustness of the monoalphabetic GLUV structure, introduced in Chapter 5, a simulation was carried out. The simulation imitated transmission of a complete order n monoalphabetic GLUV over the s-ary erasure channel and attempted recovery post-transmission, in order to determine the percentage of recovered grids at various erasure probabilities.

At the heart of the simulation is a solver that, at each iteration of the simulation, takes in a partially completed monoalphabetic GLUV and attempts to recover the erased information. Both the solver construction and running of the simulation were conducted using the computer algebra system Maple[1]. Details of the solution strategies employed by the solver are given in Section 6.1, followed by details on the simulation in Section 6.2.

6.1 Solution Strategies of the Monoalphabetic GLUV Solver

The solver predominantly uses candidate sets to attempt recovery and is split into two main stages: a pre-processing stage and a main solver stage. The purpose of the pre-processing stage is to create the variables and vectors required to keep track of changes occurring during the solving process, and also to create and subsequently minimise candidate sets for each erasure in the grid or length vectors. No assignment of deduced values is permitted during the pre-processing stage. The main solver stage then loops continuously while deductions can still be made, employing a variety of solution strategies to further minimise candidate sets, assign values to erasures that only have a single candidate value and appropriately update candidates sets and any other required variables after assignment of deduced values.

6.1.1 Pre-Processing

The stages of pre-processing for the different components of the GLUV structure are here described.

[1]Information on the computer algebra software Maple can be found at `http://www.maplesoft.com/`.

6.1.1.1 Length Vector Pre-Processing

1. Create a new set of length vectors that convert the partially completed s-ary length vectors of the monoalphabetic GLUV into length vectors with decimal values; this creates a polyalphabetic structure that is easier to work with throughout the solving attempts:

    - If no erasures occur in a $\log_s(n+1)$-tuple of s-ary symbols in a s-ary length vector symbol position, then the values are simply converted from s-ary to decimal.
    - If one or more erasures occur in a $\log_s(n+1)$-tuple of s-ary symbols in a s-ary length vector symbol position, then assign the erasure symbol 'e' in place of a decimal value in the new decimal length vector.

2. Count the number of occurrences of each symbol in each partially completed row and column of the grid; place these values into a new set of length vectors, called *count length vectors.* Also count and store the number of erasures in each row and column.

3. Generate a set of candidate values for every decimal length vector erasure using the original s-ary length vectors; if there are p erasures in a $\log_s(n+1)$-tuple for a single s-ary length vector symbol then there are s^p different candidates for the number of occurrences of the given s-ary symbol within the grid.

    For example, let $n = 15$ and $s = 4$ (so $p = 1$ or 2) then every length vector position is represented by $\log_4(15+1) = 2$ symbols from the 4-ary alphabet. Let a 4-ary length vector from a monoalphabetic GLUV be $\begin{pmatrix} 1e & 10 & 10 & ee \end{pmatrix}$; conversion to a decimal length vector gives $\begin{pmatrix} e & 4 & 4 & e \end{pmatrix}$. The candidate sets for each of the two erased positions contain $4^1 = 4$ and $4^2 = 16$ elements respectively. From the 4-ary length vector, the sets are $\{4, 5, 6, 7\}$ and $\{0, 1, \ldots, 15\}$.

4. Next, minimise the candidate sets for each erased length vector symbol, by eliminating values that are not feasible:

    - Eliminate candidate values that are less than the current number of occurrences of the symbol for the row or column, as they are too small to be feasible.
    - Eliminate candidates that are greater than the current number of occurrences of the symbol plus the number of erasures for the row or column, as this is the maximum number of occurrences of the symbol and cannot be exceeded.

6.1.1.2 Up-Down Vector Pre-Processing

Create a new set of up-down vectors that convert the partially completed s-ary up-down vectors into their original binary form, mapping across strings of $\log_2(s)$ erasures in the correct positions in the partially completed binary vectors.

The recovery of missing up-down symbols, or candidate set creation, is not necessary for up-down vectors as their recovery depends solely upon the grid symbols; if the up-down symbol can be recovered then the grid information is already in place and as we only need recover grid symbols, up-down symbol recovery is unnecessary.

6.1.1.3 Grid Pre-Processing

1. Generate candidate sets for each erased grid symbol:

    - If the current number of occurrences of a symbol (indicated by the count length vector) is less than the number of required occurrences of a symbol (indicated by the partially completed decimal length vector) in both the row length vector and column length vector associated with some grid position, then the symbol is a candidate for the position.
    - This is also the case when erasures occur in the decimal length vector and the required number of occurrences is not known.

2. Minimise candidate sets for each erased grid symbol using the erasure's (non-erased) surrounding values and the associated up-down vector symbols:

    Candidate sets for grid symbols can be minimised using the grid symbols horizontally and vertically adjacent to the erasure in question, alongside the associated up-down vector symbols – provided that neither of the required elements are erased. If both the grid symbol and the up-down vector symbol are present then candidate set minimisation is possible by comparing each value of the candidate set to its adjacent grid symbol and ruling out those that do not satisfy the inequality that is deduced from the associated up-down symbol.

    For example, if an erasure has candidate set $\{1,2,3\}$ and the erasure's preceding element is a '2', then the associated up-down symbol can be utilised to minimise the candidate set $\{1,2,3\}$. If the up-down symbol is a '1', this indicates that the erasure must be strictly greater than 2, hence the candidate set may be minimised to simply $\{3\}$ – this value can subsequently be assigned as the erasure has only a single candidate. If, however, the associated up-down symbol is a '0', this indicates that the erasure must be less than or equal to 2, resulting in a minimised candidate of $\{1,2\}$; further minimisation is hence required to allow inference of an appropriate value to be made.

6.1.1.4 Up-Down Automatic Assumptions

Up-down automatic assumptions have been named thus as they allow a grid erasure's candidate set to be automatically reduced to a candidate set of size one; that is, they allow an inference/assignment to be made. Such assumptions can only be made when an adjacent value and the up-down symbol associated with an erasure take certain values:

- If an erasure is preceded by the symbol $s-2$ and the associated up-down symbol is a '1' then the erasure's candidate set may be minimised to simply $\{s-1\}$.
- If an erasure is followed by the symbol $s-1$ and the associated up-down symbol is a '0' then the erasure's candidate set may be minimised to $\{s-1\}$.
- If an erasure is preceded by a '0' and the associated up-down symbol is a '0' then the erasure's candidate set may be minimised to simply $\{0\}$.
- If an erasure is followed by a '1' and the associated up-down symbol is a '1' then the erasure's candidate set may be minimised to $\{0\}$.

6.1.2 Solver

The solver uses a variety of solution strategies to attempt to recover the erasures in the grid. Each algorithm is attempted in turn, with the algorithm being applied if certain conditions are met by the concerned values, this continuing until none of the solver algorithms are capable of resulting in the assignment of an erased grid symbol. The algorithms used by the solver are detailed below.

6.1.2.1 Assignments and Updates

1. Search through the candidate sets of all length vector erasures. If any candidate set contains only a single value, then assign this value to the erased (decimal) length vector position.
2. If only a single length vector position contains an erasure, the erased value may be recovered without use of candidate sets, as all s values in the length vector must sum to n.
3. If a grid erasure's candidate set is of size one, then assign the single value contained in the candidate set to the erased grid position.
4. Following assignment, appropriate updates must be made to the meta-data and candidate sets:
    - Update the row and column count length vectors associated with the new assignment, alongside the erasure count for the row and column.
    - Check if any row or column length vector candidate sets can be minimised further as both the erasure count and the number of occurrences of one of the length vector symbols has now changed for the row and column.
    - Minimise any grid erasures' candidate sets that may have been affected by the assignment of a grid value.

6.1.2.2 Additional Solution Techniques

1. If there is only a single erasure in a given row or column, it may be possible to identify a single value that should be assigned to the erased grid position without utilising candidate sets. If the current number of occurrences of a symbol is one fewer than the required number of occurrences of a symbol (as indicated by the count length vectors and decimal length vectors respectively), then the value that should be assigned to the grid position can be deduced. As updates are required on every occasion that a grid assignment is made, the candidate set for the erasure is minimised to contain only the deduced value; an assignment will be made at the next iteration through the solver, once all algorithms have been attempted. This algorithm relies on the length vector position relating to the erased symbol not being erased and hence will not always yield an inference of value.

2. If only two values are missing from a given row or column with the same candidate set of size two, and these erasures are immediately contiguous then the up-down vector symbol associated with the contiguous erasures can be used to deduce the order of values to be assigned to the positions. If the associated up-down vector symbol is a '1' then the values must be ascending, if the up-down vector symbol is a '0' the values must be placed in descending order; assignment is not made directly – the deduced values are assigned to the positions contained in minimised candidate sets of size one, and the assignment is made at the next iteration to avoid unnecessary repetition of meta-data and candidate set updates.

3. If the required number of occurrences of a symbol equals the current number of occurrences of a symbol then no further occurrences of the symbol are required for the particular row or column. As a result, the symbol in question may be removed as a candidate from all candidate sets in the row or column.

4. If any of the up-down vectors contain a string of $s-1$ consecutive values equal to 1, then the grid values associated with the up-down symbols must be all the values of the s-ary alphabet in natural order. Assign a minimised candidate set of size one to each cell of the grid in the sequence, containing the value of the s-ary alphabet relating to the cell's position in the sequence. The assignment of actual values to cells, as opposed to the assignment of minimised candidates will occur at the next iteration to avoid the requirement to constantly update meta-data and candidate sets.

5. If the required number of occurrences of a symbol in a row or column is one more than the current number of occurrences of the symbol (indicated by the decimal length vectors and count length vectors respectively), and the symbol in question occurs as a candidate for only a single position in the row or column, then the symbols should be assigned. Minimise the candidate set of this position such that it contains only the symbol satisfying the above conditions; assignment and appropriate updates will be made at the next iteration.

6.2 Simulation Incorporating the Monoalphabetic GLUV Solver

The simulation used was an iterative process that imitated the transmission of a complete order n monoalphabetic GLUV over the s-ary erasure channel, simulating the erasures that would occur within the structure during transmission by randomly erasing symbols of the monoalphabetic GLUV according to a variable probability of erasure. Following the generation of a GLUV containing errors, the solver described in Section 6.1 was employed to endeavour to recover the erased grid symbols, and the success or failure of the recovery attempt was determined.

The simulation is made up of three main stages: creation and deletion; solution attempt; and determining results. These are discussed in more detail in the following sections.

6.2.1 Creation and Deletion

Let n be the grid size of the monoalphabetic GLUV and s be the size of the single alphabet that will be used, such that $n = 2^i - 1$, $s = 2^j$ and $s^k = n+1$ where i, j and k are integers. Additionally, let its be the number of iterations required for the simulation and p be the probability of erasure to be used throughout a run of the simulation (runs of the simulation will be implemented for various values of p).

For each iteration required by its, complete the following steps in turn:

1. Using a random number generator, generate an order n array of symbols from the alphabet $\{0, 1, \ldots, s-1\}$.

2. Create empty row and column vectors in which to record length vector and up-down vector information. These are in decimal and s-ary form for length vectors, and binary and s-ary form for the up-down vectors.

3. Complete the decimal length vectors by counting the number of occurrences of each symbol 0 to $s-1$ in each row and column of the array and convert the decimal values obtained to s-ary and assign to equivalent positions in the s-ary vectors.

4. Complete the binary up-down vectors by observing the relative increases, equalities and decreases between successive symbols of the array.

5. Complete the s-ary up-down vectors by combining successive strings of $\log_2(s)$ bits and converting to s-ary values.

6. Upon completion of the monoalphabetic and polyalphabetic meta-data, a second random number generator can be utilised to randomly erase s-ary symbols from the monoalphabetic GLUV according to a specified erasure probability p – the simulation is run on a variety of erasure probabilities.

7. Copies of the completed grid and meta-data must be created and stored for comparison after a solution attempt is made using the solver, and a partially completed monoalphabetic GLUV is generated by randomly applying erasures to the grid.

6.2.2 Solution Attempt

The partially completed monoalphabetic GLUV generated above is then passed into the solver where candidate sets are generated and a variety of solution strategies are employed to attempt to deduce missing grid values. When no further inferences or assignments can be made the solver terminates.

6.2.3 Determining Results

Following the utilisation of the solver to attempt to recover the partially completed monoalphabetic GLUV for the given iteration, the original grid is compared to the grid output by the solver. The recovery of all meta-data is not critical to the GLUV recovery, and so a grid is considered to be successfully recovered if all information symbols are recovered correctly. If the grids are equal then recovery is deemed successful and a success count variable can be incremented.

Once all iterations are complete, the success rate can be calculated for the particular erasure probability p used, and the simulation may be run again for a different value of p.

6.3 Monoalphabetic GLUV Simulation with $n = 15$ and $s = 4$

6.3.1 Monoalphabetic GLUV Simulation Results

As indicated and concluded in Sections 5.2.2 and 5.3 respectively, the most favourable values for monoalphabetic GLUV are $n = 15$ and $s = 4$, as values satisfying the monoalphabetic constraints $n = 2^i - 1$, $s = 2^j$ and $s^k = n + 1$ where $i, j, k \in \mathbb{Z}$ beyond those values stated are extremely large.

A simulation was carried out on 1000 monoalphabetic GLUVs ($its = 1000$) of order 15 ($n = 15$) with symbols from an alphabet $S = \{0, 1, 2, 3\}$, that is $s = 4$. The simulation was carried out at various erasure probabilities p, and the results of the simulation are given in both tabular (Table 6.1) and graphical form (Figure 6.1).

Sample of 1000 Randomly Generated Monoalphabetic GLUVs ($n = 15, s = 4$)	
Probability of Erasure (p)	Percentage Recovered (%)
0.05	100%
0.1	99.9%
0.15	99%
0.2	95.4%
0.25	82.3%
0.3	54.3%
0.35	14.7%
0.4	1.2%
0.45	0%
0.5	0%
⋮	⋮

Table 6.1: *Tabular results of simulation on 1000 randomly generated monoalphabetic GLUVs of order $n = 15$ over an alphabet $S = \{0, 1, 2, 3\}$ at various erasure probabilities.*

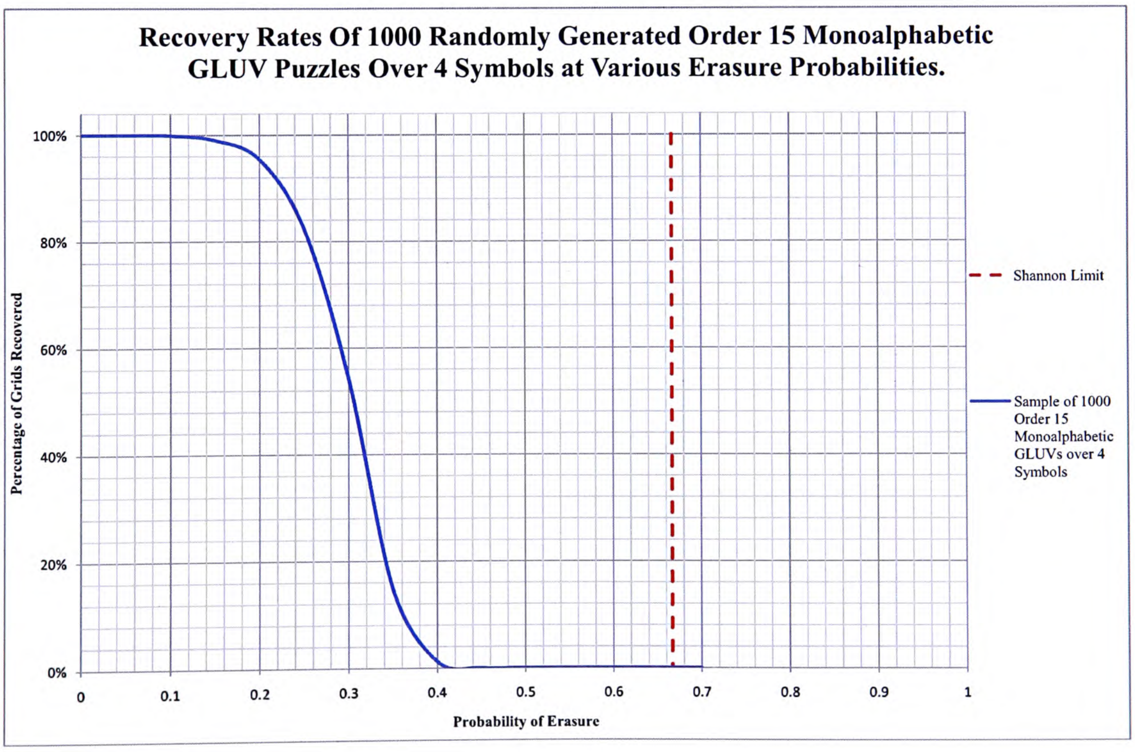


Figure 6.1: *Graphical results of simulation on 1000 randomly generated monoalphabetic GLUVs of order $n = 15$ over an alphabet $S = \{0, 1, 2, 3\}$ at various erasure probabilities.*

6.3.2 Alternative GLUV Structure Simulation Results

In addition to the monoalphabetic GLUV simulation detailed in Section 6.3.1, further simulations have been conducted on a similar structures. These structures consist of only two components, the grid itself along with a single set of meta-data; either length vectors or up-down vectors, deemed GLV and GUV respectively for brevity. The existing solver was modified in each case to consider the two required components for each simulation, in order to ascertain whether the the improved rate offered by only a single set of meta-data would yield more encouraging results when compared to the standard three component monoalphabetic GLUV. The outcomes of each of the simulations are detailed in the subsequent sections.

6.3.2.1 GLV

The GLV structure consists of length vectors and the grid itself. The rate of the two component GLV structure is given by

$$R = \frac{n^2 \log_2 s}{n^2 \lceil \log_2 s \rceil + 2ns \lceil \log_2 (n+1) \rceil}$$

and evaluation when $n = 15$ and $s = 4$ (as with monoalphabetic GLUV) results in a rate of 0.4839. This rate is much improved upon monoalphabetic, three component GLUV, and the GLV structure hence needs only perform at probabilities approaching $1 - 0.4839 = 0.5161$ to be considered near optimal, as opposed to the erasure probability 0.6667 required by monoalphabetic, three component GLUV. However, the simulation results given in Table 6.2 and Figure 6.2 indicate that the combination of two sets of meta-data in monoalphabetic GLUV yield better recovery rates than a single set of meta-data.

Sample of 1000 Randomly Generated Monoalphabetic GLVs ($n = 15, s = 4$)	
Probability of Erasure (p)	Percentage Recovered (%)
0.05	99.8%
0.1	94%
0.15	71.1%
0.2	31.1%
0.25	4.4%
0.3	0.2%
0.35	0%
0.4	0%
⋮	⋮

Table 6.2: *Tabular results of simulation on 1000 randomly generated alternative monoalphabetic GLUV structures – GLVs, of order $n = 15$ over an alphabet $S = \{0, 1, 2, 3\}$ at various erasure probabilities.*

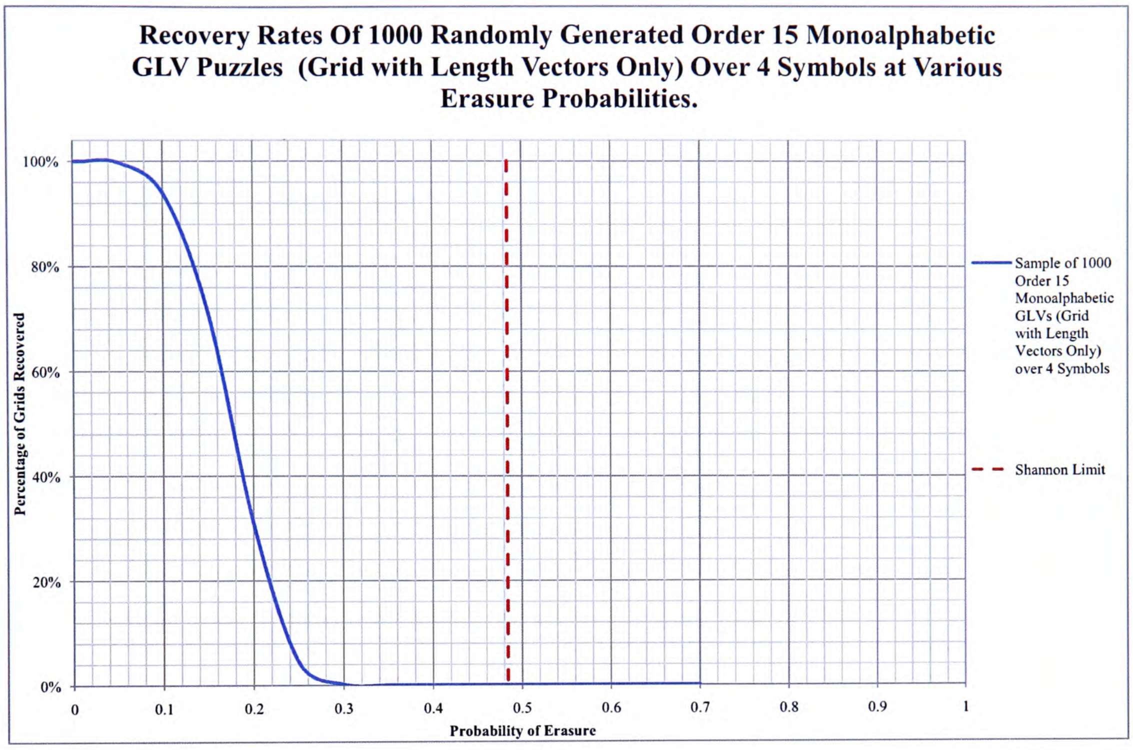


Figure 6.2: *Graphical results of simulation on 1000 randomly generated GLVs, of order $n = 15$ over an alphabet $S = \{0, 1, 2, 3\}$ at various erasure probabilities.*

6.3.2.2 GUV

The GUV structure consists of up-down vectors and the grid itself. The results of the simulation indicate that up-down vectors alone provide poor erasure correction, failing to recover any grids in 1000 iterations at even the smallest erasure probability $p = 0.05$.

6.3.3 Remarks on GLUV Simulation Results

The simulation results detailed in Table 6.1 and Figure 6.1 indicate that monoalphabetic GLUV does not offer the improvements that were expected over the Sudoku scheme. An investigation into some of the partially completed puzzles that were not solved at lower erasure probabilities during the simulation indicated the presence of sub-structures similar to Latin sub-squares within Sudoku.

While some of these structures are 'Latin' in type, many are not. However they all share the property that the removal of all sub-structure symbols results in a grid with multiple possible solutions. At an erasure probability of 0.15 detailed in Table 6.1, the attempted recovery of the 1000 analysed grids yielded ten 'unrecoverable' grids. Section 6.3.4 details and investigates each of the ten failed grids in turn and highlights the presence of any sub-structures that are hindering recovery.

6.3.4 Analysis of Failed Grids

Each of the subsequent ten partially-completed monoalphabetic GLUVs is not fully recovered by the solver used in the simulation. Each puzzle is given in the partially-recovered state from which it was output by the solver, where all deduced grid symbols and length vector symbols have been assigned. Erased up-down values have not been deduced or assigned as their deductions do not aid the recovery of deleted grid symbols. Although the solver and simulation convert to decimal or binary vectors when attempting to recover grids, all components of the monoalphabetic GLUVs are given in 4-ary within each of the ten puzzles below; utilising 4-ary allows the number of erasures within sub-structures to be more easily identified (as will be shown in Chapter 7). All unrecoverable sub-structures shall henceforth be referred to as *halting sets.*

6.3.4.1 Unrecovered Grid I

	2	1	2	e	2	2	e	e	0	3	1	1	0	0	1	
	e	1	1	1	1	3	0	2	e	1	1	e	e	2	0	
	2	2	0	0	0	0	e	2	1	0	0	e	1	1	3	
	2	3	2	1	3	e	2	0	0	e	e	2	0	e	0	
	2	1	2	0	0	0	e	e	e	0	2	0	1	1	0	
	2	0	2	e	3	3	0	1	0	2	0	2	1	e	1	
	2	2	0	2	2	e	2	1	1	0	1	1	e	e	0	
[0 2 e 0 e 2 0]	0	0	0	3	0	0	3	3	2	1	1	3	3	2	1	[11 03 02 11]
[1 1 0 1 0 3 e]	e	0	2	1	3	3	0	0	2	2	0	1	2	1	1	[10 1e 10 0e]
[0 2 e 0 2 e e]	1	1	0	2	2	0	3	1	0	3	3	2	1	0	2	[10 10 10 03]
[0 e 1 0 1 1 e]	3	1	0	2	2	1	2	2	2	0	1	1	2	2	2	[02 10 20 01]
[2 e 0 2 e 2 0]	1	2	1	3	3	3	0	2	2	3	2	3	3	1	0	[02 03 10 12]
[0 0 2 2 1 2 1]	3	3	0	0	0	3	2	3	1	1	2	3	0	0	2	[11 02 03 11]
[0 0 2 3 1 1 1]	1	1	0	0	0	2	0	2	3	0	2	0	3	2	3	[12 02 10 03]
[0 1 2 e e 2 2]	2	2	2	0	1	3	2	2	3	3	2	3	2	3	2	[01 01 20 11]
[2 1 0 e 0 e 2]	2	3	2	1	3	3	1	2	3	2	0	3	1	3	2	[01 03 11 12]
[e e 0 2 1 0 2]	3	1	3	1	3	1	1	3	3	1	3	3	1	2	1	[00 13 01 13]
[e 0 1 1 1 1 2]	1	3	2	0	0	0	3	1	3	0	3	0	2	3	1	[11 03 02 11]
[1 0 1 e 2 2 2]	3	0	3	1	1	1	2	0	1	3	1	3	0	3	1	[03 12 01 11]
[1 0 0 2 e 2 2]	0	0	2	2	2	2	0	3	1	3	1	3	1	3	3	[03 03 10 11]
[2 2 0 0 0 1 1]	2	3	2	3	3	1	1	0	0	0	0	0	3	2	3	[11 02 03 11]
[3 1 0 2 1 e 3]	0	1	2	0	2	0	0	2	2	0	1	2	0	2	3	[12 02 12 01]
	03	10	11	11	10	10	11	03	02	11	03	03	03	02	01	
	ee	11	01	10	02	10	03	02	03	03	11	02	10	02	11	
	03	02	13	03	10	02	10	12	11	02	10	02	10	12	11	
	1e	10	02	03	11	11	03	14	11	11	03	20	10	11	10	

Figure 6.3: *A* 15 × 15 *'unrecoverable' monoalphabetic GLUV over 4 symbols, containing a single halting set (highlighted in yellow).*

The first unrecoverable grid is shown in Figure 6.3, with the erasures forming the halting set highlighted in yellow. When combined with the associated row and column length vectors, the single grid erasure has a candidate set of $\{1, 3\}$, but the up-down vector information does not allow a deduction of which candidate should be placed in the grid.

6.3.4.2 Unrecovered Grid II

Figure 6.4 shows the second grid that the solver was unable to fully recover. This figure shows only the necessary row and column of the grid, along with the appropriate meta-data, with the erasures forming the halting set again highlighted in yellow. When combined with the associated row and column length vectors, the single grid erasure has a candidate set of $\{1, 2\}$, the deduction of which candidate should be assigned to the erased position is not possible when combining up-down vector information and values of the erasure-adjacent cells.

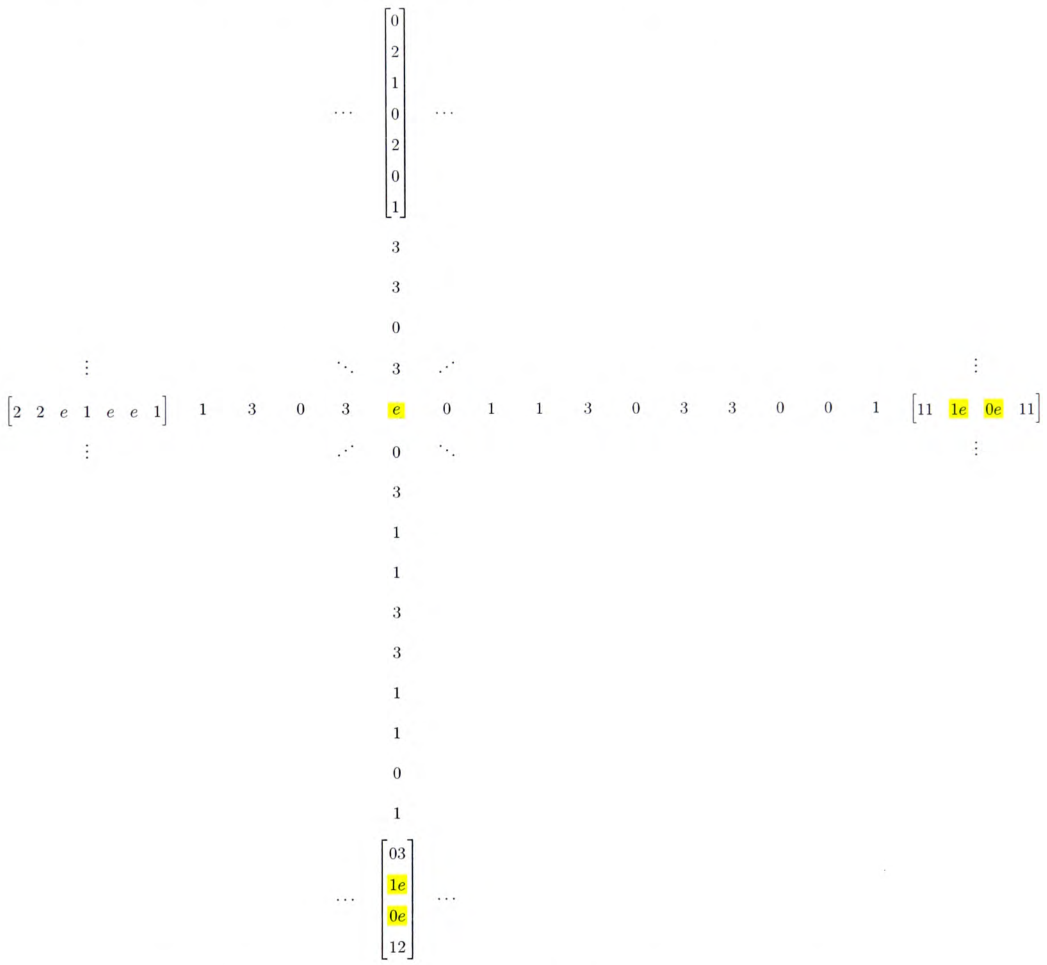
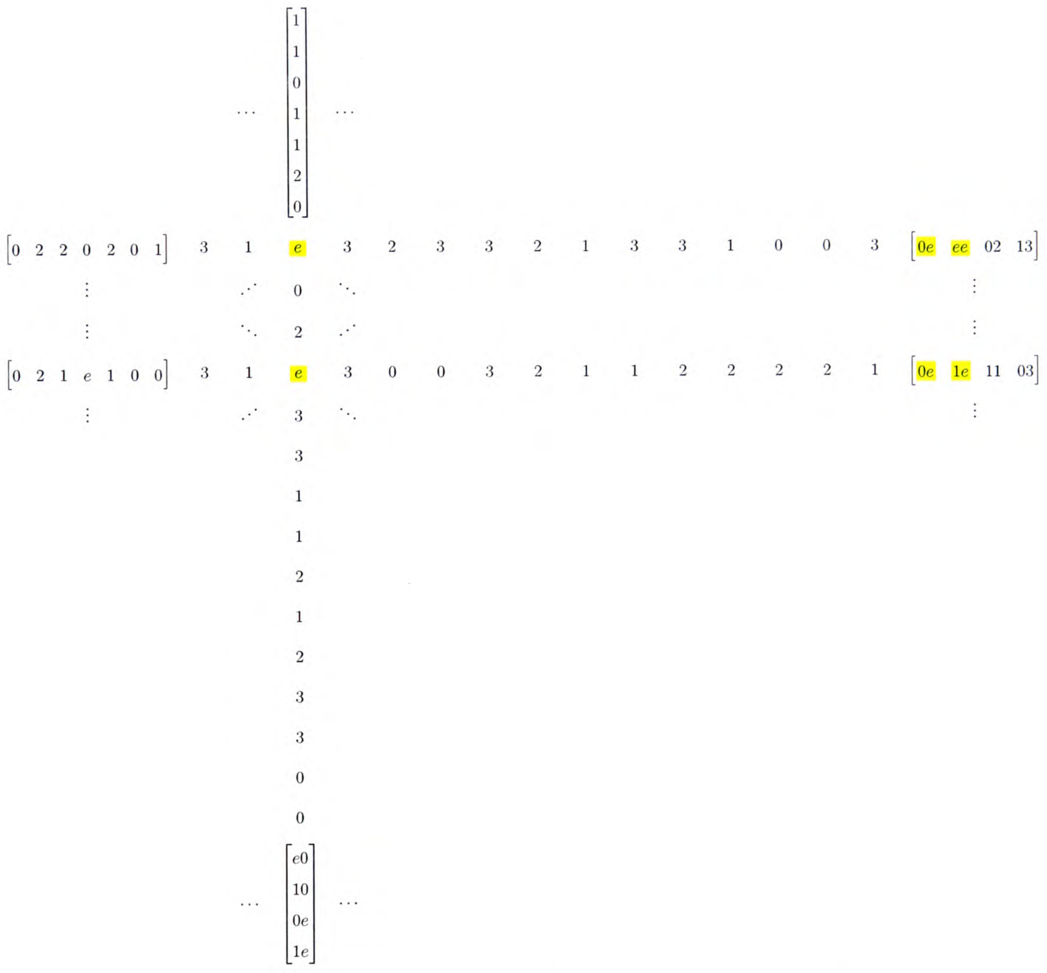


Figure 6.4: *A single halting set (highlighted in yellow) occurring within* 15×15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.3 Unrecovered Grid III

Unrecovered grid three is given in Figure 6.5, with the erasures generating the halting set highlighted in yellow, and only appropriate grid symbols and meta-data given. This halting set consists of two grid erasures; examination of the row and column length vectors shows that each grid erasure has candidate set $\{0, 1\}$. The values surrounding each of the grid erasures are such that, when combined with up-down vector information, the deduction of either of the erased values is not possible.



[1 1 0 1 1 2 0]
[0 2 2 0 2 0 1] 3 1 e 3 2 3 3 2 1 3 3 1 0 0 3 [0e ee 02 13]
0
2
[0 2 1 e 1 0 0] 3 1 e 3 0 0 3 2 1 1 2 2 2 2 1 [0e 1e 11 03]
3 3 1 1 2 1 2 3 3 0 0
[e0 10 0e 1e]

Figure 6.5: *A halting set consisting of two grid erasures and additional length vector erasures (highlighted in yellow) occurring within a* 15×15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.4 Unrecovered Grid IV

Figure 6.6 details another 'unrecoverable' grid output by the solver. The halting set in this example consists of four grid erasures and an additional up-down vector erasure; each grid erasure having candidate set $\{2, 3\}$, identified by comparing the row and column totals to the appropriate length vectors. The halting set is 'Latin' in type as every row and column must be assigned one of each of the candidate values, this type of halting set is hence equivalent to the intercalates observed in the Sudoku structure in Chapter 4. The additional up-down vector erasure removes the possibility of deducing the correct candidate when combined with the erasure-adjacent cell value.

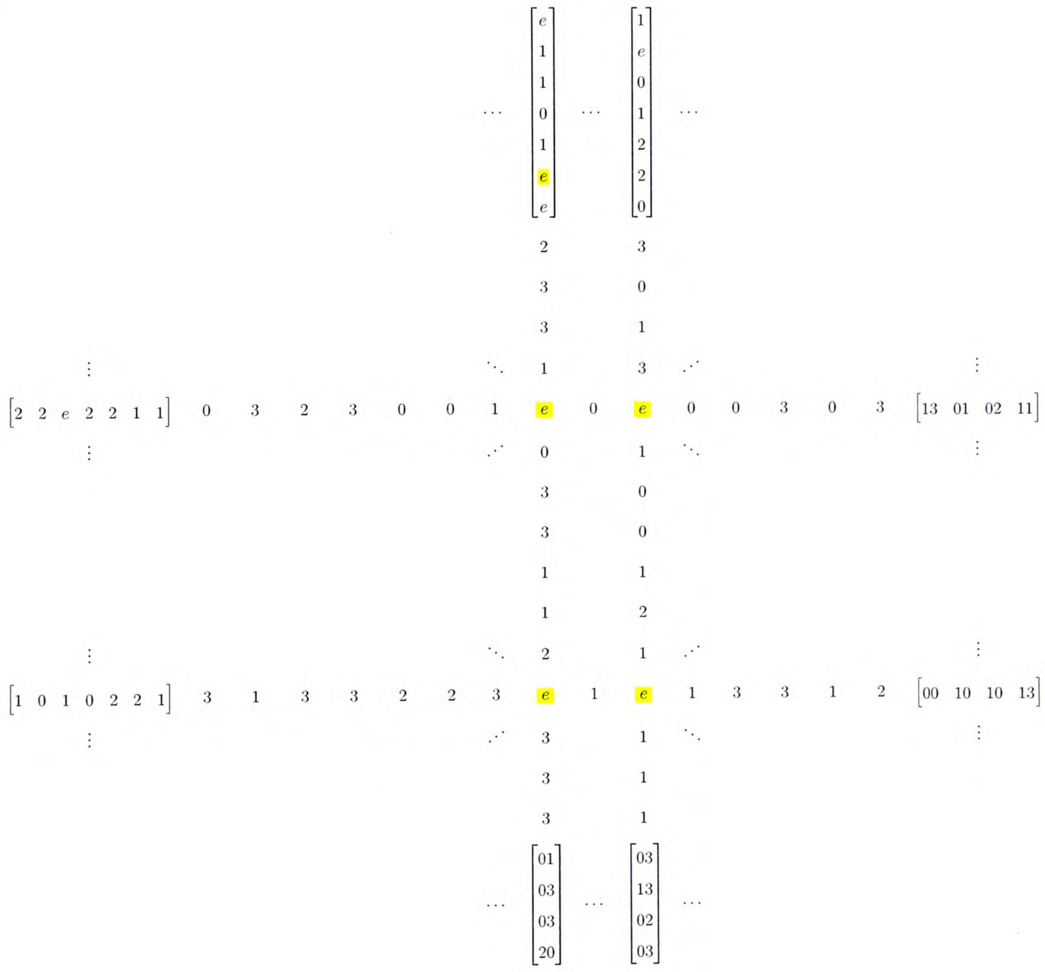


Figure 6.6: *A halting set consisting of four grid erasures and an additional up-down vector erasure (highlighted in yellow) occurring within a* 15×15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.5 Unrecovered Grid V

Unrecovered grid five is given in Figure 6.7, where the single grid erasure and additional length vector erasures are highlighted in yellow and only necessary grid symbols and meta-data are included. The single grid erasure has candidate set $\{1, 2\}$ inferred from the partially erased length vectors. The values surrounding the single grid erasure are such that the associated up-down vector symbols cannot be exploited to deduce the correct candidate value to assign and the erasure pattern (and it's surrounding values) is thus a halting set.

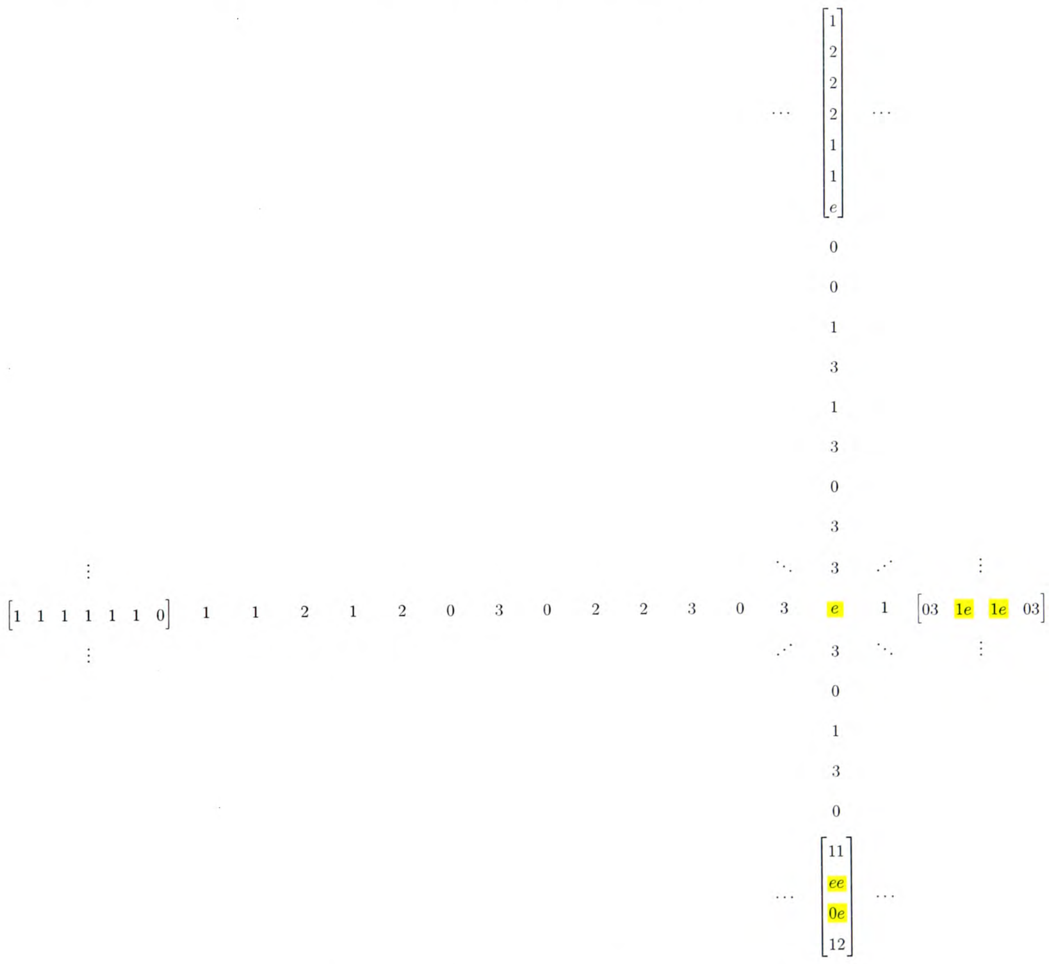


Figure 6.7: *A single halting set (highlighted in yellow) occurring within* 15×15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.6 Unrecovered Grid VI

Figure 6.8 details another 'unrecoverable' grid output by the solver. The halting set in this example consists of a single grid erasure accompanied by a length vector erasures; generating a grid erasure candidate set of $\{2, 3\}$, identified by comparing the row and column totals to the appropriate length vectors; although there are additional erasures in each of the row and column length vectors, erasures only occur in both in positions 2 and 3. The values horizontally and vertically adjacent to the grid erasure are such that their combination with the appropriate up-down information does not allow for the deduction of the correct value. As a result, a halting set is formed.

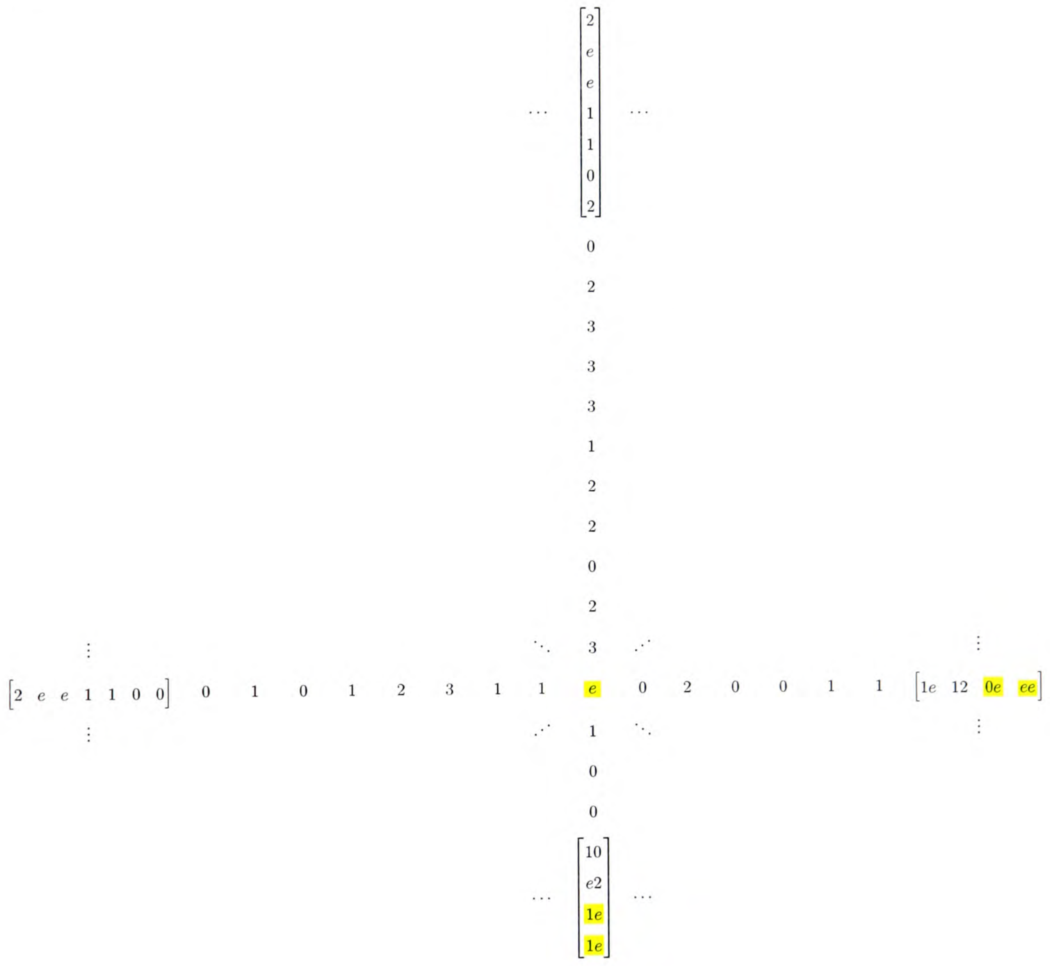


Figure 6.8: *A single halting set consisting of one grid erasure alongside length vector erasures (highlighted in yellow), occurring within* 15 × 15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.7 Unrecovered Grid VII

The halting set in Figure 6.9 consists solely of four grid erasures; each grid erasure having candidate set $\{1, 2\}$, identified by comparing the row and column totals to the appropriate length vectors. The halting set is 'Latin' in type, as in Figure 6.6, as every row and column must be assigned one of each of the candidate values. All erasure-adjacent values are such that they do not allow for a deduction of an erased grid value when combined with the information in the up-down vectors; as no values can be deduced, a halting set exists within the GLUV structure.

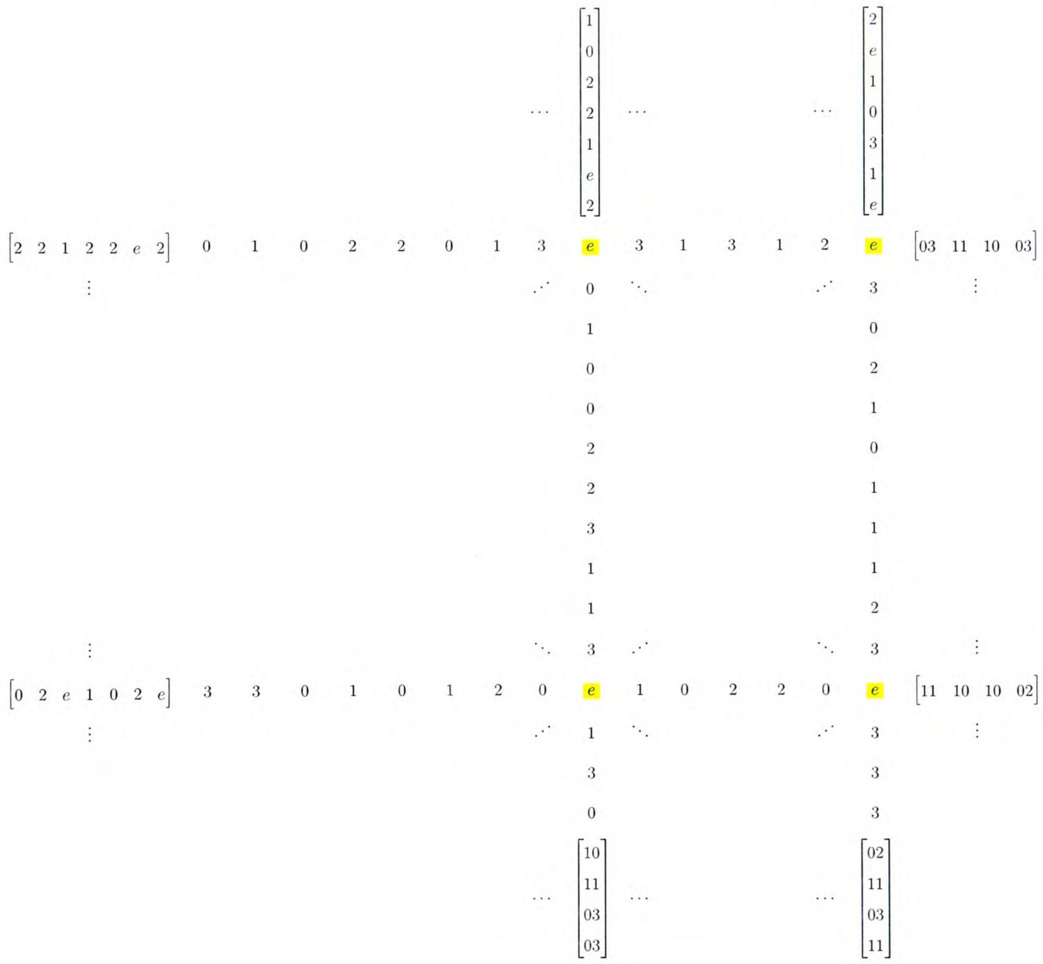

[1 0 2 2 1 e 2]
[2 e 1 0 3 1 e]
[2 2 1 2 2 e 2] 0 1 0 2 2 0 1 3 e 3 1 3 1 2 e [03 11 10 03]
0 1 0 0 2 2 3 1 1 3
3 0 2 1 0 1 1 1 2 3
[0 2 e 1 0 2 e] 3 3 0 1 0 1 2 0 e 1 0 2 2 0 e [11 10 10 02]
1 3 0
3 3 3
[10 11 03 03]
[02 11 03 11]

Figure 6.9: *A halting set consisting of four grid erasures (highlighted in yellow) occurring within a* 15×15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.8 Unrecovered Grid VIII

Figure 6.10 contains another halting set that is 'Latin' in type. The set consists of four grid erasures only, each row and column requires an assignment of one of each of the elements of the candidate set $\{2, 3\}$. All erasure-adjacent cells contain values that do not lead to a deduction of an erased value when combined with the up-down vectors.

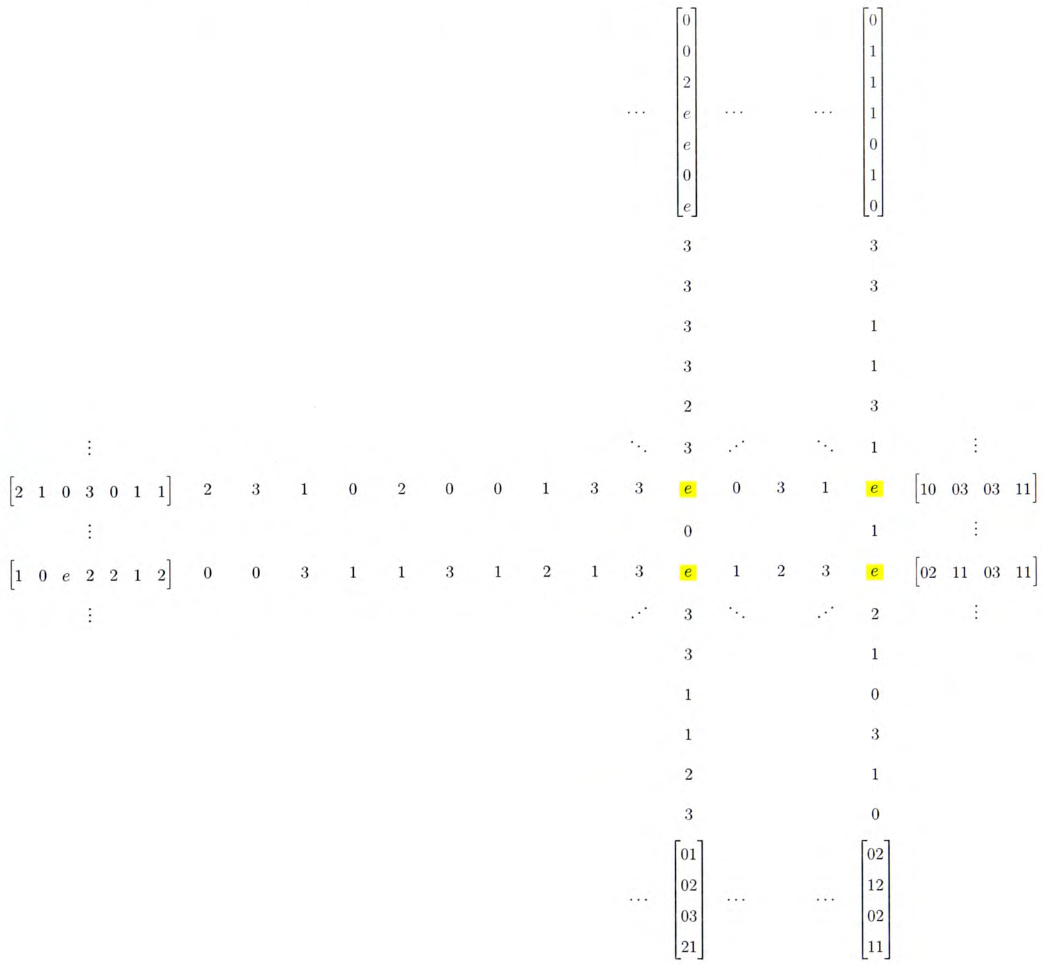
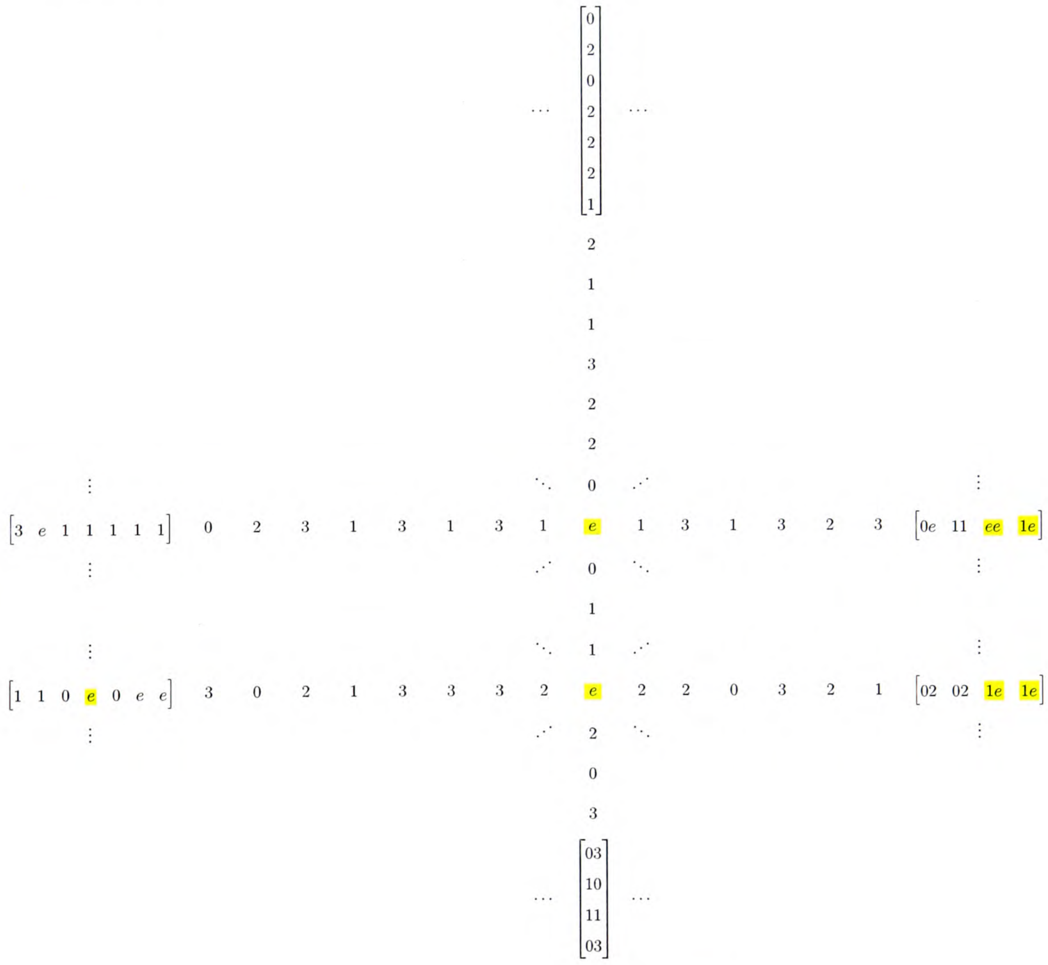

[0 0 2 e e 0 e]
[0 1 1 1 0 1 0]
3 3 3 3 2 3
3 3 1 1 3 1
[2 1 0 3 0 1 1] 2 3 1 0 2 0 0 1 3 3 e 0 3 1 e [10 03 03 11]
0 1
[1 0 e 2 2 1 2] 0 0 3 1 1 3 1 2 1 3 e 1 2 3 e [02 11 03 11]
3 3 1 1 2 3
2 1 0 3 1 0
[01 02 03 21]
[02 12 02 11]

Figure 6.10: *A* 15 × 15 *'unrecoverable' monoalphabetic GLUV over 4 symbols, containing a single halting set consisting of four grid erasures (highlighted in yellow).*

6.3.4.9 Unrecovered Grid IX

Figure 6.11 details another 'unrecovered' grid output by the solver. The halting set within the figure consists of two grid erasures, a single up-down vector erasure and additional length vector erasures. The candidate set for both grid erasures, generated by the length vectors is $\{2, 3\}$ and the position of the up-down vector erasure is such that the deletion of its value does not allow for the deduction of the related grid symbol (if the up-down symbol was not erased the grid value could be deduced).

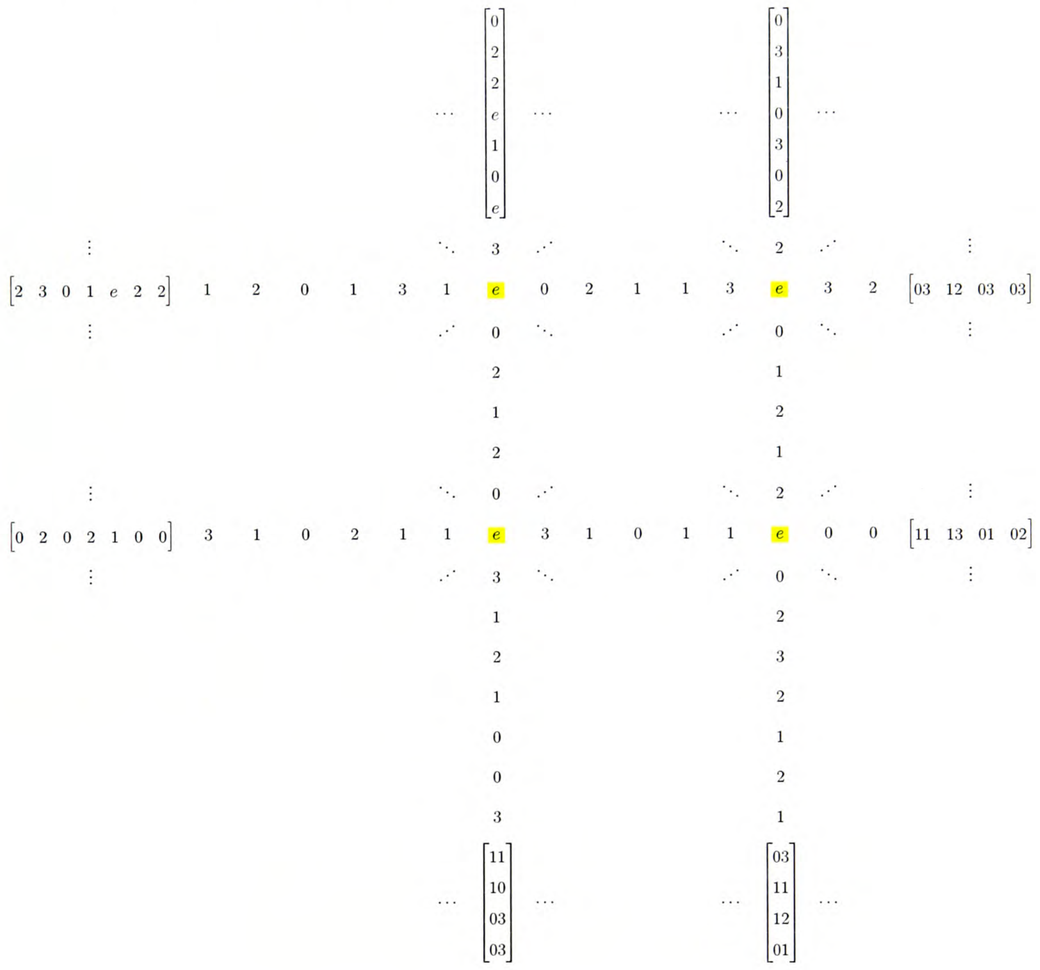


[0 2 0 2 2 2 1]
2 1 1 3 2 2 0
[3 e 1 1 1 1 1] 0 2 3 1 3 1 3 1 e 1 3 1 3 2 3 [0e 11 ee 1e]
0 1 1
[1 1 0 e 0 e e] 3 0 2 1 3 3 3 2 e 2 2 0 3 2 1 [02 02 1e 1e]
2 0 3
[03 10 11 03]

Figure 6.11: *A halting set consisting of two grid erasures, a single up-down vector erasure and additional length vector erasures (highlighted in yellow) occurring within a* 15 × 15 *monoalphabetic GLUV, where unnecessary grid symbols and meta-data are omitted.*

6.3.4.10 Unrecovered Grid X

The final 'unrecoverable' grid is given below in Figure 6.12 and details another 'Latin' type halting set. The halting set consists of four grid erasures, each with candidate set $\{0, 1\}$. The erasure-adjacent values cannot be combined with up-down vector information to allow the deduction of an erased value, and the grid is hence unrecoverable.



[0 2 2 e 1 0 e]
[0 3 1 0 3 0 2]
3
2
[2 3 0 1 e 2 2] 1 2 0 1 3 1 e 0 2 1 1 3 e 3 2 [03 12 03 03]
0 2 1 2 0
0 1 2 1 2
[0 2 0 2 1 0 0] 3 1 0 2 1 1 e 3 1 0 1 1 e 0 0 [11 13 01 02]
3 1 2 1 0 0 3
0 2 3 2 1 2 1
[11 10 03 03]
[03 11 12 01]

Figure 6.12: *A* 15 × 15 *'unrecoverable' monoalphabetic GLUV over 4 symbols, containing a single halting set that consists of four erasures and is 'Latin' in type (highlighted in yellow).*

6.4 Concluding Remarks

Although further algorithms could be added to the GLUV solver to slightly improve its capacity to recover erased values, it has already been demonstrated the prevalence of sub-structures that prevent recovery altogether. As symbols from the input alphabet may be placed in any positions within the information grid, the occurrence of sub-structures seems more frequent than in Sudoku. While the up-down vectors go some way to allowing Latin sub-structures to be recovered, the relationship between length vector symbols and grid symbols does not prevent the emergence of different types of sub-structure. These sub-structures and the 'Latin' type sub-structures are collectively referred to here as halting sets. Halting sets are discussed in more detail in Chapter 7, where the smallest and most frequently occurring sets are described more thoroughly.

Chapter 7

Small Halting Sets Classification

The aim of this chapter is to categorise small halting sets of up to five erasures into classes. The classification of halting sets will enable the prevalence of halting sets within order 15 monoalphabetic GLUVs over four symbols to be analysed. The chapter includes in-depth descriptions of each type of halting set, and also includes lower bounds for the probability of a GLUV containing at least one of particular small halting sets.

7.1 1-Erasure, 2-Erasure & 3-Erasure Sets

Halting sets consisting of fewer than four erasures cannot exist. In every example of a 1-erasure, 2-erasure or 3-erasure set, some remaining meta-data or grid information will allow for the recovery of the erased grid values (if any grid values are erased). In order to create a halting set, there needs to be at least one grid symbol erased; if only meta-data are erased then no recovery of information is required. If there is only a single grid erasure, meta-data erasures are required to create doubt over what the erased symbol was originally – thus a candidate set of size at least two is required for this. Intuitively, it can be seen that length vectors erasures are required to accompany a single grid erasure in order to form a halting set, as the erased grid value is easily recoverable with no length vector erasures. There must be four length vector erasures in total, two in the row length vector and two in the column length vector relating to the grid erasure to create the required candidate set of size two. The halting set described is a 5-erasure halting set and is described thoroughly in Section 7.3; this is the smallest possible halting set consisting of a mix of grid and meta-data erasures. The only halting set pattern consisting of fewer erasures is a four erasure halting set, equivalent to intercalates within Sudoku and Latin squares; this consists only of grid erasures – it is the smallest possible halting set, independent sets of data of less than four symbols do not exist for GLUV.

7.2 4-Erasure (Minimal) Halting Sets

A 4-erasure halting set consists of four grid erasures occurring in a 2×2 sub-square pattern, in non-contiguous cells of the grid. If the cells were contiguous then the up-down vectors would allow for recovery of erased data. Each of the two row length vectors and each of the two column length vectors associated with the grid erasures must, when compared with the row/column symbol counts, result in a candidate set of size two, with each erasure in the sub-square pattern having the same two values as candidates.

No additional meta-data erasures are required to produce a halting set, provided that every erasure-adjacent cell adheres to the constraints described below, where a cell is considered to be erasure-adjacent if it is either a horizontal or vertical neighbour of a grid erasure.

Consider the general case of a 4-erasure halting set, where there is no intersection of erasure-adjacent cells and all four erasures do not lie on any edge of the grid. A cell is said to be non-edge-contiguous if it is not on the top, bottom, left or right edges of the grid. This general case requires erasures to be separated by at least two rows or columns, also referred to as two-cell-separable throughout. For this general case, let $C = \{c_1, c_2\}$, where $c_1 \neq c_2$ and $c_1, c_2 \in \{0, 1, 2, 3\}$, be the candidate set for each of the four erased values (candidate sets are equal for all of the grid erasures). If the erasure occurs within cell (i, j) of the grid, then:

1. The erasure-adjacent cells $(i - 1, j)$ and $(i, j - 1)$ may take any value that is greater than or equal to $max(c_1, c_2)$ or strictly less than $min(c_1, c_2)$.
2. The erasure-adjacent cells $(i + 1, j)$ and $(i, j + 1)$ may take any value that is strictly greater than $max(c_1, c_2)$ or less than or equal to $min(c_1, c_2)$.

In cases where the erasure-adjacent cells intersect (are adjacent to more than one erasure), that is where erasures are not two-cell-separable, a further constraint needs to be considered.

3. If a pair of erasures that share the same row or column occur in cells (i_1, j_1) and (i_2, j_2) of the grid, and $i_1 = i_2 \pm 2$ or $j_1 = j_2 \pm 2$ then the cell that is horizontally or vertically adjacent to both erasures may only contain a value that is strictly greater than $max(c_1, c_2)$ or strictly less than $min(c_1, c_2)$.

In cases where an erasure (i, j) is edge-contiguous, at least one of the erasure-adjacent cells $(i - 1, j)$, $(i, j - 1)$, $(i + 1, j)$ and $(i, j + 1)$ does not exist, and any constraints relating to this erasure-adjacent cell may be ignored.

Erasure-adjacent cells satisfying the above constraints result in the up-down vector being insufficient to differentiate between potential placement of the candidate set values c_1 and c_2. The number of different ways of creating a set of erasure-adjacent values that satisfy these constraints depends on the relationship between the candidate set values c_1 and c_2.

Without loss of generality, it may be assumed that $c_1 > c_2$. If there exists a value a such that $a \in \{0, 1, 2, 3\}$ and $c_2 < a < c_1$ then no erasure-adjacent cell (distinct or intersecting) may contain the value a as this would allow the up-down vectors to be utilised to differentiate between the correct placements of c_1 and c_2. As a result, the number of different ways of creating a set of erasure-adjacent values is maximised when the two values in the candidate set for the four grid erasures are consecutive values from the set $\{0, 1, 2, 3\}$.

Table 7.1 summarises the values that specific erasure-adjacent positions may take for specific candidates, if the erasure is considered to have occurred in position (i, j) of the grid. The figure details initial results for the general case only, where all erasures are two-cell-separable and non-edge-contiguous. Specific cases with erasure-adjacent cell intersections and edge-contiguous erasures are detailed in Table 7.2.

Candidate Set	No Erasure-Adjacent Position May Contain	Any Erasure-Adjacent Position May Contain	Positions $(i-1,j)$ and $(i,j-1)$ May Contain	Positions $(i+1,j)$ and $(i,j+1)$ May Contain	Total Number of Ways of Surrounding a Single Erasure	Total Number of Ways of Surrounding the 4-Erasure Halting Set
$\{0,1\}$	–	2 or 3	1,2 or 3	0,2 or 3	$3^4 = 81$	$3^{16} = 43,046,721$
$\{1,2\}$	–	0 or 3	0,2 or 3	0,1 or 3	$3^4 = 81$	$3^{16} = 43,046,721$
$\{2,3\}$	–	0 or 1	0,1 or 3	0,1 or 2	$3^4 = 81$	$3^{16} = 43,046,721$
$\{0,2\}$	1	3	2 or 3	0 or 3	$2^4 = 16$	$2^{16} = 65,536$
$\{1,3\}$	2	0	0 or 3	0 or 1	$2^4 = 16$	$2^{16} = 65,536$
$\{0,3\}$	1 or 2	–	3	0	$1^4 = 1$	$1^{16} = 1$

Table 7.1: *Potential erasure-adjacent cell value sets, for each of the six candidate sets possible with a size four alphabet, accompanied by total number of ways of surrounding the candidate sets in order to create a 4-erasure halting set for the general two-cell-separable, non-edge-contiguous erasure pattern.*

The calculation of the total number of ways of creating a 4-erasure halting set for a specific candidate set for the general two-cell-separable, non-edge-contiguous erasure pattern can be generalised. Given a candidate set $C = \{c_1, c_2\}$ where $c_1, c_2 \in \{0, 1, 2, 3\}$ it may be assumed, without loss of generality, that $c_1 > c_2$. The total number of ways of creating a 4-erasure halting set for the candidate set $\{c_1, c_2\}$ when all erasures in the halting set are two-cell-separable and non-edge-contiguous is:

$$(4 - c_1 + c_2)^{16} \tag{7.1}$$

The value 4 in Equation (7.1) communicates the alphabet size and 16 details the number of horizontally or vertically erasure-adjacent cells.

The number of ways of creating a 4-erasure halting set differs from the values detailed in the generalisation above when erasures are not two-cell-separable; that is, when there is considered to be an intersection between erasure-adjacent cells, or when one cell is neighbouring two erasures. Table 7.2 considers all possible intersections of erasure-adjacent cells (row intersection, column intersection or row and column intersections) for each of the six possible candidate sets, assuming all four erasures are non-edge-contiguous.

Candidate Set	Number of Erasure-Adjacent Position Intersections	Erasure-Adjacent Intersecting Positions May Contain	Non-Intersecting Positions $(i-1,j)$ and $(i,j-1)$ May Contain	Non-Intersecting Positions $(i+1,j)$ and $(i,j+1)$ May Contain	Number of Ways of Surrounding the 4-Erasure Halting Set
$\{0,1\}$	2	2 or 3	1,2 or 3	0,2 or 3	$3^{12} \times 2^2 = 2,125,764$
	4				$3^8 \times 2^4 = 104,976$
$\{1,2\}$	2	0 or 3	0,2 or 3	0,1 or 3	$3^{12} \times 2^2 = 2,125,764$
	4				$3^8 \times 2^4 = 104,976$
$\{2,3\}$	2	0 or 1	0,1 or 3	0,1 or 2	$3^{12} \times 2^2 = 2,125,764$
	4				$3^8 \times 2^4 = 104,976$
$\{0,2\}$	2	3	2 or 3	0 or 3	$2^{12} \times 1^2 = 4,096$
	4				$2^8 \times 1^4 = 256$
$\{1,3\}$	2	0	0 or 3	0 or 1	$2^{12} \times 1^2 = 4,096$
	4				$2^8 \times 1^4 = 256$
$\{0,3\}$	2	–	3	0	$1^{12} \times 0^2 = 0$
	4				$1^8 \times 0^4 = 0$

Table 7.2: *Potential erasure-adjacent cell value sets, for each of the six candidate sets possible with a size four alphabet, accompanied by total number of ways of surrounding the candidate sets in order to create a 4-erasure halting set for the non-edge-contiguous erasure pattern with erasure-adjacent cell intersection.*

The results of Table 7.2 can be generalised as the results from Table 7.1 could be. Given a candidate set $C = \{c_1, c_2\}$, where $c_1, c_2 \in \{0,1,2,3\}$ it may be assumed, without loss of generality, that $c_1 > c_2$. The total number of ways of creating a 4-erasure halting set for the candidate set $\{c_1, c_2\}$, when the number of erasure-adjacent intersecting cells is d and all erasures in the halting set are non-edge-contiguous, is:

$$(4 - c_1 + c_2)^{16-2d} \times (4 - c_1 + c_2 - 1)^d \tag{7.2}$$

$$= (4 - c_1 + c_2)^{16-2d} \times (3 - c_1 + c_2)^d \tag{7.3}$$

The value 4 in Equations (7.2) and (7.3) communicates the alphabet size and 16 details the number of erasure-adjacent cells expected in a 4-erasure halting set, before taking intersecting erasure-adjacent cells into account. The subtraction by 1 in the second bracket of Equation (7.2) accounts for the reduction in the number of possible values that can be placed in intersecting erasure-adjacent cells, as Table 7.2 also outlines.

To complete the general two-cell-separable, non-edge-contiguous case of 4-erasure halting sets, consideration must be given to some sub-cases for which the erasures themselves are edge-contiguous; either one, two or four erasure-adjacent cells will be absent. There are 5 different cases to consider, classified by the number of remaining erasure-adjacent cells for each of the erasures. For example, a case classified by the vector $\begin{pmatrix} x & y & z & w \end{pmatrix}$ where $x, y, z, w \in \{2, 3, 4\}$ has x remaining erasure-adjacent cells for the top-left erasure in the 2×2 sub-square, y remaining erasure-adjacent cells for the top-right erasure of the 2×2 sub-square, z and w remaining erasure-adjacent cells for the bottom-left and bottom right erasures respectively. (It should be noted that if $x = y = z = w = 4$, then we are considering a sixth, now trivial case of four non-edge-contiguous erasures, discussed previously).

In the case where some or all of the four erasures in the halting set are edge-contiguous but remain two-cell-separable, the number of different ways of creating a 4-erasure halting set can easily be generalised. If the number of remaining erasure-contiguous cells for each erasure is classified by the vector $\begin{pmatrix} x & y & z & w \end{pmatrix}$, then the total number of ways of creating a halting set for the candidate set $\{c_1, c_2\}$ is given by:

$$(4 - c_1 + c_2)^{x+y+z+w} \tag{7.4}$$

Equation (7.4), however, does not take into account the possibility of intersecting erasure-adjacent cells combined with edge-contiguous erasures. This final inclusion results in a general equation that returns the total number of ways of creating a 4-erasure halting set for the candidate set $\{c_1, c_2\}$ and is given by:

$$(4 - c_1 + c_2)^{x+y+z+w-2d} \times (4 - c_1 + c_2 - 1)^d \tag{7.5}$$

$$= (4 - c_1 + c_2)^{x+y+z+w-2d} \times (3 - c_1 + c_2)^d \tag{7.6}$$

where the values x, y, z and w represent values from the vector $\begin{pmatrix} x & y & z & w \end{pmatrix}$ and relate to the number of remaining erasure-adjacent values associated with each of the erasures (intersecting or disjoint) in the 2×2 sub-square (top-left, top-right, bottom-left and bottom-right respectively). The value d represents the total number of intersecting erasure-adjacent cells and the values c_1 and c_2 are the two candidate set values where $c_1 > c_2$ (without loss of generality). Notice that Equation (7.6) simplifies to Equations (7.1), (7.3) or (7.4) depending on the values assigned to x, y, z, w and d; for example, allowing $x = y = z = w = 4$ and $d = 0$ results in Equation (7.1). The above discussion can be summarised in the following theorem.

Theorem 5. *Given a 2×2 sub-square of erasures, each with candidate set $\{c_1, c_2\}$, where $c_1 > c_2$ without loss of generality, the number of ways of surrounding the sub-square with symbols such* *that a 4-erasure halting set is generated is given by*

$$(4 - c_1 + c_2)^{x+y+z+w-2d} \times (3 - c_1 + c_2)^d$$

where d represents the total number of intersecting erasure-adjacent cells and x, y, z and w are values from the vector $\begin{pmatrix} x & y & z & w \end{pmatrix}$ and communicate the number of remaining erasure-adjacent values associated with each of the erasures (intersecting or disjoint) in the 2×2 sub-square (top-left, top-right, bottom-left and bottom-right respectively)

Proof. The discussion throughout Section 7.2 describes how the given formula is derived. □

7.2.1 Examples of 4-Erasure Halting Sets

7.2.1.1 Example One

In the example shown in Figure 7.1, all erasures are non-edge-contiguous and two-cell-separable. The vectors surrounding the grid are up-down vectors in their original binary alphabet (where only the significant up-down values are included – those surrounding the grid erasures). Although specific length vectors are not shown (as the 4-erasure halting set does not require any deletion of length vector information), all erasures in the 2×2 sub-square in this example have a candidate set $\{1, 2\}$. That is, given row and column length vectors of the form $\begin{pmatrix} a & b & c & d \end{pmatrix}$, all row and column *count* length vectors are of the form $\begin{pmatrix} a & b-1 & c-1 & d \end{pmatrix}$ where $a, b, c, d \in \{0, 1, ..., 15\}$ and $a + b + c + d = 15$, resulting in every erasure having a candidate set of $\{1, 2\}$.

All erasure-adjacent cells contain values that adhere to the constraints described previously and, as a result, the deduction of any of the four erased values is not possible. The above is therefore a typical minimal 4-erasure halting set (minimal describing the fact that additional erasures may occur, but it is only those erasures outlined that are of significance in creating the halting set).

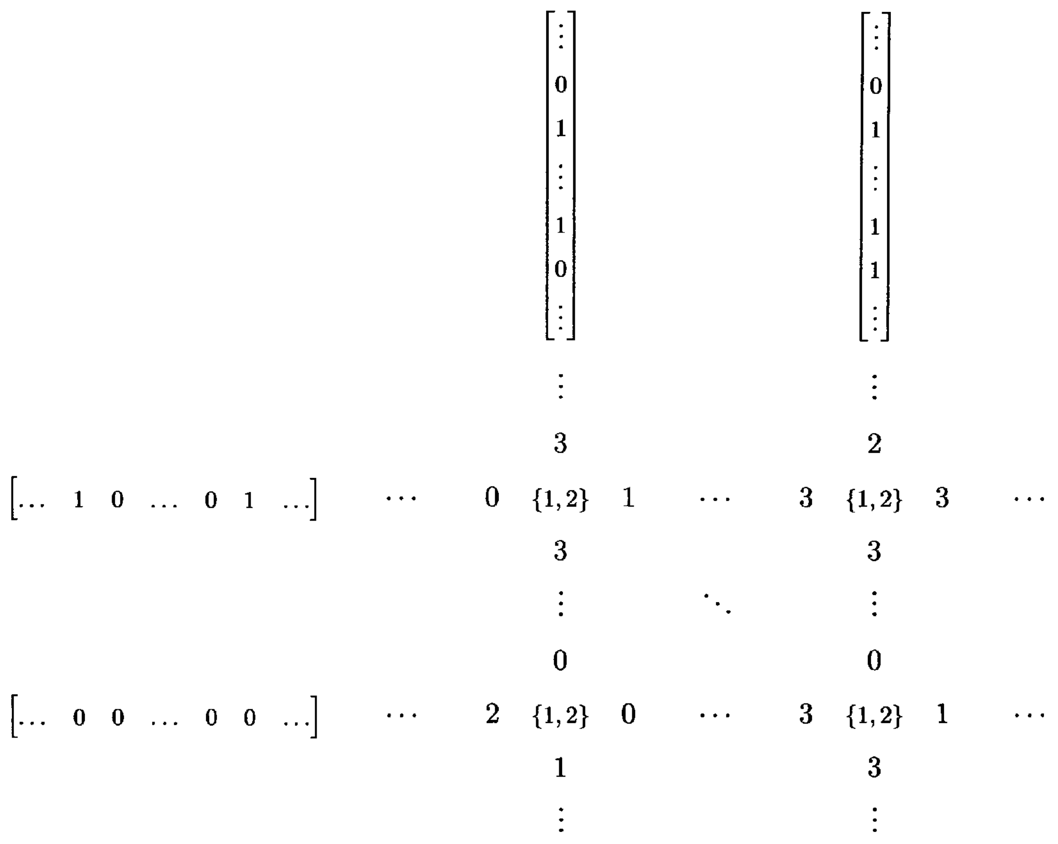


Figure 7.1: *An example of a non-edge-contiguous, two-cell-separable 4-erasure halting set with candidate set* $\{1, 2\}$.

7.2.1.2 Example Two

Figure 7.2 shows a second example of a two-cell-separable, non-edge-contiguous 4-erasure halting set. As in the previous example, the vectors in the figure surrounding the grid are up-down vectors in their original binary alphabet and specific length vectors are not shown. All erasures in the 2×2 sub-square in this example have a candidate set $\{1, 3\}$. It is therefore known that given row and column length vectors of the form $\begin{pmatrix} a & b & c & d \end{pmatrix}$, all row and column count vectors are of the form $\begin{pmatrix} a & b-1 & c & d-1 \end{pmatrix}$ where $a, b, c, d \in \{0, 1, ..., 15\}$ and $a+b+c+d = 15$, resulting in every erasure having a candidate set of $\{1, 3\}$.

All erasure-adjacent cells contain values that adhere to the constraints described previously and, as a result, the deduction of any of the four erased values is not possible. Note that no erasure-adjacent cell contains the value 2, as any value that falls between the two candidates would allow for differentiation between the values' placements. The above is therefore a second typical minimal 4-erasure halting set.

$$\begin{array}{cccccccccc}
 & & & \begin{bmatrix}\vdots\\1\\0\\\vdots\\1\\0\\\vdots\end{bmatrix} & & & & \begin{bmatrix}\vdots\\0\\0\\\vdots\\1\\0\\\vdots\end{bmatrix} & & \\
 & & & \vdots & & & & \vdots & & \\
 & & & 0 & & & & 3 & & \\
\begin{bmatrix}\cdots & 0 & 0 & \cdots & 1 & 0 & \cdots\end{bmatrix} & \cdots & 3 & \{1,3\} & 1 & \cdots & 0 & \{1,3\} & 1 & \cdots \\
 & & & 1 & & & & 0 & & \\
 & & & \vdots & & \ddots & & \vdots & & \\
 & & & 0 & & & & 0 & & \\
\begin{bmatrix}\cdots & 1 & 0 & \cdots & 0 & 0 & \cdots\end{bmatrix} & \cdots & 0 & \{1,3\} & 0 & \cdots & 3 & \{1,3\} & 1 & \cdots \\
 & & & 1 & & & & 0 & & \\
 & & & \vdots & & & & \vdots & &
\end{array}$$

Figure 7.2: *An example of a non-edge-contiguous, two-cell-separable 4-erasure halting set with candidate set* $\{1, 3\}$.

7.2.1.3 Example Three

As with the previous examples, the vectors in the Figure 7.3 surrounding the grid are up-down vectors in their original binary alphabet and specific length vectors are not shown. This example however contains intersecting erasure-adjacent cells (highlighted in grey shading) and has edge-contiguous erasures. All erasures in the 2×2 sub-square have a candidate set $\{0, 2\}$. All erasure-adjacent cell values (intersecting or disjoint) conform to the constraints discussed previously and as a result the above is an example of a 4-erasure minimal halting set.

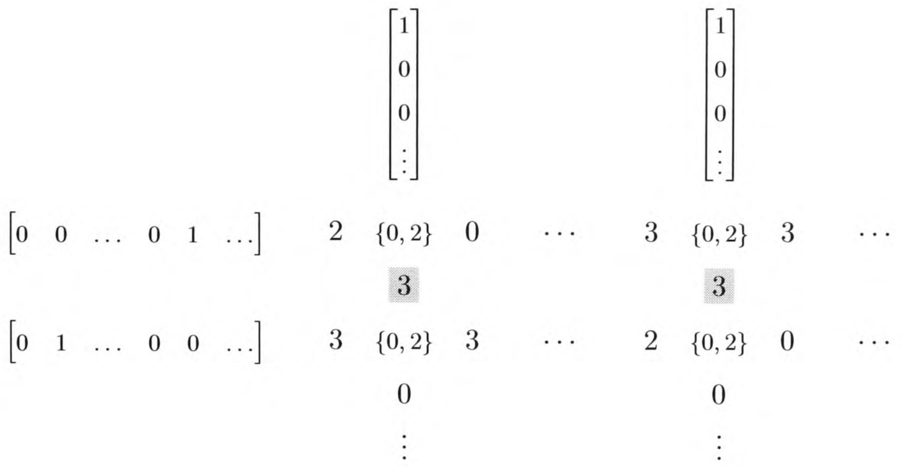


Figure 7.3: *An example of a non-edge-contiguous, 4-erasure halting set with intersecting erasure-adjacent cells (shaded grey); the erasures each have candidate set* $\{1, 3\}$.

7.3 5-Erasure (Minimal) Halting Sets

A 5-erasure halting set consists of a single grid erasure accompanied by two length vector erasures in each of the associated row length vector and column length vector.

Each 4-ary length vector is of the form $\begin{pmatrix} a & b & c & d \end{pmatrix}$ where each of the symbols a, b, c and d is a pair of quaternary symbols representing a value from the set $\{0, 1, ..., 15\}$. For example, let $a = 23$ (23 represents $2 \times 4^1 + 3 \times 4^0 = 11$ in decimal) which communicates that there are 11 occurrences of the value '0' in the row or column associated with the length vector (as a is in the first, or '0' position of the vector). Each quaternary digit in a string of quaternary symbols is referred to here as a *crumb*. Throughout this halting set classification the *lower-order crumb* refers to the quaternary digit in the 4^0 position of the quaternary length vector symbol pair, whereas the *higher-order crumb* refers to the quaternary digit in the 4^1 position of the quaternary pair.

In terms of the position of the length vector erasures within a 5-erasure halting set, erasures must occur within the lower-order crumb of the length vector symbol pair to generate a halting set. There must be two lower-order crumb erasures in both the row length vector and the column length vector associated with the single grid erasure. These lower-order crumb erasures must occur within the same length vector symbol pairs in the row and column length vectors in order to generate a candidate set of size two for the single grid erasure.

However, not all length vectors satisfying the prescribed erasure position conditions form 5-erasure halting sets. There are two generalised exceptions to the erasure pattern described above that do not result in a halting set consisting of 5 erasures:

1. Considering a single length vector (either row or column) in quaternary form $\begin{pmatrix} a & b & c & d \end{pmatrix}$

where each of the symbols a, b, c and d is a pair of quaternary symbols, lower-order crumb erasures may occur in any two of the four length vector symbol pairs a, b, c and d. Let the quaternary symbols that are to be erased by the lower-order crumb erasures (in any two of the symbol pairs a, b, c and d) be labelled e_1 and e_2 (order of e_1 and e_2 is unimportant). If $e_1 = e_2 = 0$ or $e_1 = e_2 = 3$ then a 5-erasure halting set will not be formed. This is because the length vector containing the lower-order crumb erasures, combined with the total number of symbol occurrences for the row or column in question, would allow for the recovery of the length vector and hence the deduction of the single erased grid symbol.

For example, given the decimal length vector $\begin{pmatrix}3 & 7 & 1 & 4\end{pmatrix}$, which is equivalent to quaternary length vector $\begin{pmatrix}03 & 13 & 01 & 10\end{pmatrix}$, applying erasures to the lower-order crumb of the symbol pair for position '0' and symbol pair for position '1' would result in the length vector $\begin{pmatrix}0e & 1e & 01 & 10\end{pmatrix}$ and a candidate set of $\{0, 1\}$ for the single grid erasure. There are only two possibilities for the row/column count (if the grid values are unknown), which in decimal vector format are $\begin{pmatrix}3 & 6 & 1 & 4\end{pmatrix}$ or $\begin{pmatrix}2 & 7 & 1 & 4\end{pmatrix}$ – one less in each of the two candidate positions. The row or column count is required to increase by one in either position '0' or position '1' when the single grid erasure is recovered and it can be observed that this is only possible in one way without altering the value of the higher-order crumb for each of the two possible row/column counts. This results in the recovery of the decimal length vector $\begin{pmatrix}3 & 7 & 1 & 4\end{pmatrix}$ and hence the deduction of the single missing grid value.

It should be noted that the case when $e_1 = e_2 = 0$ may indicate lower-order crumb erasures to two values that may be zero in the decimal length vector. A halting set over 5 symbols cannot be formed in this case either, as neither of the symbols occur in the row or column associated with the length vector in question. For example, given $\begin{pmatrix}0e_1 & 0e_2 & 20 & 13\end{pmatrix}$ it is obvious that $e_1 = e_2 = 0$ as $20 + 13 = 33$ in quaternary, which is 15 in decimal – no more values are required to complete the length vector.

2. Again, considering a single length vector (either row or column) in quaternary form $\begin{pmatrix}a & b & c & d\end{pmatrix}$ where each of the symbols a, b, c and d is a pair of quaternary symbols, lower-order crumb erasures may occur in any two of the four length vector symbol quaternary pairs a, b, c and d. Let the quaternary symbols that are to be erased by the lower-order crumb erasures be labelled e_1 and e_2 (order of e_1 and e_2 is unimportant). If either of the following is true, then the formation of a 5-erasure halting set is dependent on the total number of symbol occurrences for the row or column of the grid in question:

   (a) $e_1 = 0$ and $e_2 \in 1, 2, 3$

   (b) $e_1 = 3$ and $e_2 \in 0, 1, 2$

More specifically, allow the total number of symbol occurrences in the row or column associated with the grid erasure to be represented by a vector in quaternary form, in exactly the same way as the length vectors are. If the lower order crumb in this row/column count vector that corresponds to erasure e_1 is 3 then a 5-erasure halting set is not formed. This is also the case for the crumb corresponding to e_2 in the cases where $e_2 = 0$ or $e_2 = 3$; that is a halting set is not formed if the lower-order crumb corresponding to e_2 in the row/column count vector is equal to 3.

There are always two potential row/column count vectors however for each occurrence of a single grid erasure with two candidates – the second of these will not contain 3 in the position corresponding to e_1 (and e_2 when $e_2 = 0$ or $e_2 = 3$) and will hence always form a 5-erasure halting set.

For example, the decimal length vector $\begin{pmatrix}4 & 4 & 2 & 5\end{pmatrix}$ is equivalent to $\begin{pmatrix}10 & 10 & 02 & 11\end{pmatrix}$ in quaternary numbering. If lower-order crumb erasures occur to length vector symbol pairs in the '1' and '3' positions then the resultant partially completed length vector is $\begin{pmatrix}10 & 1e & 02 & 1e\end{pmatrix}$ and the candidate set for the single grid erasure is $\{1, 3\}$. There are two possibilities for the row/column count vector associated with this erasure; the row/column count vector gives the number of occurrences of each symbol in a given row or column neglecting any erasures occurring in the row or column – in this case only a single erasure has occurred in the row/column hence the sum of the row/column count vector's components should equal 14 $(n - 1)$. The two possibilities for the row/column count vector in this case are $\begin{pmatrix}4 & 3 & 2 & 5\end{pmatrix}$ and $\begin{pmatrix}4 & 4 & 2 & 4\end{pmatrix}$ in decimal form, where the count vectors differ from the original length vector in positions '1' and '3'. Notice that the value 3 in position '1' of the first vector must be increased to 4 in order for the higher-order crumb to correspond to that observed in the partially completed length vector. Therefore, due to the 3 in this position, a 5-erasure halting set cannot be formed as it is recoverable. For the second of the two row/column count length vectors, an increase in either of the two erased positions is plausible and hence a halting set is formed.

Provided the row and column length vectors do not fall into the excepted cases outlined above, then the erasure patterns and the values of such positions before erasure may enable the formation of a 5-erasure halting set. The patterns and values must however adhere to one further condition: all grid-erasure-adjacent values must be such that the up-down vector information does not allow for the deduction of which of the two candidates in the candidate set should be assigned to the grid erasure.

More specifically, let $C = \{c_1, c_2\}$, where $c_1 \neq c_2$ and $c_1, c_2 \in \{0, 1, 2, 3\}$, be the candidate set for the erased grid value as generated by the row and column length vectors. If the erasure occurs within cell (i, j) of the grid, then:

1. The erasure-adjacent cells $(i - 1, j)$ and $(i, j - 1)$ may take any value that is greater than or equal to $max(c_1, c_2)$ or strictly less than $min(c_1, c_2)$.

2. The erasure-adjacent cells $(i+1, j)$ and $(i, j+1)$ may take any value that is strictly greater than $max(c_1, c_2)$ or less than or equal to $min(c_1, c_2)$.

In cases where the erasure pattern and pre-erasure length vector values are not outlined in the excepted cases, and all grid-erasure-adjacent values (if they exist, that is the erasure is not edge-contiguous) adhere to the above constraints, a 5-erasure halting set is formed.

7.3.1 Examples of 5-Erasure Halting Sets

7.3.1.1 Example One

The example in Figure 7.4 details a 5-erasure halting set, where the single grid erasure has candidate set $\{0, 1\}$.

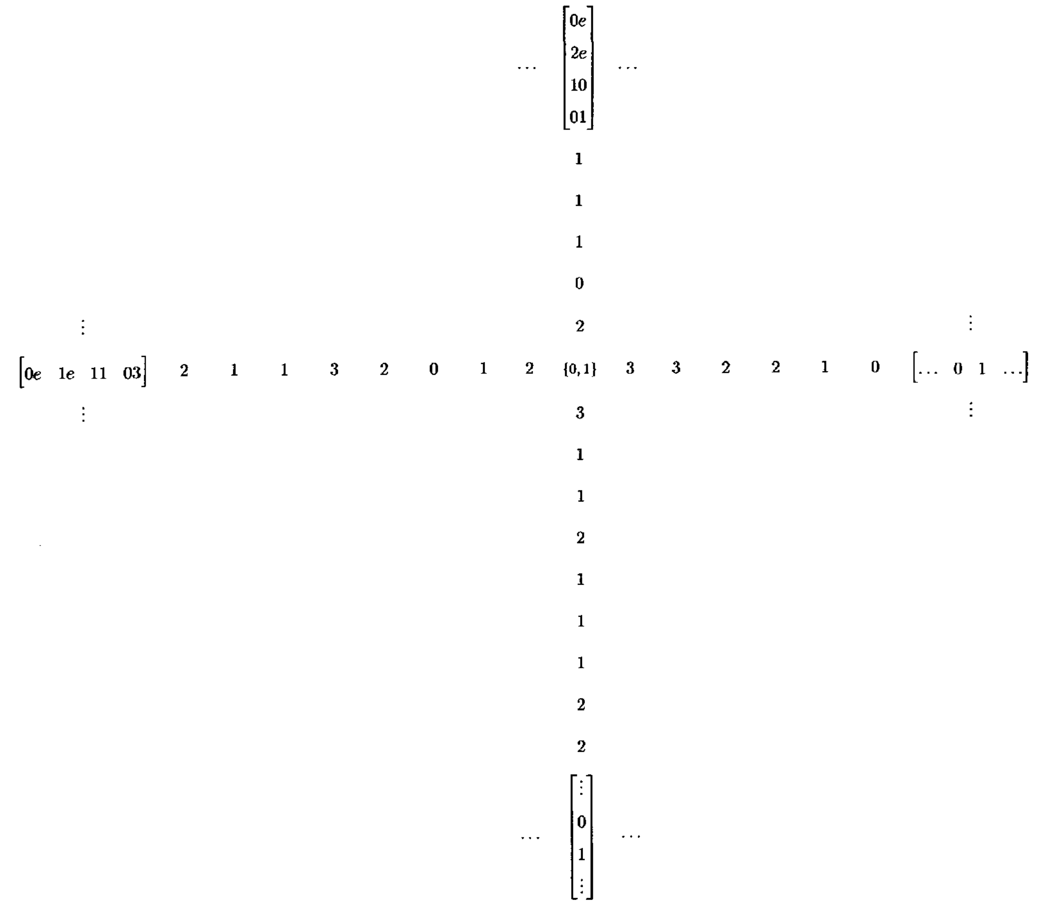


Figure 7.4: *An example of a 5-erasure halting set with candidate set* $\{0, 1\}$.

The length vectors (with erasures) are given above the grid column and to the left of the grid row. Below and to the right of the grid are the up-down vectors, left in binary alphabet, detailing the values of significance only. All grid-erasure-adjacent values contain values from $\{2,3\}$ and hence the up-down vectors cannot be exploited to deduce the missing symbol.

7.3.1.2 Example Two

The example in Figure 7.5 details a second 5-erasure halting set, where the single grid erasure has candidate set $\{1,3\}$. As with example one, the length vectors (with erasures) are given above the grid column and to the left of grid row. Below and to the right of the grid are the binary up-down vectors detailing the bits of significance only. All grid-erasure-adjacent values contain values that adhere to the erasure-adjacent conditions described in the classification.

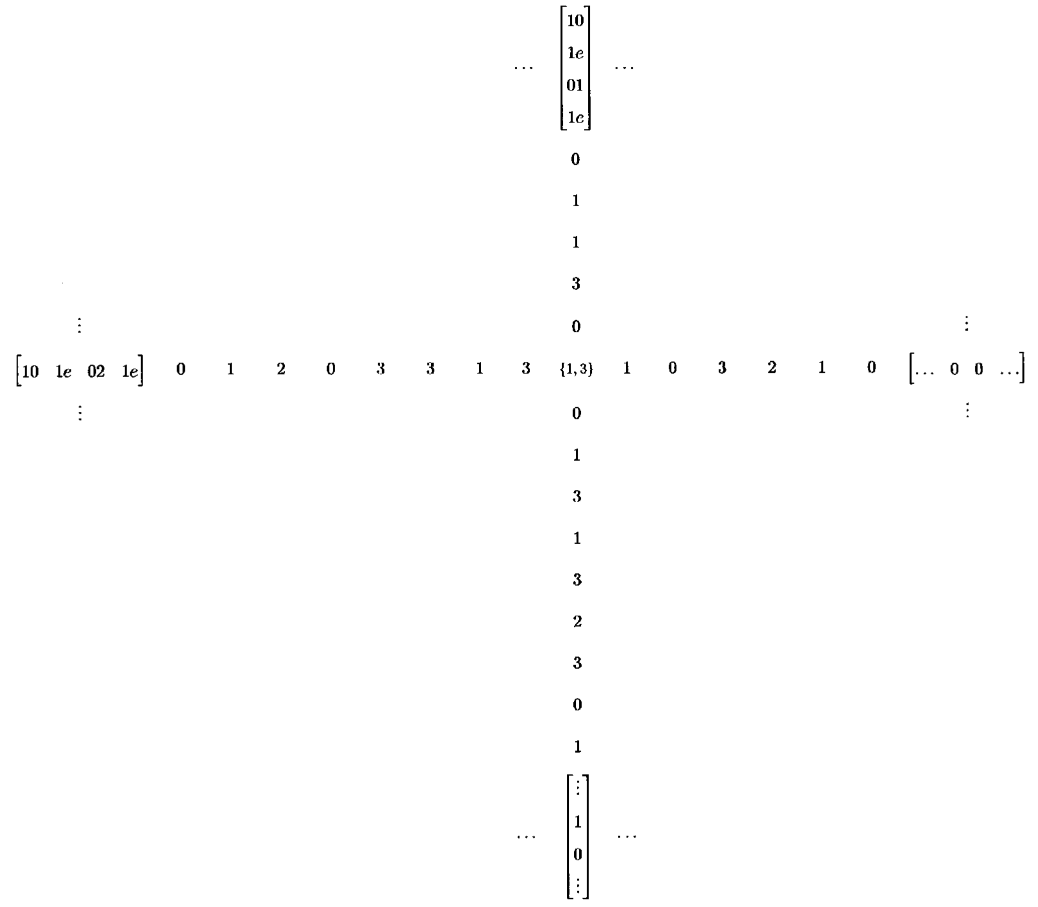


Figure 7.5: *An example of a 5-erasure halting set with candidate set* $\{1,3\}$.

7.4 (4+1)-Erasure (Minimal) Halting Sets

(4+1)-erasure halting sets are a subset of 5-erasure halting sets; they are formed by an additional 4-ary symbol erasure occurring alongside a 2×2 Latin sub-square pattern of erasures that typically constitutes a 4-erasure halting set. In order for the (4+1)-erasure halting set to be considered *minimal*, as is required, the 2×2 sub-square pattern alone must not form a 4-erasure halting set. In order to form a (4+1)-erasure halting set, the fifth erasure must occur in the up-down vector component in positions relating to one of the erasures in the 2×2 sub-square and at least one of its adjacent cells. For the outlined (4+1)-erasure halting set to be considered minimal, the 2×2 erasure adjacent cell(s) discussed must contain a *disallowed* value, where a disallowed value is any value that allows for the deduction of one of the 2×2 sub-square erasures, hence any value not adhering to the constraints discussed previously for 4-erasure halting sets. The requirements for a cell to be considered to contain a disallowed value are given below.

Consider the general case of a 4-erasure halting set, where all four erasures have four distinct erasure-adjacent cells and there is no intersection of erasure-adjacent cells. This general case requires erasures to be two-cell-separable and non-edge-contiguous. For this general case, let $C = \{c_1, c_2\}$, where $c_1 \neq c_2$ and $c_1, c_2 \in \{0, 1, 2, 3\}$, be the candidate set for each of the four erased values. If the erasure occurs within cell (i, j) of the grid, then:

1. The erasure-adjacent cells $(i-1, j)$ or $(i, j-1)$ are considered to contain a disallowed value if the cell contains a value x that satisfies the inequality $min(c_1, c_2) \leq x < max(c_1, c_2)$.
2. The erasure-adjacent cells $(i+1, j)$ and $(i, j+1)$ are considered to contain a disallowed value if the cell contains a value x that satisfies the inequality $min(c_1, c_2) < x \leq max(c_1, c_2)$.

In cases where the erasure-adjacent cells intersect (are adjacent to more than one erasure), that is where erasures are not two-cell-separable, a further constraint needs to be considered:

3. If a pair of row or column erasures occur in cells (i_1, j_1) and (i_2, j_2) of the grid, and $i_1 = i_2 \pm 2$ or $j_1 = j_2 \pm 2$ then the cell that is horizontally or vertically adjacent to both erasures is considered to contain a disallowed value if the cell contains a value x satisfying the inequality $min(c_1, c_2) \leq x \leq max(c_1, c_2)$.

In cases where an erasure (i, j) is edge-contiguous, at least one of the erasure-adjacent cells $(i-1, j)$, $(i, j-1)$, $(i+1, j)$ and $(i, j+1)$ does not exist, and any constraints relating to this erasure-adjacent cell may be ignored.

Typically, if a single cell that is adjacent to one (or two in the intersecting case) of the 2×2 sub-square erasures contains a disallowed value, then the erasure of the up-down vector symbol that communicates the relationship between the 2×2 sub-square erasure and the erasure-adjacent cell results in a (4+1)-erasure halting set. There are exceptions to this, however; it is possible for a (4+1)-erasure halting set to be formed when two disallowed values occur in the erasure-adjacent cells. For this to occur, the two erasure-adjacent cells must both be horizontal neighbours to a single erasure or vertical neighbours to a single erasure. The erasure in question also needs to be in an even row or column position within the grid depending on whether they are horizontally or vertically bordered by disallowed values. This exception is explained by the fact that when initially constructed, the up-down vector is represented in bits where each bit communicates the relative increases or decreases (or equality) between successive elements in a row or column. The binary vector is translated to a 4-ary vector before transmission and erasures that occur in the vector results in the erasure of pairs of consecutive bits. As a pair of consecutive bits is always erased, two disallowed values can be contained in a single (4+1)-erasure halting set provided the erasure that is surrounded by the disallowed values is in an even position in the grid.

In conclusion, a $(4 + 1)$-erasure halting set is formed when a 2×2 sub-square of erasures is bordered by a single disallowed value (or a pair of disallowed values in specific cases of grid positioning as outlined previously). The inclusion of the additional disallowed value(s) requires that its associated up-down information is erased in order to form a $(4 + 1)$-erasure halting set. Examples of this type of halting set are given below.

7.4.1 Examples of (4+1)-Erasure Halting Sets

7.4.1.1 Example One

In the example shown in Figure 7.6, all erasures are non-edge-contiguous and two-cell-separable. The vectors surrounding the grid are up-down vectors in their original binary alphabet (where only the significant up-down values and erasures are included – those surrounding the grid erasures). Although specific length vectors are not shown (as the (4+1)-erasure halting set does not require any deletion of length vector information), all erasures in the 2×2 sub-square have a candidate set $\{1, 2\}$. That is, given row and column length vectors of the form $\begin{pmatrix} a & b & c & d \end{pmatrix}$, all row and column *count* length vectors are of the form $\begin{pmatrix} a & b-1 & c-1 & d \end{pmatrix}$ where $a, b, c, d \in \{0, 1, ..., 15\}$ and $a + b + c + d = 15$, resulting in every erasure having a candidate set of $\{1, 2\}$.

All but one of the erasure-adjacent cells contain values that adhere to the constraints described previously, the disallowed erasure-adjacent value is highlighted in yellow. The up-down vector symbols that communicate the relative increase or decrease (or equality) between the disallowed value and the 2×2 sub-square erasure has been erased and, as a result, the deduction of any of the four erased values in the sub-square is not possible. The above is therefore a typical minimal (4+1)-erasure halting set.

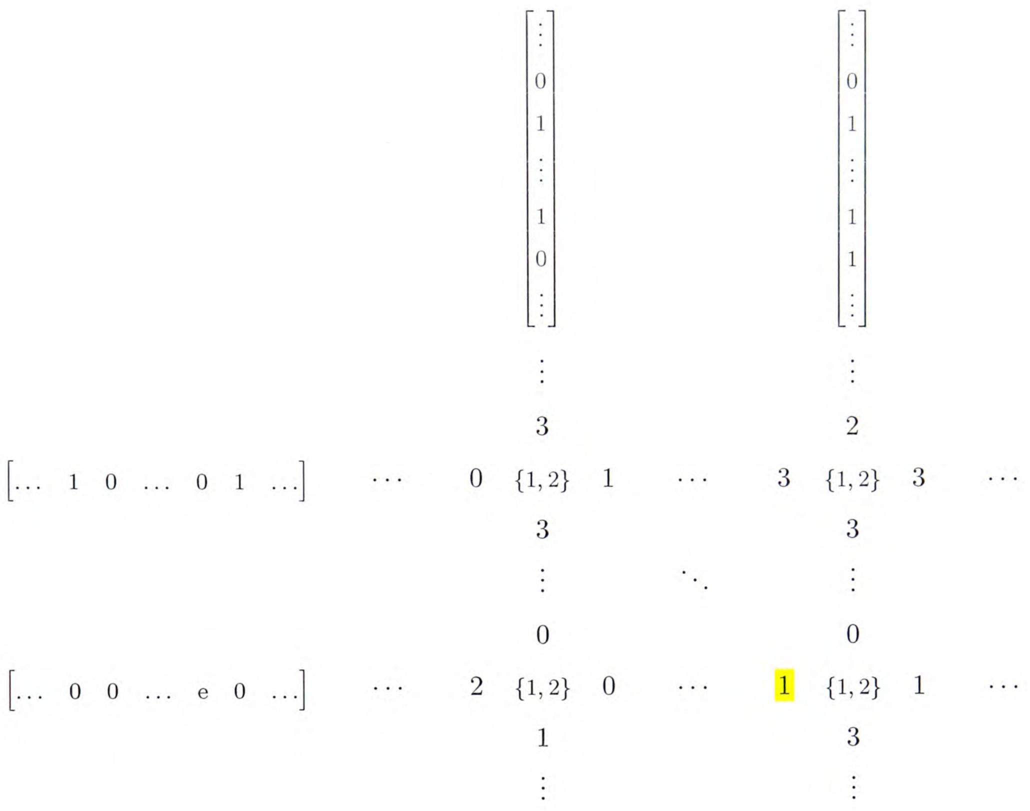


Figure 7.6: *An example of a non-edge-contiguous, two-cell-separable (4+1)-erasure halting set with candidate set* $\{1,2\}$ *and disallowed value '1' highlighted in yellow.*

7.4.1.2 Example Two

Figure 7.7 gives an example of a (4+1)-erasure halting set in which two disallowed values surround a single erasure of the 2×2 sub-square. As all binary up-down vector information is converted to 4-ary pre-transmission, erasures occur to pairs of binary symbols as opposed to individuals; therefore, two disallowed values may occur in a single (4+1)-erasure halting set provided the 2×2 sub-square erasure is in an even row or column position within the grid (depending on whether the disallowed values horizontally or vertically neighbour the sub-square erasure).

In contrast to Figure 7.6 given in the previous example, Figure 7.7 shows the entire rows and columns associated with each erasure in the 2×2 sub-square and gives the complete partially-erased up-down vectors to the left and upper of the partial grid. Although specific length vectors are not shown (as the (4+1)-erasure halting set does not require any deletion of length vector information), all erasures in the 2×2 sub-square have a candidate set $\{2,3\}$. That is, given row and column length vectors of the form $\begin{pmatrix} a & b & c & d \end{pmatrix}$, all row and column *count* length vectors are of the form $\begin{pmatrix} a & b & c-1 & d-1 \end{pmatrix}$ where $a, b, c, d \in \{0, 1, ..., 15\}$ and $a+b+c+d = 15$, resulting in every erasure having a candidate set of $\{2, 3\}$.

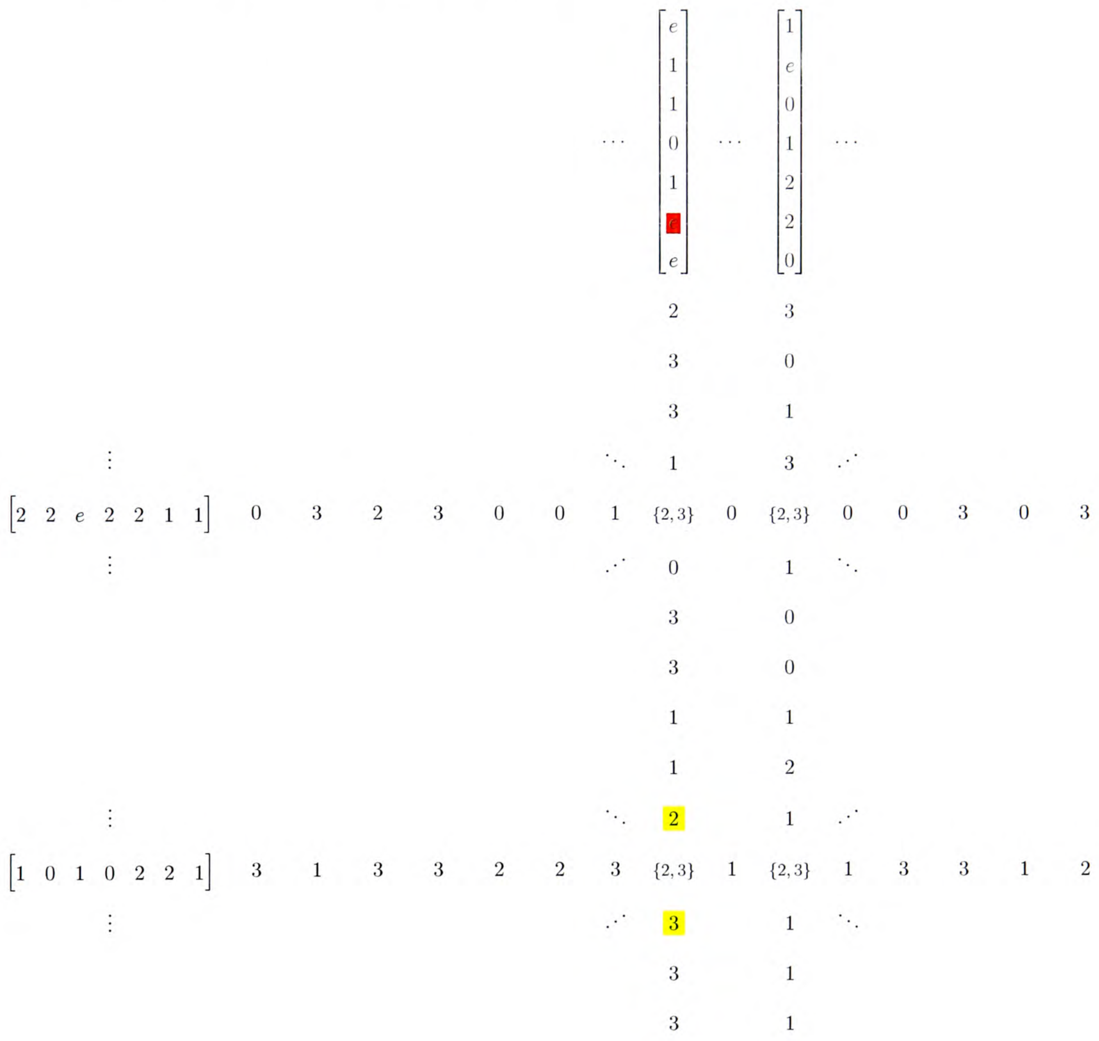


Figure 7.7: *An example of a non-edge-contiguous, two-cell-separable (4+1)-erasure halting set with candidate set* $\{2, 3\}$ *and a pair of erasure-adjacent disallowed values.*

The two disallowed values bordering one of the 2×2 sub-square erasures are highlighted in yellow; the up-down vector information relating to these disallowed values is erased (highlighted in red) and, as a result, the deduction of any of the four erased values in the sub-square is not possible. The above is therefore a typical minimal (4+1)-erasure halting set.

7.5 Deriving Lower Bounds for Prevalence of Halting Sets

Using the classification of 4-erasure and 5-erasure halting sets described in Section 7.2 and 7.3 respectively, it is possible to calculate a lower bound for the probability of an order 15 monoalphabetic GLUV containing at least one halting set of the 4-erasure class or of the 5-erasure class. An estimate at the prevalence of halting sets would allow conclusions to be drawn on the suitability of monoalphabetic GLUV for erasure correction.

7.5.1 Lower Bound for the Prevalence of 4-Erasure Halting Sets

A four erasure halting set consists of four erasures, arranged in a 2×2 sub-square pattern with no additional meta-data erasures. The candidate sets for all four erasures are equal and of size two and consequently each row and column of the sub-square needs to be assigned one of each candidate value. The values surrounding each of the erasures are each such that its value combined with the relevant up-down vector information does not result in a deduction of one of the four erased values. The number of appropriate surrounding values for each candidate set were detailed previously in Table 7.1.

A lower bound for the probability of an order 15 monoalphabetic GLUV containing at least one halting set of the 4-erasure class can be calculated using information detailed in Section 7.2 along with the counting of the number of independent 4-erasure halting sets that may be placed within a single order 15 grid. The lower bound will concentrate only on non-edge-contiguous, two-cell-separable 4-erasure halting set. Examination reveals that a maximum of five 2×2 sub-squares can be placed independently on a 15×15 grid, an example of which is shown in Figure 7.8. The number of non-independent or intersecting sub-squares would be expected to be much higher and is difficult to enumerate.

Given that there are potentially at most five independent 2×2 sub-squares, the probability of at least one 4-erasure halting set being contained within the 15×15 grid is given by 1 -[P(sub-square s_i does not form a halting set)]^5 where s_1 to s_5 represent each of the potential sub-squares and s_i is a general choice of sub-square ($i \in 1, 2, \ldots, 5$). The probability of a chosen s_i not forming a halting set is given by 1 - P(s_i forms a halting set) and it is hence the probability P(s_i forms a halting set) that needs to be calculated.

As the possibilities for erasure-adjacent values varies depending on the specific candidate set, this lower bound will concentrate only on the candidate set type that results in the most significant values. Table 7.1 shows that the erasure-adjacent values for a grid erasure with candidate set $\{0,1\}$, $\{1,2\}$ or $\{2,3\}$ may each be filled with any choice of three values from the alphabet $\{0,1,2,3\}$ without resulting in erasure recovery and as a result contribute most significantly to the lower bound. Candidate sets $\{0,2\}$, $\{1,3\}$ and $\{0,3\}$ only being able to be surrounded by a total of two, two and one values respectively. Considering only candidates of the 'consecutive' type, that is sets $\{0,1\}$, $\{1,2\}$ and $\{2,3\}$, the probability that a general choice of sub-square s_i

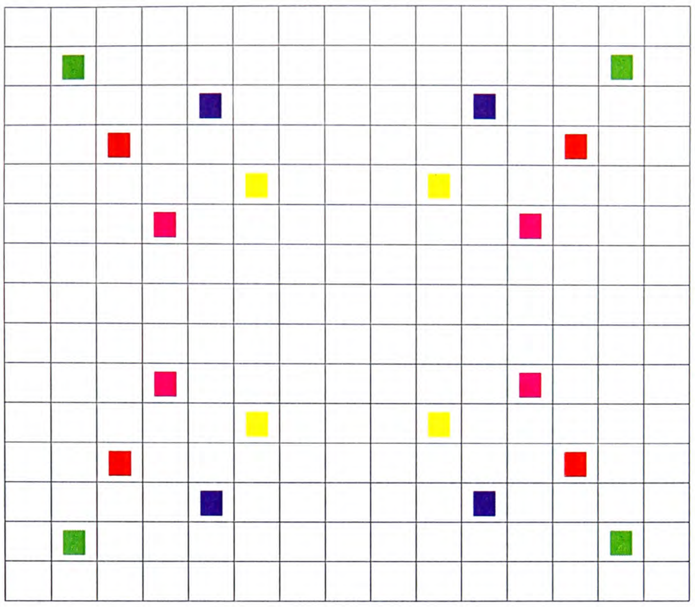


Figure 7.8: *An example of an arrangement of five independent 2×2 sub-squares where cells of differing colour represent one of each of the five sub-squares.*

forms a halting set is given by:

$$\mathrm{P}(s_i \text{ forms a halting set}) = \left(\frac{3}{4}\right)^{16} \times \frac{6}{256} \times p^4 = \frac{3^{17}}{2^{39}}p^4 \tag{7.7}$$

where the fraction $\left(\frac{3}{4}\right)^{16}$ describes the number of ways of surrounding all four erasures of the sub-square with appropriate values such that no deduction of candidate values can be made; the fraction relates only to candidates sets of the 'consecutive' type as this type produce the most significant values. The fraction $\frac{6}{256}$ describes the fact that there are 6 potential 2×2 erasure patterns constructible from 'consecutive' type candidate sets, out of a total of $4^4 = 256$ ways in total of randomly choosing four sub-square values. Finally, p^4 indicates that all four cells of the sub-square must be erased to form a halting set.

Given that P(s_i forms a halting set) has been calculated, 1 - P(s_i forms a halting set) communicates the probability that s_i does not form a halting set. Using the formula derived previously, the probability of at least one halting set being contained within the grid is given by 1 -[P(sub-square s_i does not form a halting set)]^5, it can be calculated that:

$$\mathrm{P}(\text{at least one halting set in GLUV}) = 1 - \left(1 - \frac{3^{17}}{2^{39}}p^4\right)^5 \tag{7.8}$$

The value of the probability given in Equation (7.8) is dependent on the erasure probability p. Substituting a value of $p = 0.3$ into Equation (7.8) gives P(at least one halting set in GLUV at $p = 0.3) = 9.5135 \times 10^{-6}$, a rather small value which does not reflect the number of 4-erasure halting sets observed during simulation. Increasing the probability p to closer to the Shannon capacity for the scheme (around $p = 0.66$) to $p = 0.6$, returns P(at least one halting set in GLUV at $p = 0.6$)= 1.5221×10^{-4}. At $p = 0.6$, a lower bound estimate on the prevalence of 4-erasure halting sets indicates the 0.001% of GLUVs contain at least one halting set, post-transmission. These results do not reflect the prevalence already observed during simulation, with 4% of grids containing one halting set at $p = 0.15$ for 1000 iterations. Clearly the number and/or type of halting sets being counted are not adequate to construct a satisfactory lower bound.

7.5.2 Lower Bound for the Prevalence of 5-Erasure Halting Sets

As stated previously, a 5-erasure halting set consists of a single grid erasure and four length vector erasures; where each length vector erasure occurs in the lower-order crumb of the length vector's 4-ary symbol pairing for a given length vector position. Erasures in both the row and column length vectors must occur in the same lower-order crumb positions to generate a candidate set of size two for the single grid erasure.

Although all 5-erasure halting sets adhere to this erasure pattern, some combinations of lower-order crumb values within a row or column length vector permit the deduction of the single erased grid value and hence do not form a 5-erasure halting set. The lower-order crumb pairings in both the row length vector and column length vector must therefore be such that they do not facilitate the deduction of the erased grid symbol, and the probability of both length vectors simultaneously satisfying this condition needs to be calculated.

Begin by considering just one of the length vectors. Given a single length vector (let us assume row for ease of explanation, although choice is inconsequential), there are $4^2 = 16$ possible pre-erasure lower order crumb pairings: $(0,0)$, $(0,1)$, $(0,2)$, $(0,3)$, $(1,0)$, $(1,1)$, $(1,2)$, $(1,3)$, $(2,0)$, $(2,1)$, $(2,2)$, $(2,3)$, $(3,0)$, $(3,1)$, $(3,2)$ and $(3,3)$. Given that the single erased grid symbol must be assigned a value corresponding to one of the pair of lower-order crumb erasures, there are two possible counts for the entire row of values containing the grid erasure – each equally probable. It is possible that none, one or both of these row counts combined with the lower-order crumb pairing will form a halting set.

For example, if the lower order crumb pairing in question was $(0,1)$ then an example of such a length vector would be $\begin{bmatrix}10 & 11 & 03 & 03\end{bmatrix}$ pre-erasure and $\begin{bmatrix}1e & 1e & 03 & 03\end{bmatrix}$ post-transmission, where the lower-order crumbs of positions '0' and '1' constitute the lower-order crumb erasure pairing. Being aware of the post-transmission vector, the actual row count for the row containing the single grid erasure could be either $\begin{bmatrix}3 & 5 & 3 & 3\end{bmatrix}$ or $\begin{bmatrix}4 & 4 & 3 & 3\end{bmatrix}$ and as the grid symbols are randomly generated we can assume that each of the two counts are equally probable. If the first of the two row counts was the actual row count, then the single grid erasure could be deduced as the post-transmission vector $\begin{bmatrix}1e & 1e & 03 & 03\end{bmatrix}$ communicates that between four and seven zeros must occur in the row – a halting set cannot be formed. If however the second of the two row counts was the actual row count, then the single grid erasure could not be deduced and therefore a halting set could be formed.

Examining each of the ten different lower-order crumb erasure pairings (conjugate pairings such as (1, 2) and (2, 1) yield halting sets equally frequently), a symmetric matrix can be constructed that communicates the frequency of halting set formation for specific lower-order crumb erasure pairings. A '0' in the matrix indicates that neither of the two row counts relating to the lower-order crumb erasure pairing allow halting set formation; that is the single erased grid value can be deduced regardless of the row count. A '$\frac{1}{2}$' indicates that a halting set is formed on half of the occasions for the specific lower-order crumb erasure pairing; that is that one of the two row counts combined with the lower-order crumb pairing forms a halting set. Finally, a '1' in the matrix indicates that both row counts relating to the lower-order crumb erasure pairing generate halting sets; the single grid value can never be deduced.

$$\begin{array}{c|cccc|} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 1 & \frac{1}{2} & 1 & 1 & \frac{1}{2} \\ 2 & \frac{1}{2} & 1 & 1 & \frac{1}{2} \\ 3 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \end{array}$$

Therefore, of the sixteen potential pairings for a single length vector, two pairings never form halting sets, ten pairings would be expected to form halting sets on half of the occasions and the final four pairings always forms halting sets. There is therefore a probability of $(\frac{2}{16} \times 0) + (\frac{10}{16} \times \frac{1}{2}) + (\frac{4}{16} \times 1) = \frac{9}{16}$ that the row length vector values and lower-order crumb erasures satisfy the necessary conditions for potential halting set formation.

The $\frac{9}{16}$ value obtained relates only to the row length vector, the column length vector needs to also be considered in order to determine the probability of both of the length vectors being satisfactory for halting set formation. The probability of the column length vector satisfying the conditions is the same as that calculated for the row length vector, and consequently the probability of both length vectors satisfying the necessary lower-order crumb conditions and hence of forming a 5-erasure halting set is:

$$\text{P(length vectors conducive with halting set formation)} = \left(\frac{9}{16}\right)^2 = \frac{81}{256} \qquad (7.9)$$

However, the lower bound for the probability of an order 15 monoalphabetic GLUV containing at least one halting set of the 5-erasure class is dependent on other aspects of the structure in addition to the probability of the length vector erasures meeting the halting set conditions. The probability of all values surrounding the single grid erasure being such that they do not allow for the deduction of the erased value must be calculated, and the number of independent 5-erasure halting sets that can be placed on an order 15 grid must also be identified.

It can be identified that the maximum number of independent 5-erasure halting sets that can exist within a monoalphabetic GLUV of order 15 is thirteen; where each of the thirteen independent (potential) single grid erasures for each of the halting sets occurs alone within the row or column of the GLUV grid, with the exception of the first and last row/column of the grid (as edge-contiguity alters the number of erasure adjacent cells that need to be accounted for). An example of such an arrangement is shown diagrammatically in Figure 7.9.

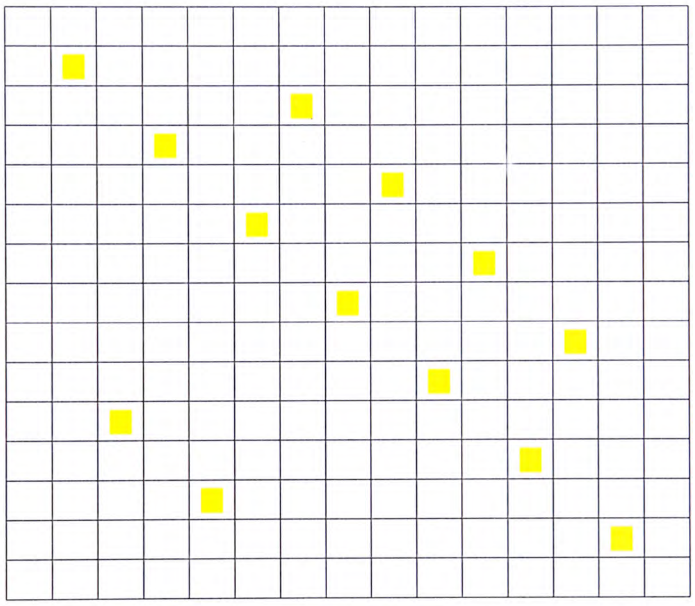


Figure 7.9: *An example of an arrangement of thirteen independent single grid erasures which when combined with length vector erasures allows for thirteen potential 5-erasure halting sets within a single order 15 GLUV.*

Given that there are thirteen independent potential single grid erasures, and hence potential halting sets, the probability of at least one halting set being contained within the grid is given by 1 -[P(single grid erasure s_i does not form a halting set)]^{13} where s_1 to s_{13} represent each of the potential single grid erasures and s_i is a general choice of single grid erasure ($i \in 1, 2, \ldots, 13$). The probability of a chosen s_i not forming a halting set is given by 1 - P(s_i forms a halting set) and it is hence the probability P(s_i forms a halting set) that needs to be calculated.

The probability that an arbitrary choice of s_i forms a halting set is dependent on the probability of the single grid erasure's adjacent cells containing values that do not lead to the grid erasure's deduction, the choice of lower-order crumbs that are erased, the probability of both length vectors being conducive with halting set formation and the probability of all 5-erasures in the halting set being erased. As the possibilities for erasure-adjacent values varies depending on the specific candidate set, this lower bound will concentrate only on the candidate set type that results in the most significant values. The erasure-adjacent values for a grid erasure with candidate set $\{0,1\}$, $\{1,2\}$ or $\{2,3\}$ may each be filled with any choice of three values from the alphabet $\{0,1,2,3\}$ without resulting in erasure recovery and as a result contribute most significantly to the lower bound. Candidate sets $\{0,2\}$ and $\{1,3\}$ and $\{0,3\}$ only being able to be surrounded by a total of two, two and one values respectively. Considering only candidates of the 'consecutive' type, that is sets $\{0,1\}$, $\{1,2\}$ and $\{2,3\}$, the probability that a general choice s_i forms a halting set is given by:

$$\begin{aligned}\text{P}(s_i \text{ forms a halting set}) &= \text{P(erasure-adjacent cells conducive with halting set formation)}\\ &\quad \times\text{P(length vectors conducive with halting set formation)}\\ &\quad \times 3 \times \frac{1}{2} \times p^5 \qquad (7.10)\end{aligned}$$

where the value 3 relates to the choice of the second lower-order crumb to be erased, given that the lower-order crumb corresponding to the single grid erasure is erased. The fraction $\frac{1}{2}$ communicates the proportion of candidate sets of the 'consecutive' type, three out of a total six candidate sets. Finally, p^5 is the probability of all five halting set values being erased, where p is the probability of single 4-ary value erasure. The probability P(length vectors conducive with halting set formation) was given in Equation 7.9 and is $(\frac{9}{16})^2$. The probability P(erasure-adjacent cells conducive with halting set formation) can be easily calculated as only the 'consecutive' case candidate sets are now under consideration; if all cells surrounding the single grid erasure have three potential values of a possible alphabet of four, then the probability of all four erasure-adjacent being conducive with halting set formation is $(\frac{3}{4})^4$. Therefore,

$$\text{P}(s_i \text{ forms a halting set}) = \left(\frac{3}{4}\right)^4 \times \left(\frac{9}{16}\right)^2 \times 3 \times \frac{1}{2} \times p^5 = \frac{3^9}{2^{17}}p^5 \qquad (7.11)$$

Given that P(s_i forms a halting set) has been calculated, 1 - P(s_i forms a halting set) communicates the probability that s_i does not form a halting set. Using the formula derived previously, the probability of at least one halting set being contained within the grid is given by 1 -[P(single grid erasure s_i does not form a halting set)]^{13}, it can be calculated that:

$$\text{P(at least one halting set in GLUV)} = 1 - \left(1 - \frac{3^9}{2^{17}}p^5\right)^{13} \qquad (7.12)$$

The value of the probability given in 7.12 is dependent on the erasure probability p, through simulation it has already been seen that a code based on monoalphabetic GLUV fails to recover sufficient amounts of information to be considered desirable for erasure correction. The rate of the scheme for 15×15 GLUVs over 4 symbols is one third, and consequently the code needs to recover values reliably at erasure probabilities approaching 66%. Substituting a value of $p = 0.3$ into Equation 7.12 gives P(at least one halting set in GLUV at $p = 0.3$) $= 4.7335 \times 10^{-3}$ indicating that even a lower bound would expect a 5-erasure halting set to occur in at least 0.5% of grids. Increasing the probability p to closer to the values at which it needs to be optimal, $p = 0.6$, returns P(at least one halting set in GLUV at $p = 0.6$)$=$ 0.14161. At $p = 0.6$, a lower bound estimate on the prevalence of 5-erasure halting sets indicates the 14% of GLUVs contain at least one halting set, post-transmission. The prevalence of 5-erasure halting sets indicates that monoalphabetic GLUV is not suitable for erasure correction purposes.

Chapter 8

Conclusion

The aim of this research was to investigate the possibility of developing erasure-correction codes that incorporate Sudoku or related combinatorial structures. The research was initially motivated by Soedarmadji and McEliece, whose 2007 paper [32] introduced a new class of erasure-correcting codes based on Latin squares and Sudoku. The paper introduced an iterative decoding algorithm for recovering partially erased Sudoku grids post-transmission although little detail was provided on the encoding or transmission processes; these aspects of a Sudoku-based erasure correction scheme were investigated in Chapters 3 and 4 of the thesis.

The investigation into encoding messages into a Sudoku structure and into transmitting the redundant data demonstrated the practical infeasibility of such an erasure correction scheme, although theoretically a Sudoku based coding scheme can be constructed. The shortcomings include a poor rate of 0.21579, indicating that Sudoku codes must perform at erasure probabilities approaching $1 - 0.21579 = 0.78421$ to be considered practically feasible and able to compete with established, near-optimal codes such as Fountain codes. A simulation of the encoding, transmission, decoding and recovery processes carried out on the Sudoku structure showed that recovering grids at such high probabilities was not possible. In addition to the poor rate, the Sudoku structure often contains small Latin sub-squares (and sub-rectangles), the most common of which are 2×2 intercalates, whose removal from the grid results in an unrecoverable partially-completed grid. The prevalence of these sub-structures contributes to the low recovery rates. Finally, Sudoku is a permutation-based structure and is therefore considerably limiting in the possible placement of values. This property also contributes to the low rate of the scheme, leading to the conclusion that a puzzle structure related to permutations will not be effective.

The conclusions drawn about the practical infeasibility of Sudoku-derived erasure correcting codes prompted the development of GLUV, a three component puzzle-type structure whose properties would be more desirable for erasure correction purposes. Chapters 5, 6 and 7 provide details on the investigation of GLUV. In summary, the favoured structure for an erasure correcting scheme was an order 15 monoalphabetic GLUV over 4 symbols, where all components utilised a single 4-ary alphabet. This chosen structure has a rate of one third, and hence needed to recover at erasure probabilities approaching two thirds to be considered competitive with established codes. A simulation was carried out using a solver developed for recovering GLUV. The results of the simulation were not as encouraging as were initially anticipated, as although GLUV was deliberately developed to avoid the permutation and Latin sub-structure issues experienced with Sudoku, the structure introduced new problematic sub-structures.

Two additional simulations were carried out on structures related to GLUV, with the removal of one of the meta-data components in each case. The original GLUV solver, developed and described in Chapter 6, was altered to facilitate the simulations. The simulation for an order 15 GUV (a grid with up-down vectors alone) over 4 symbols yielded no recovered grids even at the lowest erasure probabilities, $p = 0.05$. However the GLV simulation, a grid with length vectors alone, with the same alphabet size and dimensions, was able to recover partially erased grids post-transmission to a much greater extent. The higher rate of GLV, a result of the removal of the up-down vector meta-information, is such that an erasure correcting scheme incorporating the structure would be required to recover erased grids effectively at a reduced erasure probability in comparison to the original, monoalphabetic GLUV structure. Chapter 6 details graphical results for the outcome of the three simulations related to the GLUV structure, and it can be seen that although monoalphabetic GLUV has a lower rate, the combination of both sets of meta-data result in more effective recovery of erased grids, and hence monoalphabetic GLUV is the most favourable structure.

Although the developed solver is not exact in its recovery of GLUVs with erasures (some additional solution strategies could also be added), investigation into a small number of grids that were not recovered by the solver highlighted the existence of sub-structures, Latin and otherwise, within the grid. These sub-structures were named halting sets and their prevalence, and the extent to which they hinder the recovery of monoalphabetic GLUVs was investigated in Chapter 7. Once classified, lower bounds were calculated for the probability of a specific class of halting set being contained within a GLUV. It was found that at probabilities approaching the Shannon capacity for the scheme (at $p = 0.6$), the lower bound on the proportion of GLUVs that contain at least one 5-erasure halting set is 14%. The prevalence of this case of halting set is such that the scheme cannot be considered practically feasible, as recovery would be hindered significantly by this type of halting set.

Examining each of the schemes presented in this thesis, it appears that puzzle-type combinatorial structures are not effective tools for erasure correction. The main difficulties with the structures considered are a result of the strong cell inter-relationships, born from the use of permutations in the case of Sudoku and from count vectors in the case of GLUV, each indicating information about the potential placement of values within the respective grids. The strong inter-relationships and dependencies between grid cells (and meta-data) result in sub-structures that are difficult to identify or anticipate, and it is ultimately the existence of sub-structures in both

GLUV and Sudoku that are predominant problems.

In addition to the sub-structure problem, the use of puzzle structures requires a far more complex decoding method than is usually required with more traditional erasure correction techniques. A puzzle structure requires a solver to be developed that is capable of efficient recovery of all information erased during transmission. Developing a solver that is able to employ a large variety of solution strategies in a time that is competitive with traditional schemes creates an additional challenge, and further establishes that the use of puzzle structures is undesirable for erasure correction.

Based on the results outlined in this thesis, the direction of future work in terms of combinatorial structures being employed for erasure correction is limited. Of the puzzles outlined in Chapter 2, the result of this research is such that it is difficult to identify any structure that is suitable for erasure correction – all share unsuitable characteristics as the majority of structures are permutation-based. Although the puzzles outlined may not be particularly useful for erasure correction, it is possible that a small number of the additional constraints applied to Sudoku variants in Chapter 2 may be able to provide some aid to erasure correction, but not in conjunction with a permutation-based puzzle structure. Colouring, in particular, could provide advantages for erasure correction, where distinctions between specific cells could be made by colouring cells according to a set of properties. For example, all cells containing odd values could be coloured and all cells containing even values left unchanged (as in Even-Odd Sudoku), allowing a distinction to be made when sub-structures involving both odd and even values occur. Alternatively the grid could be coloured in a variety of regions where all cells coloured a certain way share a particular property or sum to a specific total, much like Pento-Sudoku or Hyper-Sudoku. The problem with such constraints is that when applied alone, to a grid into which any input symbol may be encoded into any cell, the constraint is not likely to facilitate erasure recovery to the extent required. However, if applied in addition to other constraints or meta-data the rate may be compromised to such an extent that the high erasure probabilities at which the derived scheme must perform are unattainable. Additionally, the thesis has identified that the strong cell inter-relationships typical of number placement puzzles and puzzle-type constraints require a variety of solution strategies to be employed to recover erasures, creating difficulties with decoding. These strong cell inter-relationships, and structured nature of such puzzles, also result in the small sets of data that are independent of the rest of the puzzle and the existence of such sub-structures can result in small unrecoverable sets of data. Low density parity-check codes (LDPCs), a more traditional method of erasure correction, could also be said to have strong inter-relationships between symbols, a result of the parity checks employed by the codes. Aside from simple decoding algorithms, one obvious advantage LDPCs have over puzzle-based erasure correction is that LDPCs can be deliberately constructed to avoid small sets of independent data, these sets are known as small cycles in LDPCs. By comparison of puzzle-based erasure correction and traditional erasure correction, it appears the ability to avoid sub-structures/small cycles is crucial to the construction of a practical code – the potentially most significant issue with puzzle-based erasure correction is hence the inability to avoid sub-structures due to the strong cell inter-relationships and structured nature of the puzzles.

However, the investigation of the combinatorial structures in their own right is very much an open subject. Understanding structures such as Sudoku and GLUV as independent structures would allow properties to be identified that may in turn identify more promising applications of the structures.

Bibliography

[1] M. K. Aguilera, R. Janakiraman, and L. Xu. Reliable and Secure Distributed Storage using Erasure Codes. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.2434&rep=rep1&type=pdf`, last accessed: 18/07/12.

[2] R. A. Bailey, P. J. Cameron, and R. Connelly. Sudoku, Gerechte designs, Resolutions, Affine Space, Speads, Reguli and Hamming Codes. American Mathematical Monthly, 115 (5):383–404, 2008.

[3] P. J. Cameron. Sudoku - An Alternative History, February 2007. Available at: `http://www.maths.qmul.ac.uk/~pjc/slides/beamer/sudoku31.pdf`, last accessed: 18/07/12.

[4] R. P. Davies. An Investigation into the Solution to, and Evaluation of Kakuro Puzzles. MPhil thesis, Faculty of Advanced Technology, University of Glamorgan, 2010.

[5] L. Euler. De Quadratis Magicis. Commentationes Arithmeticae, 2:593–602, 1848.

[6] B. Felgenhauer and F. Jarvis. Enumerating Possible Sudoku Grids. 2005. Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.4489&rep=rep1&type=pdf`, last accessed: 18/07/12.

[7] T. Forbes. Quasi-Magic Sudoku puzzles. M500, 215:1–10, April 2007. The Open University, Milton Keynes.

[8] S. Gupta. The Mathematics of Sudoku, October 2005. Available at: `http://theory.tifr.res.in/~sgupta/sudoku/index.html`, last accessed: 18/07/12.

[9] B. Hayes. Sudoku dans la Belle Apoque. Available at: `http://bit-player.org/2006/sudoku-dans-la-belle-epoque`, last accessed: 18/07/12.

[10] S. Huczynska. Powerline Communication and the 36 Officers Problem. Philosophical Transactions of the Royal Society A, 364:3199–3214, October 2006.

[11] S. Hurley, D. H. Smith, and S. U. Thiel. FASoft: A System for Discrete Channel Frequency Assignment. Radio Science, 32(5):1921–1939, 1997.

[12] G. A. Jones and J. M. Jones. Information and Coding Theory. Springer, 2000.

[13] S. K. Jones. The Use of Combinatorial and Heuristic-Search Techniques to Solve Sudoku-Type Puzzles. Technical report, University of Glamorgan, April 2008. UG-M-08-2.

[14] S. K. Jones, P. A. Roach, and S. Perkins. Construction of Heuristics for a Search-Based Approach to Solving Sudoku. Research and Development in Intelligent Systems XXIV: Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Artificial Intelligence, (Springer-Verlag, Bramer, M., Coenen, F. and Petridis, M. Eds.), pages 37–49, 2007.

[15] H. Kwok. Quadruple-Clue Sudoku, 2008. Available at: `http://nrich.maths.org/5797`, last accessed: 18/07/12.

[16] C. F. Laywine and G. L. Mullen. Discrete Mathematics using Latin Squares. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1998.

[17] M. Luby. LT Codes. Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Available at: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1181950`, last accessed: 18/07/12.

[18] G. F. Luger. Artificial Intelligence: Structures and Strategies for Complex Problem Solving (Fifth Ed). Addison Wesley, 2005.

[19] MAA Online. Maths Games - Sudoku Variations. Available at: `http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html`, last accessed: 18/07/12.

[20] D. J. C. MacKay. Fountain Codes. IEE Proceedings-Communication, 152(6), December 2005.

[21] A. J. McAuley. Reliable Broadband Communication Using a Burst Erasure Correcting Code. ACM SIGCOMM, 1990.

[22] G. McGuire, B. Tugemann, and G. Civario. There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem. Available at `http://arxiv.org/pdf/1201.0749v1.pdf`, last accessed: 18/07/12.

[23] B. D. McKay and I. M. Wanless. Most Latin Squares Have Many Subsquares. Journal of Combinatorial Theory (A), 86 (2):323–347, May 1999.

[24] B. D. McKay and I. M. Wanless. On the Number of Latin Squares. Annals of Combinatorics, 9:335–344, 2005.

[25] S. McKendrick and A. Roux. Hex Sudoku 2, 2009. Available at: `http://www.xword.co.za`, last accessed: 21/9/09.

[26] J. J. Metzner. Burst Erasure Correction To Improve The Efficiency Of Broadband CSMA/CD. The 2002 45th Midwest Symposium on Circuits and Systems (MWSCAS-2002), 2:318–321, 2002.

[27] Multiple contributors. Easy Sudoku Puzzle. Available at: `http://www.websudoku.com`, last accessed: 18/07/12.

[28] Nikoli Puzzle Group. Order 8 Hitori Puzzle, 2009. Available at: `http://www.nikoli.com`, last accessed: 18/07/12.

[29] E. Rich and K. Knight. Artificial Intelligence. McGraw-Hill, 2 edition, 1991. Singapore.

[30] L. Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. ACM Computer Communications Review, Vol. 27:24–36, 1997.

[31] A. Shokrollahi. Raptor Codes. IEEE Transactions on Information Theory, Vol. 52:2551–2567, 2006.

[32] E. Soedarmadji and R. J. McEliece. Iterative decoding for sudoku and latin square codes. Forty-Fifth Annual Allerton Conference, pages 488–494, 2007. Available at: `http://leecenter.caltech.edu/workshop08/papers/mceliece2.pdf`, last accessed: 18/07/12.

[33] Sudoku Space. Even/Odd Sudoku (easy), 2009. Available at: `http://www.sudoku-space.com`, last accessed: 18/07/12.

[34] Sudoku Space. Hyper Sudoku (easy), 2009. Available at: `http://www.sudoku-space.com`, last accessed: 18/07/12.

[35] Sudoku Tips. History of Sudoku - The Roots and Development of Sudoku. Available at: `http://www.sudoku-tips.com`, last accessed: 18/07/12.

[36] G. Tarry. Le Probleme de 36 Officiers. Compte Rendu de l'Assoc. Francais Avanc. Sci. Naturel, 2:170–203, 1901.

[37] J. van der Lubbe. Information Theory. Cambridge University Press, 1997.

[38] U. Wieldemann. Logi-5 (pento-sudoku), 2006. Available at: `http://www.sachsentext.de`, last accessed: 18/07/12.

[39] U. Wieldemann. Outside Sudoku, 2006. Available at: `http://www.sachsentext.de`, last accessed: 18/07/12.

[40] U. Wieldemann. Pips Rokudoku, 2006. Available at: http://www.sachsentext.de, last accessed: 18/07/12.

[41] U. Wieldemann. Rokudoku, 2007. Available at: http://www.sachsentext.de, last accessed: 18/07/12.