Concatenated Permutation Codes to Correct Substitution, Deletion or Transposition Errors

by

Reolyn Heymann

A dissertation submitted for the partial fulfillment of the requirements for the degree

Doctor Ingeneriae

in

Electrical and Electronic Engineering Science

at the

University of Johannesburg

Study Leader: Prof. H C Ferreira

Co-Study Leader: Prof. T G Swart

Summary

Permutations of integers can be used in error control coding because of their favourable symbol diversity properties. However, they are currently not widely used. Possible future applications of permutation codes include power line communications (PLC), where every symbol of the codeword is mapped to a specific frequency, and flash memories, where ranked modulation is applied. The focus of this thesis will be to correct different types of errors that occur in PLC and flash memories. Combinations of errors will also be considered.

Firstly, self-synchronizing permutation codes are presented. Usually a code can either detect that a deletion error has occurred, or correct a deletion error assuming that the boundaries of the codeword are known. A construction is presented for permutation codes that is self-synchronizing and able to correct a number of deletion errors per codeword. Although the focus is on detecting and correcting deletion errors, the effect of substitution errors on the decoding algorithm is also considered. The decoding algorithm is adapted to detect substitution errors and, for certain error patterns, even correct the substitution errors.

Then, a new class of permutation codes is presented where, instead of considering one permutation as a codeword, codewords consist of a sequence of permutations. We names this new class concatenated permutation codes and it is a subset of multipermutation codes. The advantage of using permutations, i.e. their favourable symbol diversity properties, is preserved. Importantly, using such sequences of permutations as codewords, code rates close to the upper bound can be achieved.

Different constructions for concatenated permutation codewords are presented to correct different types of errors. Firstly, a construction capable of correcting substitution errors is presented. The construction is modified to be

able to correct either substitution errors or deletion errors. Even though the focus is on deletion errors, it is shown that the decoding algorithm can easily be adapted to also correct insertion errors. The self-synchronizing codes discussed earlier can also be concatenated to achieve higher code rates. The construction is then adapted to correct adjacent transposition errors. This construction is furthermore adapted to correct a combination of transposition or substitution errors.

Opsomming

Permutasies van heelgetalle kan gebruik word in foutkodering na aanleiding van hulle gunstige simbooldiversiteiteienskappe. Nogtans word hulle tans nie algemeen gebruik nie. Moontlike toekomstige toepassings van permutasiekodes sluit in kraglynkommunikasie, waar elke simbool op 'n frekwensietoon afgebeeld word, en vastetoestandgeheue, waar rangmodulasie toegepas word, in. Die fokus van hierdie proefskrif is om verskillende tipes foute op te los wat voorkom in kraglynkommuniasie en vastetoestandgeheue. Kombinasies van foute word ook in ag geneem.

Eerstens word kodes wat self kan sinkroniseer aangebied. Gewoonlik kan 'n kode òf opspoor dat 'n weglatingsfout voorgekom het, òf 'n weglatingsfout kan reggemaak word as die grense van die kodewoord bekend is. 'n Konstruksie word aangebied wat self-sinkroniserend is en in staat is om 'n aantal weglatingsfoute per kodewoord reg te maak. Alhoewel die fokus is om weglatingsfoute op te tel en reg te maak, word die effek van substitusiefoute op die dekoderingsalgoritme ook ondersoek. Die dekoderingsalgoritme word dan aangepas om ook substitusiefoute op te spoor en, vir sekere foutpatrone, selfs die foute te korrigeer.

'n Nuwe klas van permutasiekodes word dan aangebied waar, in plaas daarvan om een permutasie as 'n kodewoord te beskou, kodewoorde uit 'n sekwensie van permutasies bestaan. Hierdie klas word saamgevoegde permutasies genoem en is 'n deelversameling van multipermutasiekodes. Die voordeel om permutasies te gebruik, d.w.s. hulle gunstige simbooldiversiteiteienskappe, word behou. Belangrik, as sekwensies van permutasies gebruik word as kodewoorde, kan kodetempos bereik word wat naby aan die bogrens is.

Verskillende konstruksies vir saamgestelde permutasies word aangebied vir verskillende tipes foute. Eerstens word 'n kode aangebied wat net substitusiefoute kan korrigeer. Die konstruksie word dan aangepas om of substi-

tusiefoute of weglatingsfoute te korrigeer. Selfs al is die fokus op weglatingsfoute, kan die dekoderingsalgoritme maklik aangepas word om ook invoegingsfoute te korrigeer. Die self-sinkroniserende kodes wat vroeër bespreek is kan ook saamgestel word om hoër kodetempos te bereik. Die konstruksie word dan aangepas om naasliggende transposisiefoute te korrigeer. Die konstruksie is ook in staat om 'n kombinasie van substitusie- en transposisiefoute reg te maak.

To my father, Johan Knoetze,

who supported me throughout this journey

but will never get to see the completed product.

Acknowledgements

A sincere word of thanks to ...

- my study leader, Prof. Hendrik Ferreira. Thank you for all your inspiration, guidance, support and mentorship. Thank you for all the opportunities that you create to help us grow,

- my co-study leader, Prof. Theo Swart, for all your input and reviews, sometimes at very short notice,

- Prof. Jos Weber, for collaborating on the topics in this thesis and always being available to discuss ideas and to review papers. Thank you also for being a good friend,

- Prof. Han Vinck, for his guidance and support and giving me the opportunity to visit his research group in Germany,

- my husband, Carl, and beautiful daughter, Lyneska, for all their love and support,

- my parents, Johan and Lynette Knoetze, for always believing in me, and

- all my colleagues at the University of Johannesburg's Electrical and Electronic Engineering Science Department and the Telecommunications Research Group.

Author's Publications

1. R. Heymann and H. C. Ferreira, "Regaining synchronization using neural networks to detect watermark codes," in *Proceedings of the IEEE International Symposium on Information Theory and Its Applications,* Auckland, New Zealand, Dec. 2008, pp. 1–6.

2. R. Heymann and H. C. Ferreira, "Using tree structures to resynchronize permutation codes," in *Proceedings of the IEEE International Symposium on Power Line Communications and its Applications,* Rio de Janeiro, Brazil, Mar. 2010 pp. 108–113.

3. R. Heymann, T. G. Swart and H. C. Ferreira, "Correcting adjacent errors using permutation code trees," in *Proceedings of the IEEE Africon,* Livingston, Zambia, Sep. 2011 pp. 1–5.

4. R. Heymann, T. G. Swart and H. C. Ferreira, "Correcting inversion and synchronization errors using binary neural networks," in *Proceedings of the Pattern Recognition Association of South Africa,* Vanderbijlpark, South Africa, Nov. 2011 pp. 70–73.

5. R. Heymann, H. C. Ferreira and T. G. Swart, "Combined Permutation Codes for Synchronization," in *Proceedings of the IEEE International Symposium on Information Theory and Its Applications,* Hawaii, USA, Oct. 2012 pp. 230–234. (Chapter 3)

For papers that have a direct contribution in this thesis, we have indicated the chapter in which they contribute, in brackets, at the end of each article.

6. R. Heymann, J. H. Weber, T. G. Swart and H. C. Ferreira, "Concatenated permutation block codes based on set partitioning for substitution and deletion error-control," in *Proceedings of the IEEE Information Theory Workshop,* Sevilla, Spain, Sept. 2013 pp. 1–5. (Chapter 4 and Chapter 5)

7. R. Heymann, J. H. Weber, T. G. Swart and H. C. Ferreira, "Concatenated permutation block codes for correcting single transposition errors," in *Proceedings of the IEEE Information Theory Workshop,* Hobart, Australia, Nov. 2014 pp. 576–580. (Chapter 6)

Other Publications

1. M. D. Wilson, H. C. Ferreira, R. Heymann and A. Emleh, "Bit error recording and modelling of in-vehicle power line communication," in *Proceedings of the IEEE International Symposium on Power Line Communications and its Applications,* Glasgow, Scotland, Mar. 2014 pp. 58–63.

2. M. D. Wilson, H. C. Ferreira and R. Heymann, "Markov modelling of in-vehicle power line communication," in *Proceedings of the IEEE Africon,* Mauritius, Mauritius, Sep. 2013, pp. 1–6.

3. P. A. Kamjom, A. R. Ndjiongue, R. Heymann and H. C. Ferreira, "Framework to establish a simplex communication system over a power line using load signatures," in *Proceedings of the IEEE International Symposium on Power Line Communications and its Applications,* Johannesburg, South Africa, Mar. 2013 pp. 179–184.

4. A. Goedhart, R. Heymann and H. C. Ferreira, "Adapting HomePlug C&C PLC for use in a low voltage smart grid," in *Proceedings of the IEEE International Symposium on Power Line Communications and its Applications,* Beijing, China, Mar. 2012 pp. 194–199.

Contents

List of Figures

List of Tables

List of Symbols

χ	A one-to-one mapping from the set $\mathcal{B}_w$ to $\mathcal{T}$
ϕ	A one-to-one mapping from the set $\mathcal{B}_v$ to $\mathcal{R}$
$\psi(r)$	The function to map a permutation sequence r to the specific subset $\mathcal{R}_i$ it belongs to.
$\sigma(j)$	The transformation of the decimal representation of j to the binary representation of j.
$\xi(t)$	The function to map a permutation sequence t to the specific subset $\mathcal{T}_i$ it belongs to.
B_v	All binary sequences of length v.
$d_{\min}$	The minimum Hamming distance of a codebook.
K	The number of data permutation subwords in a concatenated codeword.
k	The number of bits or subwords in a sequence before encoding.
L	Number of subsets which will be used during encoding and decoding.
M	The number of symbols in a permutation.
n	The length of a codeword after encoding.
P_e	The probability of a substitution error occurring.
R	Code rate in bits/symbol.
S_M	Set of all possible permutations of length M.
v	The length of the binary sequence which is mapped to a permutation.

Chapter 1

Introduction

1.1 Introduction

Information theory, specifically applied to digital communication, plays a crucial role in our everyday lives. It is estimated that 40% of the world's population has internet access, according to the International Telecommunication Union (the specialized agency for information and communication technologies of the United Nations) [1]. The number of mobile cellular subscriptions reached almost 7 billion by the end of 2014. People using mobile-broadband continues to grow at double digit rates. However, the information obtained through digital communication is only valuable if it is reliable, thus the information needs to be transmitted or stored error-free. Several factors can cause errors to occur in transmitted or stored data. It is thus important that these errors are identified and, if possible, corrected.

Most digital communication is still binary, thus strings of bits (ones and zeroes) are used to represent data. However, the throughput can be increased by increasing the number of symbols used. Instead of using 2 symbols, M different symbols may be used to represent data. Different encoding and decoding schemes need to be constructed to ensure reliable communication.

In this thesis, communication with M symbols, or M-ary communication is considered, with the added restriction that every symbol M must occur an exact number of times per codeword. This is known as permutation codes. Permutation codes are currently mostly considered for application in power line communications (PLC) and flash memory. Each of these two applications implements permutation codes differently and also experience different types of errors.

1.2 Problem Statement

The main problem investigated in this thesis is how to correct different types of errors when applying permutation codes in power line communications and flash memories. Traditionally, a subset of permutations is chosen from the complete set of possible permutations to form a codebook. These permutations are chosen in such a way that the codebook has certain desirable distance properties. Binary data is then mapped to these permutations. However, when only small subsets of all possible permutations are used, the cardinalities of these codebooks are small resulting in low code rates.

A new class of codes is presented in this thesis where permutations are concatenated to form codewords, instead of a single permutation being a codeword. Different concatenation techniques are investigated to correct different types of errors. Combinations of different types of errors are also considered. The resulting code rates, error correction capabilities and distance properties are analyzed.

1.3 Outline of Thesis

In Chapter 2 we present an overview of previous work related to the work in this thesis. This is not a complete overview of all literature related to permutations, but rather a representative subset that is related to the work in this thesis. A mathematical overview of permutations is given. Then the implementation of permutation codes in the two main applications, PLC and flash memories, is described. The different types of errors which will be focused on in this thesis is presented as well as different distance measures suitable for different types of errors.

Traditionally a code can either detect that a synchronization error occurs, for example with markers, or correct a synchronization error assuming that the boundaries of the codeword is known, for example Levenshtein codes. A construction is presented in Chapter 3 of permutation codes that are self-synchronizing and able to correct a number of deletion errors per codeword. The effect of substitution errors on the construction is taken into account and the decoding algorithm is adapted to also detect substitution errors and, for some error patterns, correct substitution errors. These codes are not concatenated but will be used again in Chapter 5.

Then, a new class of permutation codes is presented where, instead of considering one permutation as a codeword, codewords consist of a sequence of permutations. These codes are a subclass of multipermutation codes. The advantage of using permutations, i.e. their favourable symbol diversity properties, is preserved. Additionally, using sequences of permutations as codewords, code rates close to the optimum rate can be achieved.

Firstly, the complete set of permutations is divided into subsets by using set partitioning, as shown in Chapter 4. Binary data is then mapped to permutations from these subsets. These permutations, together with a parity permutation, will form the codeword. A construction capable of detecting and correcting substitution errors is presented in Chapter 4 and is adapted to be capable of detecting and correcting either substitution or deletion errors in Chapter 5. The constructions are firstly presented for the simple $M = 4$ case and then generalized for all values of M. The decoding algorithm can also be adapted to correct insertion errors. Thus, the decoding algorithm can correct either substitution or deletion or insertion errors. The self-synchronization codes from Chapter 3 are also concatenated in Chapter 5 to achieve higher code rates.

In Chapter 6, it is shown that an outer code can be used to add more parity permutations in order to also be able to correct transposition errors. A simple Hamming code can be applied as outer code. The codes presented in this chapter can correct a single adjacent transposition error per codeword. The decoding algorithm is also adapted to detect and correct a combination of transposition or substitution errors. Different outer codes can be used to increase the error-correction capability of the codes.

A summary of each chapter is given at the end of Chapters 3 to 6. Finally the work is concluded in Chapter 7. An overview is given of the work presented with specific results highlighted and suggestions for possible future extensions are discussed.

Chapter 2

Literature Study

2.1 Introduction

An overview of selected, related literature is given in this chapter. This is not a complete overview of all research relating to permutation codes and their application in power line communication or flash memories.

Firstly, a mathematical overview is given of permutations and their relevant properties. Then an introduction is given to the two main applications of permutation codes which will be considered in this thesis, namely power line communications (PLC) and flash memories. Different distance measures can be used when constructing codebooks. The most relevant distance measures are discussed in detail and only a brief overview is given of some of the others. The applications are susceptible to different kinds of errors, which will be discussed in the next section. Multipermutations are introduced and then the chapter is concluded with a brief section on set partitioning.

2.2 Permutations and Relevant Mathematical Concepts

Let $\mathcal{M} = \{1, 2, \ldots, M\}$ be the set of M integers and let S_M denote the set of all $M!$ permutations. The identity permutation, e, is the ordered permutation

$(12 \dots M)$ [2]. Two-line notation can be used to represent a permutation, for example:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 5 & 4 & 6 & 2 & 1 & 7 \end{pmatrix}. \tag{2.1}$$

In this example, 1 is replaced or renamed as 3, 2 as 5, etc. A cyclic notation can also be used. From the above example, $1 \to 3 \to 4 \to 6 \to 1$, thus (1346) forms a cycle. Similarly, (25) is a cycle. While 7 stays the same and is referred to as a singleton cycle. The identity permutation consists of only singleton cycles. A cycle of length 2 is also known as a transposition since the two symbols are swapped. The permutation in (2.1) can thus also be written in cyclic notation as (1346)(25)(7). Throughout this thesis, one-line notation will be used, thus (2.1) will simply be written as (3546217).

Every permutation has an inverse. To get the inverse of a permutation, the two lines of the two-line notation are swapped and the columns are ordered again according to the symbols which are now in the top line. The inverse of (2.1) is

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 5 & 1 & 3 & 2 & 4 & 7 \end{pmatrix}. \tag{2.2}$$

Let $\mathbf{x} = x_1, x_2, \dots, x_M$ and $\mathbf{y} = y_1, y_2, \dots, y_M$ be two permutations. The two permutations can be multiplied by applying them one after the other to the symbol set. Multiplication of permutations are not commutative and we will be using the left-to-right order when doing multiplication. Thus, if $\mathbf{z} = \mathbf{xy}$, then for $j = 1, 2, \dots, M$, $z_j = y(x(j))$.

The Hamming distance, d, between two codewords is defined as the number of positions in which the codewords differ. The minimum Hamming distance of a codebook, $d_{\min}$, is defined as the minimum Hamming distance between any two different codewords in the codebook. Hamming distance, and research focusing on using Hamming distance to correct errors in permutation codes, will be discussed in more detail in Section 2.4.1.

Let $\mathbf{x} = x_1, x_2, \dots, x_M$ be a permutation. If $i < j$ and $x_i > x_j$, then the pair (x_i, x_j) is called an inversion. The identity permutation is the only permutation that does not contain any inversions. Let $\mathbf{b} = b_1, b_2, \dots, b_M$ be

the inversion table of $\mathbf{x}$, where b_j is the number of elements to the left of $x_k = j$ that are greater than x_k. Every permutation has a unique inversion table, thus the original permutation can be constructed from the inversion table as long as the the following holds for $\mathbf{b}$:

$$\begin{aligned} b_M &= 0, \\ 0 \leq b_{M-1} &\leq 1, \\ &\vdots \\ 0 \leq b_2 &\leq M-2, \\ 0 \leq b_1 &\leq M-1. \end{aligned} \qquad (2.3)$$

Example: Let $\mathbf{x} = (32145)$. The number of inversions is 3 and the inversion pairs are $(3,1)$, $(3,2)$ and $(2,1)$. The inversion table is 21000.

If the total number of inversions in a permutation is even, then the permutation is said to be even. Similarly, if the total number of inversions is odd, the permutation is said to be odd. The set of all even permutations is denoted as $\mathcal{A}_M$, where $|\mathcal{A}_M| = M!/2$ and the minimum Hamming distance, $d_{\min} = 3$ [3].

Definition 2.1. *A permutation code C of length M is a subset of S_M, where every codeword in C contains M different integers $\{1, 2, \ldots, M\}$ as symbols.*

The cardinality of a codebook, $\mathcal{C}$, is denoted by $|\mathcal{C}|$.

A permutation can be transformed into another permutation by transposing (or swapping) symbols. For example, the permutation (1423) can be transformed into permutation (3421) by the function $swap(1, 3)$.

2.3 Applications

The main two applications for permutation codes which will be considered in this thesis is power line communications (PLC) and flash memory. Even though both applications make use of permutation codes, the implementation is different as well as the types of errors which occur. This section will only deal with how permutation codes are implemented. The different types of errors will be discussed in the next section.

In PLC, permutation codes are combined with M-ary Frequency Shift Keying (M-FSK) modulation. Every symbol is mapped to a specific frequency. In [4] it is shown that this combination is able to combat different types of noise present in PLC, especially for narrowband PLC in the CENELEC A band. Applications include automatic meter reading and demand side management.

Flash memory consist of floating gate cells which have a discreet number of levels. Every level represents a symbol. It is much more time consuming erasing cells than writing to cells. It is thus important not to overshoot when charging a cell to a certain level. Using permutation codes with rank modulation eliminates overshoot errors.

In [5], it was proposed to use permutation codes with rank modulation to eliminate overshoot errors. Instead of charging a specific cell to a predetermined, discreet level, cells are charged relative to each other. For example, lets assume that we have the permutation (51243). Position 1 of the permutation contains the number of the cell with the most charge. Position 2 contains the number of the cell with the second most charge, and so forth. This is illustrated in Figure 2.1.

Figure 2.1: Illustration of rank modulation for permutation 51243

The disadvantage of using rank modulated codes is that a large number of comparisons are needed to determine the codeword. Using a sliding window approach is suggested in [6], where the window size is small and thus less comparisons are needed. This is called local rank modulation. The use of constant-weight Gray codes with local rank modulation is proposed in [7], where Gray codes are defined as codes where the distance between adjacent symbols are a minimum. More detailed constructions are presented in [8], focusing on characteristics like code length, sliding window size, and so forth. Specific schemes, with specific parameters, are studied in [9].

2.4 Relevant Distances

An overview of the distance measures used with permutations relevant to this thesis is discussed in this section, with the focus on Hamming distance. A complete survey on distances used with permutations is given in [10].

2.4.1 Hamming Distance

Hamming distance and the minimum Hamming distance of a codebook have already been defined in Section 2.3. An upper bound on the cardinality, $|\mathcal{C}|$, of a permutation codebook with minimum distance $d_{\min}$ is given by [3]:

$$|\mathcal{C}| \leq \frac{M!}{(d_{\min}-1)!}. \tag{2.4}$$

The minimum Hamming distance of the complete set of permutations, S_M, is 2. This is trivial since permutations are generated by swapping symbols. As mentioned earlier, it is possible to divide the set S_M equally into two sets containing even and odd permutations, each with a minimum distance of 3. However, it is not possible to reach all these upper bounds. For example, for $M = 6$ and $d = 5$, it was proven in [11] that the maximum cardinality that can be achieved is only 18, and not 30, as given in the above equation. In [12], a new upper bound was found for permutations of length 10 and a minimum distance 9. In [13], a new upper bound was found for permutations of length 7 and a minimum distance 5.

Traditionally, a codebook consists of a subset of $\mathcal{S}_M$ with specific distance properties. Since binary codebooks with good distance properties already exist, the process of distance-preserving mappings from binary sequences to permutations have been proposed in [14]. The mapping is done in such a way that the permutation codebook has a minimum Hamming distance equal to, or greater than, that of the binary codebook. In this specific paper, binary convolutional codes are mapped to permutation codes, resulting in a permutation trellis code able to correct errors which occur frequently in PLC. The performance of different permutation trellis codes are simulated in [15]. Improved permutations trellis codes which were found by exhaustive computer search are presented in [16].

Several constructions of distance-preserving mappings from binary codewords to permutations are given in [17]. Constructions are also given for distance-increasing mappings, where the minimum Hamming distance of the permutation codes are larger than that of the binary codes. Constructions just focusing on distance-increasing mappings are presented in [18]. In [19], a construction is presented for distance-preserving mappings where the binary codewords have an odd length and is mapped to permutations of the same length. A multilevel construction is given in [20] for distance-preserving mappings. A general upper bound is also derived using a sum of Hamming distances between the binary codebook and the permutation codebook. This can be used to evaluate mappings.

Distance-preserving mappings from ternary codewords to permutations are given in [21]. The motivation is that large ternary codes of length n and minimum distance d, are usually larger than binary codes with the same characteristics. Thus, a construction for distance-preserving mappings from ternary codewords with $n > 13$ to permutations of the same length is given. Another construction is presented in [22] which gives an improved lower bound for $d \leq \frac{3n}{d}$. An algorithm for an input length of $8m$ is given for any integer where $m > 2$. The algorithm consists of two passes that result in a number of transitions. The algorithm is then extended for all input lengths of at least 16.

In [23], it is shown that binary codes can be generated from permutation codes. Each permutation codeword is equivalent to a binary codeword. This is determined by calculating the weight of the permutation codeword related to the inversion table and forming an equivalent binary codeword. The generated binary codewords are usually nonlinear.

A prefix method is used in [24] to construct permutations of length M from permutations of length $M - 1$. The distance properties of the $M - 1$ codebook are preserved. The method is based on the method proposed in [25]. For example, let us assume that the codebook consists of 16 codewords of length 4. Firstly, the symbol 5 is added to the start of every codeword in the codebook, resulting in 16 new codewords of length 5. To form the next 16 codewords, the symbol 5 is added to the start of each codeword and then swapped with symbol 4, thus $swap(4, 5)$. Next, symbol 5 is added and swapped with symbol 3, and so forth. A codebook consisting of codewords of length 5 is thus created with the same minimum Hamming distance as the original $M = 4$ codebook.

Constructions for permutations arrays using the Hamming distance are presented in [26]. The concept of balanced permutation codes are presented. A codebook, $\mathcal{C}$, is said to be balanced if, for each position, every possible symbol occurs the same number of times in that position in the codewords of $\mathcal{C}$. A method is introduced where two permutation arrays are combined to form a larger permutation array. More constructions, which are balanced under certain conditions, are given in [27].

A correspondence between mutually orthogonal latin squares (MOLS) and permutations is presented in [28], and thus certain MOLS can be used to construct permutations with specific distance properties. More research on MOLS constructions are given in [29]. More constructions for permutation codes based on Hamming codes for PLC are given in [30].

2.4.2 Kendall τ Distance

An adjacent transposition is when the positions of two symbols are swapped. For example, let the transmitted sequence be $\mathbf{x} = x_1 x_2 \dots x_i x_j \dots x_M$. Thus, if symbol i and j are swapped, then $\mathbf{x}' = x_1 x_2 \dots x_j x_i \dots x_M$. If $j = i+1$, then an adjacent transposition occurred. The Kendall τ distance, $d_\tau(\mathbf{x}, \mathbf{y})$, is the number of pairwise, adjacent transpositions needed to turn the permutation $\mathbf{x}$ into the permutation $\mathbf{y}$ [31]. It is defined as:

$$d_\tau(\mathbf{x}, \mathbf{y}) = \sum_{\{i,j\} \in \mathcal{P}} K_{i,j}(\mathbf{x}, \mathbf{y}) \tag{2.5}$$

where

- $\mathcal{P}$ is the set of unordered, distinct symbols in $\mathbf{x}$ and $\mathbf{y}$,
- $K_{i,j}(\mathbf{x}, \mathbf{y}) = 0$ if i and j are in the same order in $\mathbf{x}$ and $\mathbf{y}$,
- $K_{i,j}(\mathbf{x}, \mathbf{y}) = 1$ if i and j are not in the same order in $\mathbf{x}$ and $\mathbf{y}$.

If $d_\tau(\mathbf{x}, \mathbf{y}) = 0$, then $\mathbf{x}$ and $\mathbf{y}$ are identical. The upper bound of d_τ is $\frac{M(M-1)}{2}$, this is obtained if all the symbols in $\mathbf{x}$ are reversed in $\mathbf{y}$.

Example: If $\mathbf{x} = (1234)$ and $\mathbf{y} = (4321)$ then $d_\tau(\mathbf{x}, \mathbf{y}) = 6$ because the following pairs are out of order: (12), (13), (14), (23), (24) and (34). If $\mathbf{x} = (1427365)$ and $\mathbf{y} = (2147536)$ then $d_\tau(\mathbf{x}, \mathbf{y}) = 4$ because the following pairs are out of order: (12), (24), (35) and (56).

In [32] and [33], bounds are derived for permutation codes with the Kendall τ distance.

The Kendall τ distance is also related to the Cayley distance. The Cayley distance is defined as the number of transpositions needed to change one codeword into another. The transpositions do not need to be adjacent.

2.4.3 Other Distances Related to Permutation Codes

In [34] and [35], the Ulam distance is used as distance measure. A translocation is defined as moving the symbol in position i to position j and shifting all the elements between position i and j. For example, let the transmitted sequence be $\mathbf{x} = x_1 x_2 \dots x_i x_j \dots x_M$. If $i < j$, then a right-translocation is defined and $\mathbf{x}' = x_1 x_2 \dots x_{i-1} x_{i+1} \dots, x_j x_i x_{j+1} \dots x_M$. If $i > j$, then a left-translocation is defined and $\mathbf{x}' = x_1 x_2 \dots x_{j-1} x_i x_j x_{j+1} \dots, x_{i-1} x_{i+1} \dots x_M$. The Ulam distance is defined as the number of translocations needed to change one codeword into another. In other words, the Ulam distance is the length of the permutations minus the longest common subsequence.

The Chebyshev distance, L_∞ (also sometimes referred to as the L_1 max-distance in literature), is defined on a vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension. Permutations based on the Chebyshev distance were studied in [36], [37] and [38].

In [39], single error-correcting codes based on the Lee metric is presented. The Lee distance is defined as the shortest path between two codewords in the circular Cayley graph. Using a permutation adjacency graph, and showing that it is a subgraph of a multi-dimensional linear array, error-correcting codes based on the Lee metric are constructed.

Generalized Kendall τ and Cayley distances are defined in [40]. Instead of swapping symbols, intervals are defined and swapped. Breakpoints are then defined to construct larger codebooks.

In [41], a cost is assigned to transpositions. One would thus want to find transpositions with the smallest cost to transform one permutation into another. This is called similarity distances and is also applied in voting theory.

2.5 Types of Errors

Different types of errors occur in PLC and flash memories. Impulse noise, background noise and permanent frequency disturbances can occur in the PLC channel. In flash memories errors occur mainly due to cell discharge which leads to transposition or translocation errors. The main types of errors which will be considered in this thesis is discussed in this section.

2.5.1 Transposition Errors

Rank modulation is explained in Section 2.3. Charge leakage do occur over time. It is possible that the cells do not discharge at uniform rates, thus that one cell discharges quicker than another. For example, let us consider the permutation illustrated in Figure 2.1. If cell 1 discharges to such a degree that its charge level drops below that of cell 2, the resulting permutation will be (52143). An adjacent transposition error has thus occurred where the symbols in position 2 and 3 were swapped. The cells with their new charge levels are illustrated in Figure 2.2.

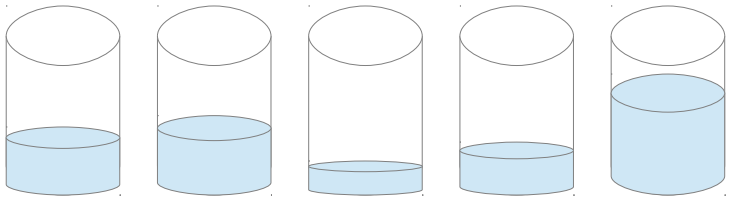

Figure 2.2: Illustration of transposition errors: 51243 $\rightarrow$ 52143

To correct transposition errors, the Kendall τ metric is usually used since it is defined as the number of adjacent transpositions needed to transform one

codeword into another. Since good error-correcting codes already exist for binary codes based on Hamming distance, a method to modify these codes to correct errors based on the Kendall τ distance is presented in [42]. Explicit examples are given for Reed-Solomon and BCH codes.

In [43], the use of systematic permutation codes are proposed. Systematic codes are traditionally defined for binary codes. A systematic (n, k) code is a code of length n, consisting of two parts [44]. The first part contains the k information bits, thus every possible k sequence appear once in the codebook. The second part contains the $n - k$ parity bits. The information bits which are encoded are thus embedded in the codeword. In an (n, k) systematic permutation code, every possible k permutation appear once in the codebook containing codewords of length n. These k permutations are also called the information symbols. Systematic permutation codes are constructed using the Kendall τ and the Chebyshev distance in [43]. In [45], constructions for systematic permutation codes using the Kendall τ metric is presented. This is also generalized for multipermutations (refer to Section 2.6).

In [46], a metric embedding technique is used to translate M-ary codewords based on the Lee distance, to permutation codes, based on the Kendall τ distance. Relevant bounds are also derived.

2.5.2 Synchronization Errors

Establishing and maintaining synchronization is important in communication systems. A loss of synchronization, even if it is just for a short period, can lead to bursts of errors that may be catastrophic. Synchronization errors can be modelled as deletion(s) and/or insertion(s) of bits or symbols.

Definition 2.2. *A deletion error is defined as an error that causes one symbol to be deleted and all subsequent symbols to move forward in time and is modelled as all subsequent symbols in the sequence moving one index to the left.*

Definition 2.3. *An insertion error is defined as an error that causes a symbol to be inserted into a sequence and all subsequent symbols to be delayed in time and is modelled as all subsequent symbols in the sequence moving one index to the right.*

For example, let the transmitted sequence be $\mathbf{x} = x_1x_2 \dots x_n$. If a deletion error occurs in position i, then the received sequence will be $\mathbf{y} = y_1y_2 \dots y_{i-1}y_{i+1} \dots y_{n-1}$, where $y_j = x_j$ for all $j < i$ and $y_j = x_{j+1}$ otherwise. If the symbol z was inserted in position i, then the received sequence will be $\mathbf{y} = y_1y_2 \dots y_{i-1}zy_iy_{i+1} \dots y_{n+1}$, where $y_j = x_j$ for all $j < i$ and $y_j = x_{j+1}$ otherwise.

The simplest solution to ensure that a sequence can be resynchronized is by using markers (also known as commas). A marker is a known sequence that is periodically placed in the transmitted sequence. No other codeword in the codebook should include this sequence. At the receiver the positions of the markers are used to determine if synchronization has been lost and to resynchronize. Markers for binary codes have been studied in [47] and [48]. In [49] a construction for markers used with permutation codes is given.

A disadvantage of markers is that they add additional redundancy to the sequence. Markers can detect the loss of synchronization and help with recovery, but cannot correct the specific deletion or insertion errors. The use of watermarks have been proposed as an alternative to markers [50], [51].

Another approach is to construct comma-free codebooks [52]. In a comma-free codebook, the codewords are constructed in such a way that the overlap between two codewords does not result in a valid codeword.

The definition of whether a codebook is synchronizable is given for a permutation codebook C:

Definition 2.4. *$X = x_1x_2 \dots x_M$ and $Y = y_1y_2 \dots y_M$ are two, not necessarily different, permutation codewords in our permutation codebook C. The codebook C is synchronizable if the sequence*

$$x_{j+1}x_{j+2} \dots x_My_1y_2 \dots y_j \tag{2.6}$$

is not a valid codeword. If j can be any value $j = 1, 2, \dots M-1$, then the code is called comma-free.

In [52] the upper bound on the cardinality of a comma-free codebook of words with length n, constructed from an alphabet containing the letters $0, 1, \dots, \sigma - 1$ is given as:

$$f(\sigma, n) \leq \frac{1}{n} \sum_{d|n} \mu(d)\sigma^{n/d} \tag{2.7}$$

where the summation is extended over all divisors d of n, and $\mu(d)$ is the Möbius function. A good approximation of this upper bound is given by $f(\sigma, n) \leq \sigma^n/n$ [53].

In [52] an upper bound on the cardinality of comma-free codes is also given. A construction for maximal comma-free codes is given in [54]. In [55] a suffix construction method is proposed to construct variable length codes with synchronization capability. Comma-free codebooks can also be used to regain synchronization but cannot correct information bits lost due to synchronization errors.

Prefix codes, a subclass of comma-free codes, have also been suggested to regain synchronization [56], [53]. Every codeword in a codebook starts with a predefined prefix. The prefix does not appear anywhere else in the codeword and can thus be used to determine the start of every codeword.

Many synchronization error correcting techniques have also been investigated. These schemes assume that the boundaries of the codewords are determined by using markers. In [57] constructions for codebooks consisting of M-ary codewords able to correct single deletions are given. A construction to correct two or more deletions using design theory is given in [58].

The problem of synchronization errors when using permutation codes in conjunction with M-FSK modulation has been investigated in [59]. An error-correcting scheme was proposed that could correct a single insertion or deletion error. This scheme was improved on in [60]. A method to correct synchronization errors using tree structures was proposed in [61] and [62].

In [63], the hardware implementation of rank modulated permutation codes are considered and a distinction is made between a stable and an unstable deletion error. A deletion as defined in Definition 2.2 is known as a stable deletion. However, an unstable deletion does not only affect the position of the other symbols but also the values of the other symbols. For example: refer back to Section 2.3, if rank modulation is used and one symbol is deleted, then the ranking of all the other symbols would change too. Let the stored permutation be (51243). If the symbol in the third position, 2, was deleted, then the resulting permutation would be 4132. For the rest of the thesis, whenever deletion errors are considered, it will be stable deletions.

The problem of synchronization errors when using permutation codes in conjunction with rank modulation in flash memory has been investigated in [63] and [64]. In this work stable and unstable deletion errors are investigated. Deletions in multipermutations has been investigated in [65]. (See Section 2.6 for definition of multipermutations).

2.5.3 Substitution Errors

A substitution error occurs if one symbol is transmitted and another is received. When using binary codes, this is also known as inversion, additive or reversal errors. The Hamming distance is usually used in combination with substitution errors. When using permutation codes, single substitution errors can easily be detected since one symbol will occur twice in the codeword and one symbol not at all. The error-correction code thus have to identify which of the symbols occurring twice, is in error.

In PLC, permutation codes are combined with M-FSK. At the transmitter, the M symbols are mapped to M frequencies and then transmitted in time. Usually, the receiver has a detector for each frequency and the output of the demodulator is the frequency that corresponds to the detector with the highest output, as explained in [66]. In the presence of noise, the detector with the highest output might not represent the signal that was transmitted, leading to a substitution error.

However, in [4], a modified receiver is proposed. The demodulator still has a detector for each frequency but a decision is not made with regards to which detector has the highest output. For example, if the codeword (1234) was transmitted without any errors, then the output of the demodulator will be

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

where the columns represent the time t and every row represents a frequency. Thus, at time instance t_1, frequency f_1 was detected.

PLC is vulnerable to three types of errors as described in [4]: impulse noise, background noise and permanent frequency disturbances. Impulse noise causes a signal to be present in all the detectors for a limited time period. Returning to the previous example, if impulse noise occurs in timeslot t_2, then the output will be

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

Permanent frequency disturbances, or narrow-band interference, affect a narrow band for long periods of time, thus one frequency is affected and will be presented for the entire duration of the demodulation process. For example, if frequency f_3 is affected, then the demodulator output will be

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.10}$$

Background noise is equivalent to Gaussian white noise, thus a transmitted signal may not be present at the receiver or a signal may be received which was never transmitted. For example, if the frequency f_1 which was transmitted in timeslot t_1 and the frequency f_2 in timeslot t_3 are affected, then the demodulator output will be

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.11}$$

In [67], a permutation decoding process is proposed based on soft demodulator output, enabling the decoder to perform better.

As mentioned earlier, distance-preserving mappings are used in [14], [16], [15] and [68], in order to be able to make use of trellis codes to correct impulse noise, background noise and permanent frequency disturbances. Reed-Solomon codes are used in [69] and [70]. Turbo codes are used in [71].

Some research focused only on permanent frequency disturbances. In [70] the concept of same-symbol weight is defined and used to minimize the probability of a narrow-band interference error being undetected. The concept of symbol equity is defined in [72] to determine a code's ability to recover from permanent frequency disturbances and resulted in the construction of equitable symbol weight codes. In [73], these codes are optimized.

It has also been proposed to use orthogonal frequency division multiplexing (OFDM) modulation instead of M-FSK. It is even included in the HomePlug

standard [74] for in-home smart grid power line communications. Some research focused mainly on impulsive noise. The performance degradation of OFDM modulation for coherent and differential M-ary phase shift keying is studied in [75]. In [76] and [77], low-density parity-check (LDPC) codes combined with OFDM modulation is considered. Performance degradation due to impulsive noise is limited by using an estimator and a signal level limiter. An iterative algorithm to correct impulsive noise errors when only using a small number of OFDM subcarriers is presented in [78]. Only selected subcarriers are used in [79] when using a QPSK-OFDM transmission scheme. A method of combining bit-interleaved coded modulation with iterative decoding and OFDM is presented in [80]. A bit-interleaved convolutionally coded OFDM is presented in [81]. Periodic impulse noise is considered in [82]. An algorithm to detect periodic impulse noise is implemented, which is then suppressed by an adaptive infinite impulse response filter.

OFDM modulation combined with fountain codes are presented in [83] with the focus on impulse noise and permanent frequency disturbances. A decoding procedure is presented that firstly cancels the permanent frequency disturbances and then the impulse noise, where after it decodes using maximum likelihood decoding.

M-ary differential phase shift keying is used in combination with permutation codes in [84]. Reed-Solomon codes are combined with cyclic permutation codes to combat the errors. Both simulation and implementation results are presented.

Linear programming decodable permutation codes are presented in [85]. Linearly constrained permutation matrices are defined which are used to construct the linear programming decodable permutation codes. Constructions, an error analysis and bounds are given for the permutation codes.

2.5.4 Other Types of Errors

Inter-cell coupling is when the charge level in a specific cell is affected by its neighboring cells. If a cell contains a large charge level, it can cause its neighboring cells' charge levels to also increase. In [86], constrained programming is proposed where certain adjacent cell levels are not allowed. However, rank modulation is not taken into consideration. Inter-cell coupling in rank modulation schemes are considered in [87]. The single-neighbor k-constraint is

introduced, where the charge levels between two cells may differ with at most k. A construction is given in [87] and bounds are derived. In [88], the single-neighbor k-constraint is extended to the two-neighbor k-constraint where a cell may differ with at most k from either the previous symbol or the next symbol. Constraining symbols decreases the error probability, however, errors do still occur. Error-correction codes based on the Kendall τ distance is combined with the constrained code in [88].

Translocation errors using the Ulam distance are considered in [34]. (Refer to Section 2.4.3 for the definition of a translocation.) A translocation can also be seen as the combination of a deletion and an insertion error, i.e., the symbol is deleted in one position and inserted in another position. Code constructions and bounds are given in [34].

2.6 Multipermutations

In [89], two variants of permutation codes are defined. For the first variant, $\mathbf{x}_1$ is a codeword of length n, then

$$\mathbf{x}_1 = (\overbrace{x_1 \dots x_1}^{r_1} \overbrace{x_2 \dots x_2}^{r_2} \overbrace{x_3 \dots x_3}^{r_3} \dots \overbrace{x_M \dots x_M}^{r_M}), \tag{2.12}$$

where x_1 to x_M are real numbers and it is assumed that

$$x_1 < x_2 < x_3 < \cdots < x_M. \tag{2.13}$$

Furthermore, r_1 to r_M are positive integers with

$$n = r_1 + r_2 + \cdots + r_M. \tag{2.14}$$

All the other codewords are then obtained by permuting $\mathbf{x}_1$. For the second variant, (2.13) is changed to

$$0 \leq x_1 < x_2 < x_3 < \cdots < x_M. \tag{2.15}$$

These code are also known as permutations over a multiset or multipermutations. An r-regular multipermutation is defined as a permutation of $\mathbf{x_1}$ (from (2.12)) where

$$r_1 = r_2 = \cdots = r_M. \tag{2.16}$$

In [35], r-regular multipermutations are proposed for application in rank modulated flash memory. Multipermutation codes enable one to use longer code lengths leading to better code rates. Cells are still charged relative to each other, however, cell i now represents the rank of cell i. Cells with the same rank do not need to have exactly the same charge levels. For example, let the multipermutation code be $\mathbf{x} = (1212)$. Any of the permutations in Figure 2.3 can represent $\mathbf{x}$. All four permutations are equivalent.

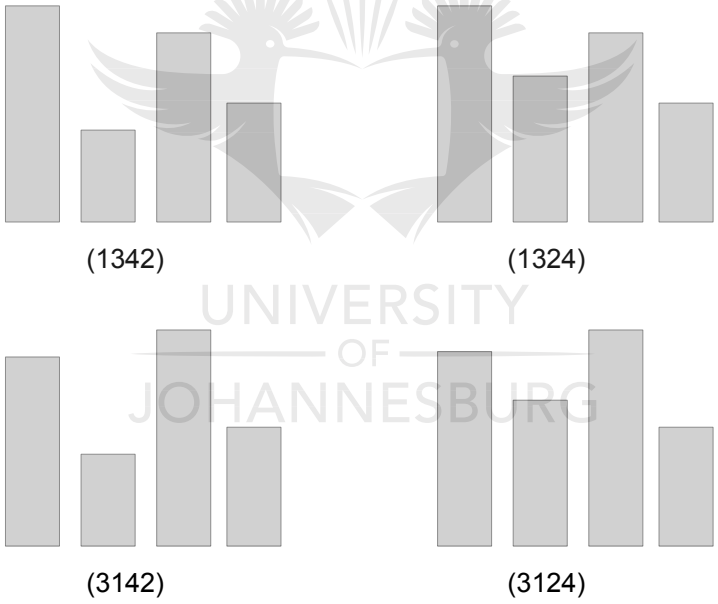

Figure 2.3: Example of equivalent permutations for the multipermutation (1212)

Bounds for rank modulated multipermutation codes using the Ulam distance and Hamming distance are derived in [35]. Code constructions are also presented using permutation interleaving, semi-Latin squares and resolvable Steiner systems. Error-correcting codes using the Kendall τ distance is presented in [90]. Multipermutation matrices are introduces in [91]. Based

on this, LP-decodable multipermutation codes are presented based on the Chebyshev distance.

2.7 Set Partitioning

Set partitioning is the process of dividing a set into smaller subsets in such a way that the subsets are nonempty and every object of the original set is included in exactly one subset. The use of set partitioning in Trellis Coded Modulation was proposed in [92]. A process is used in Trellis Coded Modulation where constellation points are partitioned into subsets, and then those subsets are partitioned into smaller subsets and so forth. Every time set partitioning is applied, the Euclidean distance between signals in the same subset is increased. In this thesis, set partitioning will be applied to permutations to increase the Hamming distance in subsets. An example is given in Section 4.2.

Chapter 3

Resynchronizable Permutation Codes Correcting Deletion Errors

3.1 Introduction

Classes of codes have been proposed that are synchronizable, i.e. they can be used to regain synchronization although the error leading to the loss of synchronization is not corrected. An example of this is when markers are used to detect synchronization errors. Typically, different classes of codes are needed to correct deletion and/or insertion errors after codeword boundaries have been detected. For example, Levenshtein codes assume that the codeword boundaries are known and then synchronization errors can be corrected.

The codebooks presented in this chapter consist of codewords divided into segments. By imposing restrictions on the segments, the codewords are synchronizable. One deletion error can be detected and corrected per segment. If the number of deletions per segment is less than the segment's length, the deletions can also be detected.

Concatenation of codes are not yet introduced in this chapter. However, it is shown in Chapter 5 that the codebooks presented in this chapter can be concatenated to achieve higher cardinalities and thus better code rates.

The constructions presented in this chapter have been published in [93].

3.2 Codebook Construction

If a symbol is deleted from a codeword, then the resulting codeword of length $M-1$ is known as a subword. Every possible such subword for every codeword in C should be unique for it to be able to correct a single deletion error.

In [57], the complete permutation set S_M is divided into M partitions, each consisting of $(M-1)!$ codewords. Each such partition forms a codebook able to correct one deletion per codeword.

The partitions for $M = 4$ from [57] is given in Table 3.1.

Table 3.1: S_4 partitioned into 4 codebooks, each able to correct one deletion error

1234	2134	1324	2341
2143	3124	1423	1243
3142	4123	2413	1342
4132	1432	2314	4312
3241	2431	3412	4213
4231	3421	4321	3214

A Subword Lookup Table will be used later during the decoding process to correct single deletion errors. The lookup table basically consists of every codeword from a specific partition with all its possible subwords. Table 3.1 from [57] is used to construct a new Subword Lookup Table for the first partition.

Table 3.2: Subword Lookup Table for the first partition of Table 3.1

Codewords	Subwords			
1234	234	134	124	123
2143	143	243	213	214
3142	142	342	312	314
4132	132	432	412	413
3241	241	341	321	324
4231	231	431	421	423

The proposed codebook will consist of codewords divided into segments. The different segments will be constructed using the method in [57] to ensure that a deletion can be corrected in every segment. Every sequence in

segment 1 will then be combined with every sequence in segment 2 to form the resynchronizable permutation codebook.

The construction can be explained step by step as follows:

1. Choose the lengths of each segment. Let l_1 denote the length of segment 1 and l_2 the length of segment 2; $l_1 + l_2 = M$.

2. The symbols $1, 2, \ldots, M$ are divided into two sets, $\mathcal{L}_1$ and $\mathcal{L}_2$, where $|\mathcal{L}_1| = l_1$ and $|\mathcal{L}_2| = l_2$. For the rest of the thesis, we will let $\mathcal{L}_1 = \{1, 2, \ldots, l_1\}$ and $\mathcal{L}_2 = \{l_1 + 1, l_1 + 2, \ldots, M\}$.

3. For segment 1: Construct sequences as in [57], consisting of symbols $\{1, 2, \ldots, l_1\}$, capable of correcting one deletion error per sequence. These sequences will form codebook C_1. The number of possible sequences is $(l_1 - 1)!$.

4. For segment 2: Construct sequences, consisting of symbols $\{1, 2, \ldots, l_2\}$, capable of correcting one deletion error per sequence. Add l_1 to every symbol so that the sequences consist of symbols ranging from $\{l_1 + 1, l_1 + 2, \ldots, M\}$. These sequences will form codebook C_2. The number of possible sequences is $(l_2 - 1)!$.

5. Concatenate every codeword from C_1 with every codeword from C_2 to form the codebook C.

Proposition 3.1. *For any codeword in C, a deletion error can be corrected in every segment.*

Proof. The segments are chosen from the deletion correction codebooks given in [57], which can correct one deletion error provided that the boundaries are known. Since each segment consists of only specific symbols, the boundaries can be inferred from this. □

Example: Create a synchronizable codebook with $M = 6$.

1. Let the length of segment 1 be equal to the length of segment 2, i.e. $l_1 = l_2 = 3$.

2. There are $(l_1 - 1)! = 2! = 2$ possible sequences: $\{123, 321\}$.

3. Since segment 1 and 2 have equal lengths, the possible sequences for segment 2 are also $\{123, 321\}$. The sequences are translated to $\{456, 654\}$.

4. The sequences from segment 1 and 2 are combined to form the codebook $\{123456, 321456, 123654, 321654\}$.

As further example where $l_1 \neq l_2$, the codebook for $M = 7$, $l_1 = 3$ and $l_2 = 4$ is given in Table 3.3.

Table 3.3: Example of a comma-free codebook capable of correcting 1 deletion error per segment ($M = 7$)

$M = 7$		
1234567	1235476	1236475
1237465	1236574	1237564
3214567	3215476	3216475
3217465	3216574	3217564

A codebook constructed in this way will be able to resynchronize after a synchronization error, as well as correct one deletion error per segment. The process for resynchronization and error correction is given in the next section.

The cardinality for codebooks constructed in this way is

$$|C| = (l_1 - 1)!(l_2 - 1)!. \tag{3.1}$$

Since binary data is mapped to the codewords, the code rate is given by:

$$R = \frac{\lfloor \log_2[(l_1 - 1)!(l_2 - 1)!] \rfloor}{M} \text{ bits/symbol.} \tag{3.2}$$

The choice of l_1 and l_2 thus influences the cardinality. For example: if $M = 8$ and $l_1 = 4$, then the cardinality would be $|C| = (4-1)!(4-1)! = 36$. However, if $M = 8$ and $l_1 = 3$, then the cardinality increases to $|C| = (3-1)!(5-1)! = 48$. The effect of the segments' lengths on the cardinality and error correction capability of the code is discussed in more detail in Section 3.3.3.

During the encoding process, binary codewords are mapped to the permutation codewords by using a lookup table called the Encoding/Decoding Lookup Table. After the decoder detected and corrected any errors, the Encoding/Decoding Lookup Table is used again to map the permutation codewords back to the binary codewords.

3.3 Detecting and Correcting Deletion Errors

A decoding procedure, capable of detecting and correcting deletion errors, is first presented. The decoding algorithm is then adapted to also be able to deal with substitution errors.

3.3.1 Decoding

Every codeword consists of 2 segments, the first constructed from symbols of $\mathcal{L}_1$ and the second from symbols of $\mathcal{L}_2$. Due to deletion errors, the lengths of these segments will vary at the receiver. To avoid confusion, the decoder will work with segment runs instead of segments. A segment run is defined as any number of consecutive symbols from either $\mathcal{L}_1$ or otherwise from $\mathcal{L}_2$. For example, Let $M = 6$ and $\mathcal{L}_1 = \{1, 2, 3\}$, then the sequence 12456254 consists of 4 segment runs: 12, 456, 2 and 54. In the decoding algorithm, the function $length(\mathbf{x}_k)$ denotes the length of segment run $\mathbf{x}_k$. Thus, from the above example, if $\mathbf{x}_2 = 456$, then $length(\mathbf{x}_2) = 3$.

At the decoder, the received sequence will first be divided into segment runs. Each segment run will be examined separately to determine whether errors occurred. The decoder will thus be able to detect and correct one deletion error or detect up to $l_i - 1$ deletions per segment run, where l_i is the length of the original segment constructed from symbols of $\mathcal{L}_i$. Once the errors in the segment runs have been corrected, then the segment runs are combined again to form codewords and are mapped back to the original binary representation.

Let the received sequence be $\mathbf{y}$, where $\mathbf{y}$ consists of several consecutive codewords.

1. Split the sequence $\mathbf{y}$ into a sequence consisting of different segment runs, $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$.

2. Set $k = 1$, where k is an index that points to the segment under consideration.

3. For segment run $\mathbf{z}_k$, determine which segment it belongs to, $\mathcal{L}_1$ or $\mathcal{L}_2$. Let i be a pointer to either $\mathcal{L}_1$ or $\mathcal{L}_2$. If k is odd, then $i = 1$, else $i = 2$.

4. If $length(\mathbf{z}_k) = l_i$, then set $\mathbf{x}'_k = \mathbf{z}_k$. If k is even, go to Step 7, else let $k = k + 1$ and go to Step 3. If $length(\mathbf{z}_k) \neq l_i$, go to Step 5.

5. If $length(\mathbf{z}_k) = l_i - 1$, then one deletion error occurred and can be corrected. By using the Subsequence Lookup Table, $\mathbf{x}'_k$ is obtained. If k is even, go to Step 7, else let $k = k + 1$ and go to Step 3. If $length(\mathbf{z}_k) \neq l_i - 1$, go to Step 6.

6. Detect that $l_i - length(\mathbf{z}_k)$ deletion errors have occurred. If k is even, declare $\mathbf{x}'_{k-1}$ and $\mathbf{z}_k$ undecodable. Let $k = k + 1$ and continue with Step 3. If k is odd, declare $\mathbf{z}_k$ and $\mathbf{z}_{k+1}$ undecodable. Let $k = k + 2$ and continue with Step 3.

7. Set the decoder output as $\mathbf{u}'_{k/2}$, where $\mathbf{u}'_{k/2}$ is obtained by combining $\mathbf{x}'_{k-1}$ and $\mathbf{x}'_k$ and then using the Encoding/Decoding Lookup Table. If $k \neq n$, then let $k = k + 1$ and go back to Step 3, else STOP.

3.3.2 Example

Let $M = 7$ and $l_1 = 3$. The codebook given in Table 3.3 is used. The encoded sequence is $\mathbf{x} = (1234567, 3217564)$. Four cases are considered for the received sequence $\mathbf{y}$ and the corresponding decoding results are given.

- If $\mathbf{y} = \mathbf{x}$ (no errors), then 4 segment runs are identified, $\mathbf{z} = (123, 4567, 321, 7564)$. Each of the segment runs has the correct length so we go from Step 1, to Step 2, 3, 4, and again 3 and 4 before ending in Step 7.

- If $\mathbf{y} = (1234673217564)$, i.e. a deletion error occurred in the fifth symbol, then in Step 1 we find $\mathbf{z} = (123, 467, 321, 7564)$. For $\mathbf{z}_1 = (123)$, we determine that the symbols belong to $\mathcal{L}_1$. In Step 4, $length(\mathbf{z}_1) = l_1 = 3$, $\mathbf{x}'_1 = 123$ and $k = 1$ so we go back to Step 3. For $\mathbf{z}_2 = (467)$, we determine that the symbols is a subset of $\mathcal{L}_2$. In Step 4, $length(\mathbf{z}_2) \neq l_2$ and we proceed to Step 5 where we find that $length(\mathbf{z}_2) = l_2 - 1 = 3$, thus one deletion error is detected. The subsequence $\mathbf{z}_2$ can only map to one sequence, since all the sequences from $\mathcal{C}_2$ have unique subsequences. The subsequence $\mathbf{z}_2 = (467)$ is thus mapped to (4567) and $\mathbf{x}'_2 = 4567$. In Step 7, $\mathbf{x}'_1 = 123$ and $\mathbf{x}'_2 = 4567$ are combined to form the codeword (123456) and the Encoding/Decoding Lookup Table is used to determine the relevant binary codeword. The next two segment runs are error-free and will be similar to the previous scenario.

- If $\mathbf{y} = (12373217564)$, i.e. deletions occurred in positions 4, 5 and 6, then in Step 1 we find $\mathbf{z} = (123, 7, 321, 7564)$. For $\mathbf{z}_1 = (123)$, we determine that the symbols belong to $\mathcal{L}_1$. In Step 4, $length(\mathbf{z}_1) = l_1 = 3$, $\mathbf{x}'_1 = 123$ and $k = 1$ so we go back to Step 3. For $\mathbf{z}_2 = (7)$, we determine that the symbol is a subset of $\mathcal{L}_2$. In Step 4 we find that $length(\mathbf{z}_2) \neq l_2$ and thus proceed to Step 5 where we find that $length(\mathbf{z}_2) \neq l_2 - 1$. Thus, in Step 6 we determine that $l_2 - length(\mathbf{z}_2) = 4 - 1 = 3$ deletions occurred. Since k is even, $\mathbf{x}'_1$ and $\mathbf{z}_2$ are undecodable. We return to Step 3 and continue decoding $\mathbf{z}_3$.

- If $\mathbf{y} = (345673217564)$, i.e. deletions occurred in positions 1 and 2, then in Step 1 we find $\mathbf{z} = (3, 4567, 321, 7564)$. For $\mathbf{z}_1 = (3)$, we determine that the symbol is a subset of $\mathcal{L}_1$. In Step 4 we find that $length(\mathbf{z}_1) \neq l_1$ and thus proceed to Step 5 where we find that $length(\mathbf{z}_1) \neq l_1 - 1$. Thus, in Step 6 we determine that $l_1 - length(\mathbf{z}_1) = 3 - 1 = 2$ deletions occurred. Since k is odd, $\mathbf{z}_1$ and $\mathbf{z}_2$ are undecodable. We return to Step 3 and continue decoding $\mathbf{z}_3$.

3.3.3 The Effect of Segment Lengths

Figure 3.1 shows the code rates for different values of M and l_1. (Remember that $l_1 + l_2 = M$.) If $l_1 = 1$ or $l_1 = 2$, then that segment will be a prefix since it will only consist of one possible sequence. For the line where $l_1 = l_2$, $l_1 = \lfloor \frac{M}{2} \rfloor$.

The lines for $l_1 = 3$ and $l_1 = l_2$ are identical for low values of M and then diverge for larger values of M. For $M = 6$, $l_1 = l_2 = 3$, resulting in the same code rate initially for the $l_1 = 3$ and the $l_1 = l_2$ line. For $M = 7$, 8 and 9, the output of the term $\lfloor \log_2[(l_1 - 1)!(l_2 - 1)!] \rfloor$ is the same for both the $l_1 = 3$ and the $l_1 = l_2$ line. Only after $M = 9$, different code rates are obtained.

The code can only correct one deletion error per segment. Let P_d be the probability of a deletion error occurring. Using Bernoulli trials, the result of independent errors with probability P_d, defined as, P, is given by

$$P = [1-(1-P_d)^{l_1} - l_1 P_d (1-P_d)^{l_1-1}] + [1-(1-P_d)^{l_2} - l_2 P_d (1-P_d)^{l_2-1}]. \quad (3.3)$$

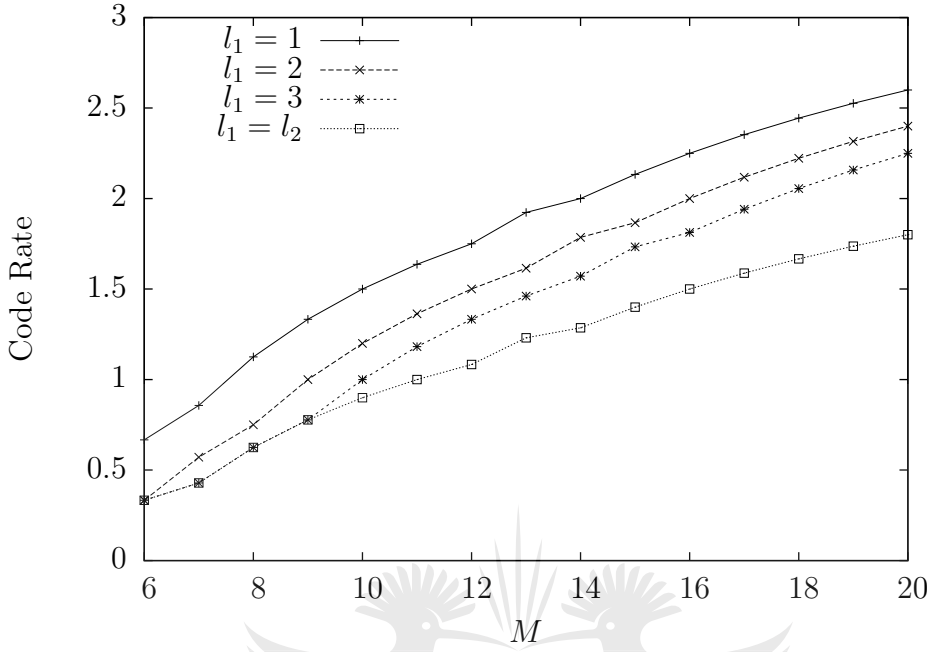

Figure 3.1: Code rates for different values of M and different segment lengths

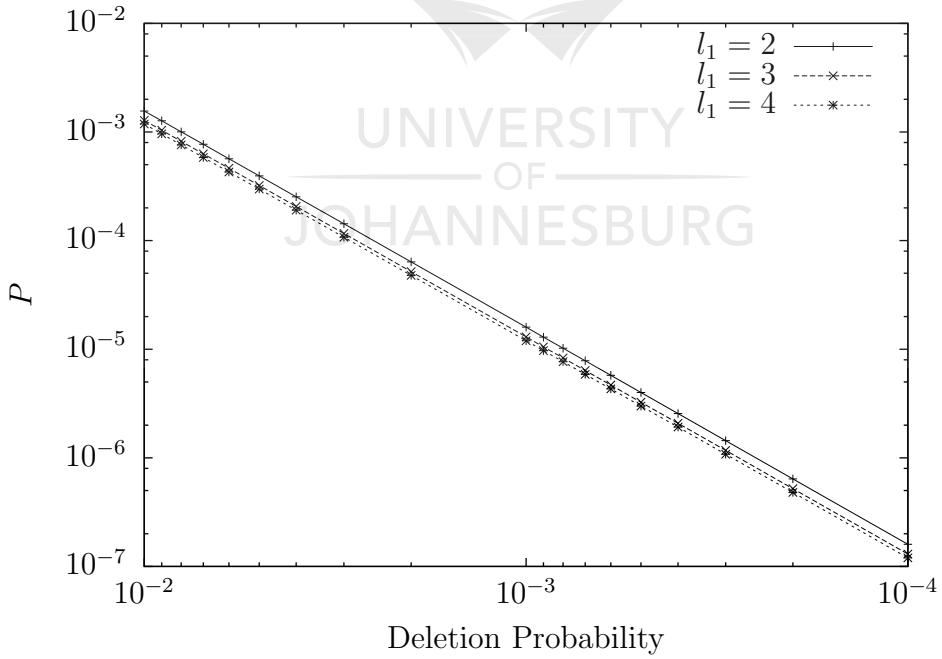

Figure 3.2: Probability that a deletion error will not be corrected: $M = 8$

The probability that an error will not be corrected is shown in Figure 3.2 for $M = 8$ and different values of l_1.

Thus, keeping one segment's length to a minimum increases the code rate considerably. However, the probability of not being able to correct a deletion error also increases. It is thus important to find a balance when choosing the segment lengths. Even though the decoder will not be able to correct more than one error per segment, it will be able to detect up to $l_i - 1$ deletions.

3.4 Detecting and Correcting Deletion and Substitution Errors

Even though the focus of the construction presented in this chapter is on deletion errors, the occurrence of substitution errors is also considered since substitution errors are usually more probable than deletion errors. The occurrence of a substitution error should not lead to infinite error propagation. The decoding algorithm from the previous section is adapted to also detect substitution errors. It is possible for certain error patterns to convert the substitution error to a deletion error and thus to correct it.

Segments still refers to the two parts of a codeword consisting of symbols from either $\mathcal{L}_1$ or otherwise from $\mathcal{L}_2$. To avoid confusion, the decoder identifies segment runs and not segments, which is any number of consecutive symbols from either $\mathcal{L}_1$ or otherwise from $\mathcal{L}_2$. Since substitution errors are also taken into account now, it is possible that a segment run can contain a symbol more than once.

To enable the decoder to detect both deletion and substitution errors, it is assumed that the codewords before and after an erroneous codeword is error-free.

For illustration purposes, let $M = 6$, $l_1 = 3$ and $\mathcal{L}_1 = \{1, 2, 3\}$. A substitution error in a segment run can lead to the following three scenarios:

- A symbol in a segment run from the set $\mathcal{L}_i$ can be replaced with another symbol from the set $\mathcal{L}_i$. This will not affect the length of the segment run but one symbol will occur more than once in the segment run. For example: $(123) \rightarrow (133)$. If the symbol which occurs twice is

adjacent, one of the symbols can be deleted. The substitution error is thus converted to a deletion error and the decoder will be able to correct it. However, if the two identical symbols are not adjacent, it is not possible to know which one is the erroneous symbol and the substitution error can only be detected.

- If the first or last symbol in a segment run is substituted with a symbol from another set, then one segment run's length will be increased by one and another segment run's length will be decreased by one. For example: $(123456) \rightarrow (125456)$. This will lead to the first segment run containing 2 symbols and the next segment run containing 4 symbols. The segment run which length increased will now contain one symbol twice. If one of the double symbols is at the start or end of the segment run and the other one not, then the double symbol at the start or end can be deleted. Again, converting the substitution error into a deletion error which will be corrected.

  Returning to the previous example, if (125456) is received, then two segment runs are identified: 12 and 5456. The second segment run contains the symbol 5 twice and one of the double symbols are at the start of the segment run. Symbol 5 at the start of the segment run is thus deleted, leading to a deletion error in the first segment run. Since the decoding algorithm is capable of correcting deletion errors, the decoder will correct the deletion error in the first segment run. However, an alternative solution is possible: since we know that a substitution error occurred in the first segment run, we know the position of the error was in the last position of the segment run and we know that the segment run is a permutation of the symbols in $\mathcal{L}_1$, we can also just check which symbol is missing and thus replace it.

  If one of the double symbols occur at the start of the segment run and the other at the end of the segment run, then one will have to determine which adjacent segment run has been decreased before a decision can be made about which symbol should be deleted. Since the adjacent segment run lengths are used to determine the position of the error, we assume that adjacent codewords are error-free.

- If a symbol that is not at the start or end of a segment run is substituted with a symbol from another set, then the segment run will be divided into three segment runs. For example: $(123456) \rightarrow (143456)$. Initially the segment runs were $(123, 456)$, but the substitution error changed it

to $(1, 4, 3, 456)$. An error like this will result in one segment run being broken into three, with the middle segment run having a length of one. If this scenario is identified, the middle segment run can be deleted and the deletion error can be corrected by the decoding algorithm.

3.4.1 Decoding

Let the received sequence be $\mathbf{y}$, where $\mathbf{y}$ consists of several consecutive codewords.

1. Split the sequence $\mathbf{y}$ into a sequence consisting of different segment runs, $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$.

2. Set $k = 1$, where k is an index that points to the segment under consideration, and $j = 1$, where j is an index that points to the position where the decoded codeword is placed in the decoded sequence.

3. For segment run $\mathbf{z}_k$, determine which set it belongs to, $\mathcal{L}_1$ or $\mathcal{L}_2$. Let i be a pointer to either $\mathcal{L}_1$ or $\mathcal{L}_2$. If k is odd, then $i = 1$, else $i = 2$.

4. If $length(\mathbf{z}_k) = l_i$ go to Step 5. If $length(\mathbf{z}_k) = l_i - 1$, go to Step 7. If $length(\mathbf{z}_k) = l_i + 1$, go to Step 8, else go to Step 9.

5. Check if one symbol occurs more than once in $\mathbf{z}_k$. If a symbol occurs more than once, go to Step 6. Otherwise, if k is even, set $\mathbf{x}'_2 = \mathbf{z}_k$ and go to Step 10. Else, set $\mathbf{x}'_1 = \mathbf{z}_k$, $k = k + 1$ and proceed to Step 3.

6. If the two symbols with the same value is adjacent, delete one of the symbols and go to Step 7. If not, it is not possible to correct the substitution error and the codeword will be undecodable. If k is even, declare $\mathbf{x}'_1$ and $\mathbf{z}'_k$ undecodable. Set $k = k + 1$ and continue with Step 3. If k is odd, declare $\mathbf{z}'_k$ and $\mathbf{z}'_{k+1}$ undecodable. Set $k = k + 2$ and continue to Step 3.

7. Use the Subsequence Lookup Table to see which codeword, $\mathbf{x}^*$, the subword $\mathbf{z}_k$ belongs to. If k is even, set $\mathbf{x}_2 = \mathbf{x}^*$ and go to Step 10, else set $\mathbf{x}_1 = \mathbf{x}^*$, $k = k + 1$ and go to Step 3.

8. If $length(\mathbf{z}_{k-1}) = l_{(i+1) \bmod 2} - 1$, then delete the first symbol of $\mathbf{z}_k$. Else, delete the last symbol of $\mathbf{z}_k$. If k is even, set $\mathbf{x}'_2 = \mathbf{z}_k$ and go to Step 10. Else, set $\mathbf{x}'_1 = \mathbf{z}_k$, $k = k + 1$ and proceed to Step 3.

9. If $length(\mathbf{z}_{k+1}) = 1$ and $length(\mathbf{z}_k) + length(\mathbf{z}_{k+2}) = l_i - 1$, then delete $\mathbf{z}_{k+1}$ and add $\mathbf{z}_{k+2}$ at the end of $\mathbf{z}_k$.

   Use the Subsequence Lookup Table to see which codeword, $\mathbf{x}^*$, the subword $\mathbf{z}_k$ belongs to. If k is even, set $\mathbf{x}_2 = \mathbf{x}^*$, $k = k + 2$ and go to Step 10, else set $\mathbf{x}_1 = \mathbf{x}^*$, $k = k + 3$ and go to Step 3.

   Else, detect that $l_i - length(\mathbf{z}_k)$ deletion errors have occurred. If k is even, declare $\mathbf{x}'_1$ and $\mathbf{z}'_k$ undecodable. Set $k = k + 1$ and continue with Step 3. If k is odd, declare $\mathbf{z}'_k$ and $\mathbf{z}'_{k+1}$ undecodable. Set $k = k + 2$ and continue to Step 3.

10. Set the decoder output as $\mathbf{u}'_j$, where $\mathbf{u}'_j$ is obtained by combining $\mathbf{x}'_1$ and $\mathbf{x}'_2$ and then using the Encoding/Decoding Lookup Table. If $j \neq \frac{n}{2}$, then let $k = k + 1$, $j = j + 1$ and go back to Step 3, else STOP.

Referring back to the scenarios mentioned earlier: the possibility that the first scenario occurred is investigated in Step 5 and 6 of the decoding algorithm. The second scenario is investigated in Step 8 and the last scenario in Step 9.

As mentioned earlier, converting the substitution error into a deletion error is not the only possible solution. It is also possible in some of the scenarios to declare an erasure rather than deleting the symbol affected by the substitution error. The permutation property that every symbol occurs only once can then be used to correct the error. The decoding algorithm given above rather changes the substitution errors to deletion errors since the focus of the algorithm is on deletion errors. Declaring an erasure will also not work in the first error scenario discussed above, since it is not clear which one of the double symbols is in error. However, since they are adjacent, it does not matter which one is deleted.

3.4.2 Example

Let $M = 7$ and $l_1 = 3$. The codebook given in Table 3.3 is used. Four cases are considered for the received sequence $\mathbf{y}$ and the corresponding decoding results are given.

- The encoded sequence is $\mathbf{x} = (1234567)$. If $\mathbf{y} = (1234667)$, i.e. a substitution error occurred in the fifth symbol, then in Step 1 we find $\mathbf{z} = (123, 4667)$. In Step 3 we determine that we are working with $i = 1$ and in Step 4 that $length(\mathbf{z}_1) = l_i = 3$, so we proceed to Step 5. Every symbol occurs only once, so we set $\mathbf{x}'_1 = 123$ and return to Step 3. For $\mathbf{z}_2$ and $i = 2$, we proceed to Step 4 where we find that $length(\mathbf{z}_2) = l_2 = 4$, so we proceed to Step 5. However, in Step 5, we find that the symbol 6 occurs twice and in Step 6 that it is adjacent. One of the symbols is deleted and the deletion error is corrected in Step 7. Since k is even, we proceed to Step 10, where the relevant lookup table is used to map the permutation codeword to the binary codeword.

- The encoded sequence is $\mathbf{x} = (1234567)$. If $\mathbf{y} = (1234547)$, i.e. a substitution error occurred in the sixth symbol, then in Step 1 we find $\mathbf{z} = (123, 4547)$. In Step 3 we determine that we are working with $i = 1$ and in Step 4 that $length(\mathbf{z}_1) = l_i = 3$, so we proceed to Step 5. Every symbol occurs only once, so we set $\mathbf{x}'_1 = 123$ and return to Step 3. For $\mathbf{z}_2$ and $i = 2$, we proceed to Step 4 where we find that $length(\mathbf{z}_2) = l_2 = 4$, so we proceed to Step 5. However, in Step 5, we find that the symbol 4 occurs twice and in Step 6 that it is not adjacent. It is thus not possible to determine which one of the double symbols is the incorrect one. Since k is even, $\mathbf{x}'_1$ and $\mathbf{z}_2$ are declared undecodable.

- The encoded sequence is $\mathbf{x} = (1234567, 1234567)$. If $\mathbf{y} = (1234567523\ 4567)$, i.e. a substitution error occurred in the eighth symbol, then in Step 1 we find $\mathbf{z} = (123, 45675, 23, 4567)$. In Step 3, we are working with $\mathbf{z}_1 = (123)$, $i = 1$ and in Step 4 that $length(\mathbf{z}_1) = l_i = 3$, so we proceed to Step 5. Every symbol occurs only once, so we set $\mathbf{x}'_1 = 123$ and return to Step 3. For $\mathbf{z}_2 = (45675)$ and $i = 2$, we proceed to Step 4 where we find that $length(\mathbf{z}_2) = l_2 + 1 = 5$, so we proceed to Step 8. Looking back, $\mathbf{z}_1 \neq l_1 - 1$, thus the last symbol of $\mathbf{z}_2$ is deleted and $\mathbf{x}'_2 = (4567)$. Since k is even, we proceed to Step 10 where the relevant lookup table is used to determine the binary representation of (1234567).

  Continuing again to Step 3, the next segment run is $\mathbf{z}_3 = 23$ and $i = 1$. The substitution error that occurred in the eight position was thus deleted. The decoding algorithm will correct the resulting deletion error like in the previous section.

- The encoded sequence is $\mathbf{x} = (1234567)$. If $\mathbf{y} = (1234517)$, i.e. a substitution error occurred in the sixth symbol, then in Step 1 we find $\mathbf{z} = (123, 45, 1, 7)$. In Step 3 we determine that we are working with $i = 1$ and in Step 4 that $length(\mathbf{z}_1) = l_i = 3$, so we proceed to Step 5. Every symbol occurs only once, so we set $\mathbf{x}'_1 = 123$ and return to Step 3.

  For $\mathbf{z}_2 = (45)$ and $i = 2$, we proceed to Step 4 where we find that $length(\mathbf{z}_2) = 2$, which does not match any of the options given, so we proceed to Step 9. In Step 9, we find that $length(\mathbf{z}_3) = 1$ and that $length(\mathbf{z}_2) + length(\mathbf{z}_4) = 3 = l_2 - 1$, thus $\mathbf{z}_3$ is deleted and $\mathbf{z}_4$ is appended to $\mathbf{z}_2$. The Subsequence Lookup Table is used to determine the codeword $\mathbf{x}^* = (4567)$. Since k is even, $\mathbf{x}'_2 = \mathbf{x}'$. We proceed to Step 10 where $\mathbf{x}'_1$ and $\mathbf{x}'_2$ are combined and the the Encoding/Decoding Lookup Table is used to determine the relevant binary codeword.

3.5 Conclusion

A construction is presented that can detect and correct deletion errors. Codewords are divided into segments and certain symbols are restricted to certain segments. The codewords are constructed in such a way that one deletion error can be corrected per codeword. The lengths of the segments can be specified, keeping one segment small and the other large leads to higher code rates but also increases the probability that a deletion error will not be corrected. Even though not all deletion errors are corrected, up to $l_i - 1$ deletion errors can be detected.

Since the probability of substitution errors is usually higher than that of deletion errors, the effect of substitution errors on the construction is also considered. The decoding algorithm is adapted to also detect substitution errors. For some error patterns, the decoding algorithm is able to detect which symbol was effected by the substitution error and delete that symbol. The resulting deletion error can then be corrected. It is assumed that the codeword before and after an erroneous codeword is error-free, if the decoder detects a combination of both substitution and deletion errors.

The construction will be revisited in Section 5.5. By concatenating the codewords, higher code rates can be achieved.

Chapter 4

Concatenated Codes to Correct Substitution Errors

4.1 Introduction

A new class of permutation codes is presented where, instead of considering one permutation as a codeword, codewords consist of a sequence of permutations. The advantage of using permutations, i.e. their favourable symbol diversity properties, is preserved. Additionally, using sequences of permutations as codewords, code rates close to the optimum rate can be achieved, where code rates are defined as the number of binary bits presented by each permutation. (The term mapping ratio can also be used instead of code rate.) Firstly, the complete set of permutations is divided into subsets by using set partitioning. Binary data is then mapped to permutations from these subsets. These permutations, together with a parity permutation, will form the codeword, as shown in Figure 4.1. Thus, error control capabilities are achieved with very little loss in rate and simple decoding. The number of permutations in every codeword is an important parameter: a larger number will increase the code rate but also the probability of undetected errors.

Only substitution errors are considered in this chapter. The construction, which is referred to as Construction I, is adapted in the following chapters to correct other types of errors. The construction is firstly presented for $M = 4$ and then generalized for all values of $M > 4$. Only the Hamming distance will be considered, thus all reference to distance is the Hamming distance.

The work presented in this chapter has been published in [94].

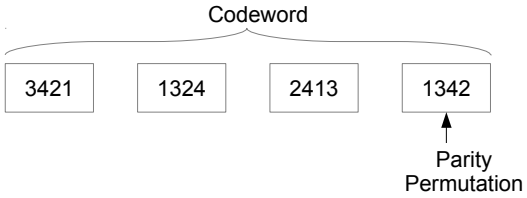

Figure 4.1: Concatenation of permutations to form a codeword ($M = 4$)

4.2 Set Partitioning

The set $\mathcal{S}_4$ will be divided into subsets using the concept of set partitioning as explained in Chapter 2. The set $\mathcal{S}_4$ has a minimum distance of 2. The set is firstly partitioned into 2 subsets, one containing even permutations and one containing odd permutations, each with a minimum distance of 3 [3]. These two subsets are further partitioned into smaller subsets with distance 4. Thus, the 24 permutations of length 4 are divided into 6 subsets:

$$\mathcal{R}_0 = \{1234, 2143, 3412, 4321\},$$

$$\mathcal{R}_1 = \{1243, 2134, 4312, 3421\},$$

$$\mathcal{R}_2 = \{1324, 3142, 2413, 4231\},$$

$$\mathcal{R}_3 = \{1342, 3124, 4213, 2431\},$$

$$\mathcal{R}_4 = \{1423, 4132, 2314, 3241\},$$

$$\mathcal{R}_5 = \{1432, 4123, 3214, 2341\}.$$

Hence, the set $\mathcal{S}_4$ has been divided in such a way that two sequences within the same subset will have a distance of 4, while two sequences from different subsets will have a minimum distance of 2.

The sets presented in this section will also form the basis for all the subsets in Section 4.4 where $M > 4$.

4.3 Code Construction for $M = 4$

Not all the subsets from the previous section will be used. Binary sequences are mapped to the permutation sequences. The number of bits that will be mapped to a permutation is $\lfloor \log_2 4! \rfloor = 4$. The number of subsets that will be used is thus $2^4/4 = 4$. (A generalized equation is given in the next section.) Let $\mathcal{B}_v$ be the set consisting of all binary sequences of length v and let K be a positive integer.

For all permutation sequences $\mathbf{r} \in \mathcal{R}_i$, define

$$\psi(\mathbf{r}) = i. \tag{4.1}$$

Let

$$\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3. \tag{4.2}$$

Let ϕ be a one-to-one mapping from the set $\mathcal{B}_4$ to $\mathcal{R}$.

4.3.1 Encoding

A source generates a sequence $\mathbf{u}$ of $4K$ bits, which is partitioned into K sequences $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$, all from $\mathcal{B}_4$. Let $\mathbf{x}_k = \phi(\mathbf{u}_k)$ for all k and let

$$c = -\sum_{k=1}^{K} \psi(\mathbf{x}_k), \tag{4.3}$$

where the summation is done modulo 4. Then the encoder output is the code sequence

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K, \mathbf{x}_{K+1}), \tag{4.4}$$

where the check sequence $\mathbf{x}_{K+1}$ is taken from $\mathcal{R}_c$. Note that $\sum_{k=1}^{K+1} \psi(\mathbf{x}_k) \equiv 0 \pmod 4$, which implies, together with the properties of $\mathcal{R}_i$, that any two different code sequences differ in at least 4 positions.

Proposition 4.1. *The minimum distance of the codebook is* $d_{min} = 4$.

Proof. Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K)$ and $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K)$ be two different codewords. Let all the subwords $\mathbf{x}_i = \mathbf{y}_i$, except for the two subwords $\mathbf{x}_j$ and $\mathbf{y}_j$. The subsets are constructed in such a way that permutations from the same subset have a minimum distance of 4 and permutations from different subsets have a minimum distance of 2.

If $\mathbf{x}_j$ and $\mathbf{y}_j$ belong to the same subset, thus $\psi(\mathbf{x}_j) = \psi(\mathbf{y}_j)$, then $\mathbf{x}_{K+1}$ and $\mathbf{y}_{K+1}$ will be chosen from the same subset. However, since $\mathbf{x}_j$ and $\mathbf{y}_j$ are from the same subset, the minimum distance between these two codewords will be 4.

If $\mathbf{x}_j$ and $\mathbf{y}_j$ belong to different subsets, thus $\psi(\mathbf{x}_j) \neq \psi(\mathbf{y}_j)$, then the minimum distance between $\mathbf{x}_j$ and $\mathbf{y}_j$ is 2. However, $\mathbf{x}_{K+1}$ and $\mathbf{y}_{K+1}$ will thus also be from different subsets and will have a minimum distance of 2. The minimum distance between these two codeword will thus be 4.

□

4.3.2 Decoding

Let the received sequence be

$$\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K, \mathbf{y}_{K+1}), \tag{4.5}$$

where each $\mathbf{y}_i$ is a sequence of four symbols from $\{1, 2, 3, 4\}$. The decoding procedure consists of the following steps:

1. If all $\mathbf{y}_k$ are in $\mathcal{R}$ and $\sum_{k=1}^{K+1} \psi(\mathbf{y}_k) \equiv 0 \pmod 4$, then set $\mathbf{x}' = \mathbf{y}$ and go to Step 4, else go to Step 2.

2. If there exists an e such that (i) $\mathbf{y}_e$ is not in $\mathcal{R}$, (ii) $\mathbf{y}_k$ is in $\mathcal{R}$ for all $k \neq e$, and (iii) $\mathbf{y}_e$ contains one symbol t from $\{1, 2, 3, 4\}$ twice, another symbol n not at all, and each of the other symbols once, then go to Step 3. Else, go to Step 5.

3. For $j = 1, 2$ set $\mathbf{x}^j$ as $\mathbf{y}$, with the jth symbol t in $\mathbf{y}_e$ replaced by symbol n. If there exists a j such that $\mathbf{x}_e^j \in \mathcal{R}$ and $\sum_{k=1}^{K+1} \psi(\mathbf{x}_k^j) \equiv 0 \pmod 4$, then set $\mathbf{x}' = \mathbf{x}^j$ for this j and go to Step 4. Else, go to Step 5.

4. Set the decoder output as $\mathbf{u}' = (\mathbf{u}'_1, \mathbf{u}'_2, \ldots, \mathbf{u}'_K)$, where $\mathbf{u}'_k = \phi^{-1}(\mathbf{x}'_k)$ for $k = 1, 2, \ldots, K$ and STOP.

5. Detect that at least two substitution errors have occurred and STOP.

4.3.3 Code Rate and Correcting Capabilities

The code has rate

$$\frac{4K}{4(K+1)} = \frac{K}{(K+1)} = 1 - \frac{1}{(K+1)} \text{ bits/symbol.} \tag{4.6}$$

This can be improved to

$$\frac{4K+2}{4(K+1)} = \frac{2K+1}{(2K+2)} = 1 - \frac{1}{2(K+1)} \text{ bits/symbol} \tag{4.7}$$

by encoding two extra information bits into the choice of $\mathbf{x}_{K+1}$, which is possible due to the fact that there are four options in the set $\mathcal{R}_c$. The rate is thus close to the optimal rate for large values of K. (Note that the code rate is specified as the number of binary bits represented by each symbol. The optimum rate, or mapping ratio, is thus the maximum number of binary bits that can be represented by each symbol.)

The proposed decoding procedure will correct any single substitution error and detect the occurrence of two substitution errors in the sequence of $4(K+1)$ transmitted symbols. The choice of K is a trade-off between efficiency (the higher the value of K, the higher the rate) and reliability (the lower the value of K, the lower the probability of uncorrected/undetected errors). The effect of K will further be investigated in Subsection 4.4.3.

This construction can also be seen as a subset of multipermutations. A codeword contains $K+1$ permutations. Thus, the codeword can be seen as a multipermutation where every symbol occurs $K+1$ times in the codeword. However, an additional restriction is imposed: the codeword is divided into $K+1$ blocks of length M and in every block a symbol may only occur once.

4.3.4 Mapping

The map ϕ used in the construction could be implemented through a table look-up. Still, it may be good to impose some structure, to allow simple encoding and decoding. The binary sequence $\mathbf{b} = (b_0, b_1, b_2, b_3)$ is uniquely mapped to a member $\mathbf{s} = (s_0, s_1, s_2, s_3)$ from $\mathcal{R}$ as follows: The integer representation p of (b_0, b_1) indicates that a member of $\mathcal{R}_p$ will be chosen. The integer representation q of (b_2, b_3) indicates that within $\mathcal{R}_p$ we choose the sequence with $s_q = 1$. For example, $\mathbf{b} = 1001$ has $p = 2$ and $q = 1$ and is thus mapped to $\mathbf{s} = 3142 \in \mathcal{R}_2$. Information can also be mapped to the parity permutations. This is discussed in Section 4.4.2.

4.3.5 Example

Let $K = 3$ and let the information sequence be $\mathbf{u} = (0111, 1000, 1010)$. Then the encoded sequence is $\mathbf{x} = (3421, 1324, 2413, 1342)$. Five cases are considered for the received sequence $\mathbf{y}$ and the corresponding decoding results are given.

- If $\mathbf{y} = \mathbf{x}$ (no errors), then we go from Step 1 to Step 4 and the decoding result is $\mathbf{u}' = \mathbf{u}$.

- If $\mathbf{y} = (3421, 1321, 2413, 1342)$, i.e. an error occurred in the eighth symbol, then we find in Step 1 that $\mathbf{y}_2$ is not in $\mathcal{R}$. Thus, in Step 2 we find that $e = 2$, $t = 1$ (the symbol which appears twice in $\mathbf{y}_2$), and $n = 4$ (the symbol which does not appear in $\mathbf{y}_2$). Hence, in Step 3 we get $\mathbf{x}_2^1 = 4321$ and $\mathbf{x}_2^2 = 1324$ and thus $\Sigma_{k=1}^4 \psi(\mathbf{x}_k^1) = 1{+}0{+}2{+}3 = 6 \equiv 2 \pmod 4$ and $\Sigma_{k=1}^4 \psi(\mathbf{x}_k^2) = 1{+}2{+}2{+}3 = 8 \equiv 0 \pmod 4$. In conclusion, $\mathbf{x}' = \mathbf{x}^2 = (3421, 1324, 2413, 1342)$ and Step 4 gives $\mathbf{u}' = \mathbf{u}$. Hence, the error has been corrected.

- If $\mathbf{y} = (3431, 1324, 2213, 1342)$, i.e. errors occurred in the third and tenth symbol, then we find that $\mathbf{y}_1$ and $\mathbf{y}_3$ are not in $\mathcal{R}$, and thus we go from Step 1 via Step 2 to Step 5, and the decoding result is the detection of (at least) two errors.

- If $\mathbf{y} = (3421, 3124, 2413, 1342)$, i.e. errors occurred in the fifth and sixth symbol, then we find in Step 1 that all symbols are in $\mathcal{R}$, but that $\Sigma_{k=1}^4 \psi(\mathbf{y}_k) = 1 + 3 + 2 + 3 = 9 \equiv 1 \pmod 4$. Via Step 2 we end up in Step 5, and the decoding result is the detection of (at least) two errors.

- If $\mathbf{y} = (3421, 1324, 3213, 1342)$, i.e. errors occurred in the ninth and tenth symbol, then we find in Step 1 that $\mathbf{y}_3$ is not in $\mathcal{R}$, and thus $e = 3$, $t = 3$ and $n = 4$ in Step 2. Hence, in Step 3 we get $\mathbf{x}_3^1 = 4213$ and $\mathbf{x}_3^2 = 3214$, and thus $\Sigma_{k=1}^4 \psi(\mathbf{x}_k^1) = 1 + 2 + 3 + 3 = 9 \equiv 1 \pmod 4$, while $\mathbf{x}_3^2 \notin \mathcal{R}$. In conclusion, we end up in Step 5, and the decoding result is the detection of (at least) two errors.

4.4 Code Construction for $M > 4$

This subsection will extend and generalize the construction given in the previous section to higher values of M. The prefix method from [24] is proposed to construct subsets of permutations of length M from the subsets of permutations of length $(M-1)$. The recursive process starts with the subsets as defined for $M = 4$ in Section 4.2.

Let $\mathcal{Q}$ be a partitioning of $\mathcal{S}_M$ into $M!/4$ subsets $\mathcal{R}_i$ of size 4 each, with the property that sequences within the same subset have distance 4. Similarly, let $\mathcal{Q}'$ be a partitioning of $\mathcal{S}_{M-1}$ into $(M-1)!/4$ subsets $\mathcal{R}'_i$ of size 4 each, with the same distance properties as $\mathcal{R}_i$.

Every subset in $\mathcal{Q}'$ can be extended and used to form M new subsets with 4 sequences of length M. Let $\mathcal{P}_{ij}$, $j = 0, 1, \ldots, M-1$, be the subsets created from $\mathcal{R}'_i$. The construction steps are:

1. Add the symbol M as a prefix to every permutation in $\mathcal{R}'_i$ to form $\mathcal{P}_{i0}$.
2. Let $j = 1, 2, \ldots, (M-1)$. To construct $\mathcal{P}_{ij}$, $swap(M, j)$ in every permutation contained in $\mathcal{P}_{i0}$.
3. All sets, $\mathcal{P}_{ij}$, $j = 0, 1, 2, \ldots, (M-1)$, are added to $\mathcal{Q}$.
4. Repeat the previous steps for all the subsets in $\mathcal{Q}'$.

The number of subsets that will be used during encoding and decoding is $L = \frac{2^v}{4}$, where $v = \lfloor \log_2 M! \rfloor$. $\mathcal{R}$ can then be formed by choosing any L subsets from $\mathcal{Q}$, thus $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_{L-1}$.

4.4.1 Example

Let $M = 5$ and $\mathcal{R}'_0 = \{1234, 2143, 3412, 4321\}$. The extended sets from $\mathcal{R}'_0$ are:

$$\mathcal{P}_{00} = \{51234, 52143, 53412, 54321\},$$
$$\mathcal{P}_{01} = \{15234, 12543, 13452, 14325\},$$
$$\mathcal{P}_{02} = \{21534, 25143, 23415, 24351\},$$
$$\mathcal{P}_{03} = \{31254, 32145, 35412, 34521\},$$
$$\mathcal{P}_{04} = \{41235, 42153, 43512, 45321\}.$$

The encoding and decoding procedures are similar to the $M = 4$ case, except that all summations are done modulo L.

4.4.2 Code Rate

The code rate is

$$\frac{vK}{M(K+1)} \text{ bits/symbol.} \tag{4.8}$$

The mapping can still be done as in the previous subsection, where the first $(v-2)$ binary bits represent the subset index and the last 2 binary bits represent the specific permutation in a subset.

The parity permutation can be any permutation in the subset $\mathcal{R}_c$. Two additional bits can be used to determine which of the permutations in $\mathcal{R}_c$ is used as the parity permutation. The code rate can thus be improved to

$$\frac{vK+2}{M(K+1)} \text{ bits/symbol.} \tag{4.9}$$

Again, the code rate, or mapping ratio, is specified as the number of binary bits represented by each symbol. The optimum rate, or mapping ratio, is thus the maximum number of binary bits that can be represented by each symbol.

4.4.3 Effect of K

The choice of K is a trade-off between efficiency and reliability. The optimal code rate for permutation codes is $\frac{\log_2(M!)}{M}$. If K is large, a code rate close to optimal can be obtained. Figure 4.2 shows the code rates for different values of K and M. The improved code rate of $\frac{vK+2}{M(K+1)}$ bits/symbol, as given in the previous subsection, is used.

However, as K increases, the probability that the error correction capability of the code will be exceeded, also increases. Errors will not be corrected if more than 1 error occurs in a codeword. Let P_e be the probability of a substitution error occurring and let $n = M(K+1)$ be the length of a codeword. Using Bernoulli trials, also known as the result of independent

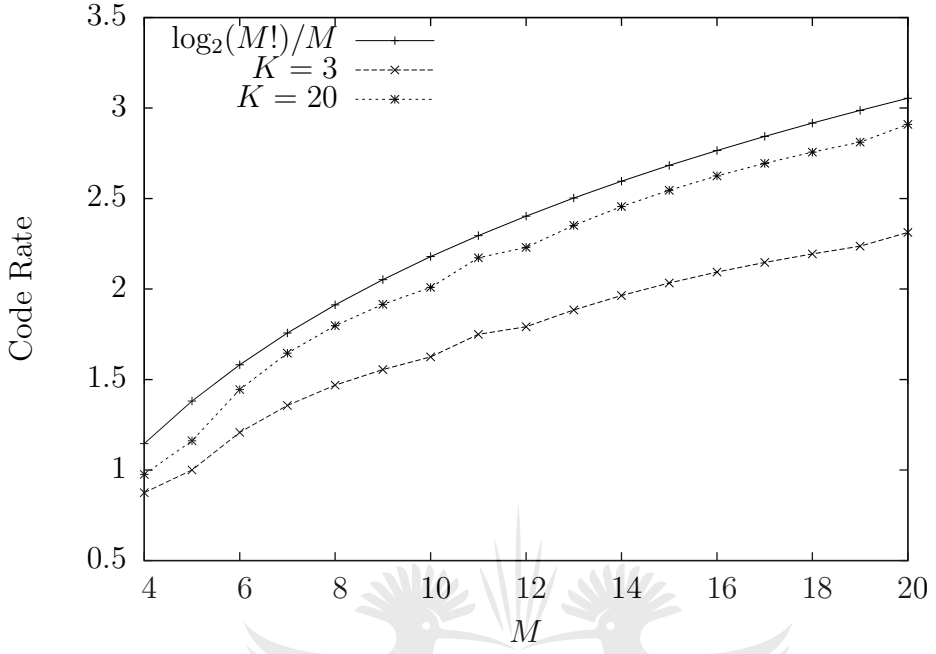

Figure 4.2: Code rates for different values of M and K

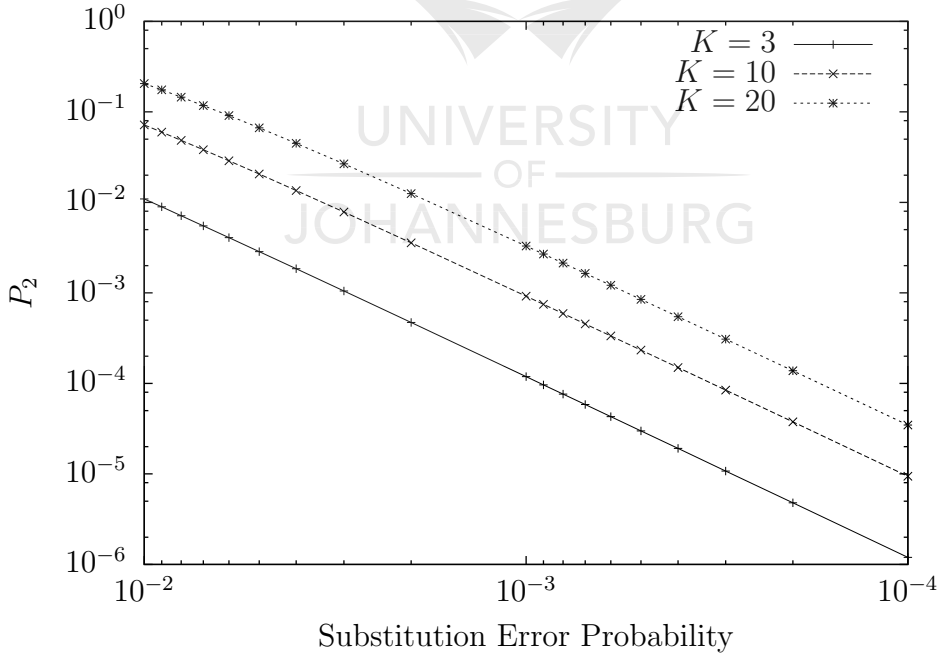

Figure 4.3: Probability that an error will not be corrected, $M = 4$

errors with probability P_e, the probability that an error will not be corrected, P_2, is given by

$$P_2 = 1 - (1 - P_e)^n - nP_e(1 - P_e)^{n-1}. \tag{4.10}$$

These probabilities are given in Figure 4.3 for $M = 4$ and different values of K.

4.5 Conclusion

Sequences of permutations, rather than single permutations, are used as codewords to achieve error control capabilities. A construction is presented capable of correcting one substitution error and is generalized for all values of M. A simple decoding algorithm is presented as well as a method to map from the binary data to permutations.

For any $M \geq 4$ and large values of K, the codes obtained have a code rate close to $\log_2(M!)/M$, which is the maximum value for the rate of any permutation code. Thus, the error control capabilities are achieved with very little loss in rate, especially when K is large.

Chapter 5

Concatenated Codes to Correct Insertion or Deletion Errors

5.1 Introduction

It was shown in the previous chapter that permutations can be concatenated to form codewords. The construction in Chapter 4 was designed to correct substitution errors. That construction is adapted in this chapter to be able to also detect and correct one deletion error per codeword. This is accomplished by using two substrings. A construction for $M = 4$ is firstly presented. The construction is then generalized for all values of $M > 4$. It is also shown that the decoding algorithm can be adapted to also detect and correct insertion errors. The decoding algorithm is thus able to determine from the length of the received sequence whether a deletion or an insertion error occurred and correct it, or it can correct two substitution errors, one in each substring. Lastly, it is shown that the resynchronizable codebooks from Chapter 3 can be concatenated to obtain better code rates.

The work presented in Section 5.2 has been published in [94].

5.2 Code Construction for $M = 4$

The construction presented in Chapter 4, which will be referred to as Construction I, is adapted in this section to provide deletion error correcting

capabilities. If a deletion does not occur, the construction will be able to detect and correct substitution errors.

In the adapted construction, or Construction II, the sequence of permutation words will be divided into two substrings. Each substring will use different subsets. The set $\mathcal{S}_4$ is divided into subsets as in Section 4.2. The two subsets which were not used in Construction I, will now form a different set that will be used exclusively by the second substring. Referring back to the subsets given in Section 4.2, let $\mathcal{T}_0 = \mathcal{R}_4$ and $\mathcal{T}_1 = \mathcal{R}_5$.

For all permutation sequences $\mathbf{r} \in \mathcal{R}_i$, define $\psi(\mathbf{r}) = i$, and for all permutation sequences $\mathbf{t} \in \mathcal{T}_j$, define $\xi(\mathbf{t}) = j$. Let $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$, and $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_1$. The construction method will exploit both $\mathcal{R}$ and $\mathcal{T}$, and all 24 permutations will be used. Using all 24 permutations gives a communications diversity benefit in some applications, e.g. the harsh PLC channel.

Let ϕ be a one-to-one mapping from the set $\mathcal{B}_4$ to $\mathcal{R}$, and let χ be a one-to-one mapping from the set $\mathcal{B}_3$ to $\mathcal{T}$.

5.2.1 Encoding

A source generates a sequence $\mathbf{u}$ of $7K$ bits, which is partitioned in K sequences $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$, all from $\mathcal{B}_7$. Each $\mathbf{u}_k$ is partitioned as $\mathbf{u}_k = (\mathbf{v}_k, \mathbf{w}_k)$, where $\mathbf{v}_k \in \mathcal{B}_4$ and $\mathbf{w}_k \in \mathcal{B}_3$. Let $\mathbf{x}_{2k-1} = \phi(\mathbf{v}_k)$ for all k and let $c_{\text{odd}} = -\Sigma_{k=1}^{K} \psi(\mathbf{x}_{2k-1})$, where the summation is done modulo 4. Similarly, let $\mathbf{x}_{2k} = \chi(\mathbf{w}_k)$ for all k and let $c_{\text{even}} = -\Sigma_{k=1}^{K} \xi(\mathbf{x}_{2k})$, where the summation is done modulo 2.

Then the encoder output is the code sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{2K}, \mathbf{x}_{2K+1}, \mathbf{x}_{2K+2})$, where the check sequence $\mathbf{x}_{2K+1}$ is taken from $\mathcal{R}_{c_{\text{odd}}}$ and the check sequence $\mathbf{x}_{2K+2}$ is taken from $\mathcal{T}_{c_{\text{even}}}$. Note that $\Sigma_{k=1}^{K+1} \psi(\mathbf{x}_{2k-1}) \equiv 0 \pmod 4$ and $\Sigma_{k=1}^{K+1} \xi(\mathbf{x}_{2k}) \equiv 0 \pmod 2$, which imply, together with the properties of the $\mathcal{R}_i$ and $\mathcal{T}_j$ subsets, that any two different code sequences differ in at least four positions.

5.2.2 Decoding

The occurrence of deletions can be observed from the length of the received sequence $\mathbf{y}$. If no deletions occur, we receive the sequence $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{2K}, \mathbf{y}_{2K+1}, \mathbf{y}_{2K+2})$, where each $\mathbf{y}_j$ is a sequence of four symbols from $\{1, 2, 3, 4\}$. By applying the procedure from the previous chapter on the odd substring $(\mathbf{y}_1, \mathbf{y}_3, \ldots, \mathbf{y}_{2K-1}, \mathbf{y}_{2K+1})$ and the even substring $(\mathbf{y}_2, \mathbf{y}_4, \ldots, \mathbf{y}_{2K}, \mathbf{y}_{2K+2})$, one substitution error can be corrected and two substitution errors can be detected in each of the substrings. In the procedure for the even substring, the roles of $\mathcal{R}$, ϕ, and ψ are substituted by $\mathcal{T}$, χ, and ξ, respectively, in a straightforward way.

If one deletion occurs (and no substitution errors), it can be corrected by the following procedure, which exploits the fact that all sequences in $\mathcal{T}$ starts with 14, 41, 23 or 32, while none of the sequences in $\mathcal{R}$ starts with such a combination.

1. Partition the received sequence $\mathbf{y}$ as $(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{2K}, \mathbf{y}_{2K+1}, \mathbf{y}_{2K+2})$, where $\mathbf{y}_j$ are sequences of symbols from $\{1, 2, 3, 4\}$, which are of length four, except the last one, which is of length three. Partition each $\mathbf{y}_j$ into two subsequences, i.e., $\mathbf{y}_j = (\mathbf{y}_j^{\text{head}}, \mathbf{y}_j^{\text{tail}})$, where each subsequence is of length two, except $\mathbf{y}_{2K+1}^{\text{tail}}$, which is of length one. Let $\mathbf{y}^{s,p,k}$ denote the sequences which are obtained from $\mathbf{y}$ by inserting the symbols $s \in \{1, 2, 3, 4\}$ at position $p \in \{0, 1, 2, 3\}$ of $\mathbf{y}_k$.

2. Find the smallest value of $e \in \{1, 2, \ldots, K+1\}$ such that $\mathbf{y}_{2e}^{\text{head}}$ is not equal to 14, 41, 23 or 32. If no violated $\mathbf{y}_{2e}^{\text{head}}$ is found, then set $e = K + 1$.

3. If (i) $\mathbf{y}_{2e-2} \in \mathcal{T}$, (ii) $\mathbf{y}_{2e-1} \in \mathcal{R}$, and (iii) $\Sigma_{k=1}^{K+1} \psi(\mathbf{y}_{2k-1}^{1,0,2e}) \equiv 0 \pmod 4$, then set $\mathbf{x}' = \mathbf{y}^{s,p,2e}$, where s and p are chosen such that $\mathbf{y}_{2e}^{s,p,2e} \in \mathcal{T}$ and $\Sigma_{k=1}^{K=1} \xi(\mathbf{y}_{2k}^{s,p,2e}) \equiv 0 \pmod 2$, and go to Step 6.

4. If (i) $\mathbf{y}_{2e-2} \in \mathcal{T}$, (ii) $\Sigma_{k=1}^{K+1} \xi(\mathbf{y}_{2k}^{1,0,2e-1}) \equiv 0 \pmod 2$, and (iii) $\mathbf{y}_{2e-1} \notin \mathcal{R}$ or $\Sigma_{k=1}^{K+1} \psi(\mathbf{y}_{2k-1}^{1,0,2e}) \equiv 1, 2, 3 \pmod 4$, then set $\mathbf{x}' = \mathbf{y}^{s,p,2e-1}$, where s and p are chosen such that $\mathbf{y}_{2e-1}^{s,p,2e-1} \in \mathcal{R}$ and $\Sigma_{k=1}^{K+1} \psi(\mathbf{y}_{2k-1}^{s,p,2e-1}) \equiv 0 \pmod 4$, and go to Step 6.

5. Set $\mathbf{x}' = \mathbf{y}^{s,p,2e-2}$, where s and p are chosen such that $\mathbf{y}_{2e-2}^{s,p,2e-2} \in \mathcal{T}$ and $\Sigma_{k=1}^{K+1} \xi(\mathbf{y}_{2k}^{s,p,2e-2}) \equiv 0 \pmod 2$.

6. Set the decoder output as $\mathbf{u}' = (\mathbf{u}'_1, \mathbf{u}'_2, \ldots, \mathbf{u}'_K)$, where $\mathbf{u}'_k = (\phi^{-1}(\mathbf{x}'_{2k-1)}), \chi^{-1}(\mathbf{x}'_{2k}))$ for $k = 1, 2, \ldots, K$, and STOP.

In Step 2, the deletion in the transmitted code sequence $\mathbf{y}$ is determined to have occurred in $\mathbf{y}^{\text{tail}}_{2e-2}$, $\mathbf{y}_{2e-1}$ or $\mathbf{y}^{\text{head}}_{2e}$. Then, in Steps 3-5, it is determined whether the deletion occurred in $\mathbf{y}^{\text{head}}_{2e}$ (if the three conditions in Step 3 are all true), in $\mathbf{y}_{2e-1}$ (if the three conditions in Step 4 are all true), or in $\mathbf{y}^{\text{tail}}_{2e-2}$ (otherwise). Insertions are made accordingly. Finally, the binary information sequence is retrieved in Step 6.

5.2.3 Code Rate and Correcting Capabilities

The code has rate $\frac{7K}{8(K+1)}$ bits/symbol. This can be improved to $\frac{7K+4}{8(K+1)}$ by encoding two extra information bits into the choice of $\mathbf{x}_{2K+1}$ and two extra information bits into the choice of $\mathbf{x}_{2K+2}$. The rate is close to $(7/8)$ for large values of K.

If no deletions occur, the proposed decoding procedure will correct any single substitution error and detect the occurrence of two substitution errors in each of the two substrings of length $4(K + 1)$. If a single deletion error occurs and no substitution errors, then the deletion will be corrected by the proposed procedure. Again, the choice of K is a trade-off between efficiency (the higher the value of K, the higher the rate) and reliability (the lower the value of K, the lower the probability of uncorrected/undetected errors).

5.2.4 Mapping

The maps ϕ and χ used in the construction could be implemented through a table look-up. Still, some structure can be imposed. For ϕ, this can be done as described in the previous section. For χ, this can be done as follows: The binary sequence $\mathbf{b} = (b_0, b_1, b_2)$ is uniquely mapped to a member $\mathbf{t} = (t_0, t_1, t_2, t_3)$ from $\mathcal{T}$ as follows: The value of $b = b_0$ indicates that a member of $\mathcal{T}_b$ will be chosen. The integer representation q of (b_1, b_2) indicates that within $\mathcal{T}_b$ we choose the sequence with $t_q = 1$. For example, $\mathbf{b} = 110$ has $b = 1$ and $q = 2$ and is thus mapped to $\mathbf{t} = 3214 \in \mathcal{T}_1$.

5.2.5 Example

Let K=3 and let the information sequence be $\mathbf{u} = (0111, 000, 1000, 111, 1010, 101)$. Then the code sequence is $\mathbf{x} = (3421, 1423, 1324, 2341, 2413, 4123, 1342, 1423)$. We consider two cases for the received sequence $\mathbf{y}$ and give the corresponding decoding results.

- If $\mathbf{y} = (3423, 1423, 1324, 2241, 2413, 4123, 1342, 1423)$, i.e., substitution errors occurred in the fourth and fourteenth symbol, then the decoder observes that $\mathbf{y}_1$ is not in $\mathcal{R}$ and $\mathbf{y}_4$ is not in $\mathcal{T}$. From $\psi(\mathbf{y}_3)+\psi(\mathbf{y}_5)+\psi(\mathbf{y}_7) = 2+2+3 = 7 \equiv 3 \pmod 4$, the decoder finds that the first substrings should be in $\mathcal{R}_1$, and thus $\mathbf{x}'_1 = 3421$. Similarly, from $\xi(\mathbf{y}_2)+\xi(\mathbf{y}_6)+\xi(\mathbf{y}_8) = 0+1+0 \equiv 1 \pmod 2$, the decoder finds that the fourth substring should be in $\mathcal{T}_1$, and thus $\mathbf{x}'_4 = 2341$. Hence, both errors are corrected.
- If $\mathbf{y} = (3421, 1423, 1342, 3412, 4134, 1231, 3421, 423)$, i.e., a deletion occurred in the eleventh symbol, then the decoder detects that a deletion occurred from the length of the sequence $\mathbf{y}$. Since $\mathbf{y}_2^{\text{head}} = 14$ and $\mathbf{y}_4^{\text{head}} = 34$, the deletion is determined to have occurred in $\mathbf{y}_2^{\text{tail}}$, $\mathbf{y}_3$ or $\mathbf{y}_4^{\text{head}}$, i.e., $e = 2$ in Step 2 of the described decoding algorithm. Since $\Sigma_{k=1}^{4}\psi(\mathbf{y}_{2k-1}^{1,0,2e}) \equiv 1 \pmod 4$, it follows from Step 4 that the symbol $s = 2$ should be inserted at position $p = 2$ of $\mathbf{y}_3$. The resulting sequence $\mathbf{x}'$ is correctly decoded in Step 6.

5.3 Code construction for $M > 4$

In the previous section, all permutation sequences in $\mathcal{T}$ starts with 14, 41, 23 or 32. Let $\mathcal{T}'$ denote all the allowable start sequences for permutations in $\mathcal{T}$ and thus $\mathcal{R}'$ contains all the allowable start sequences for permutations in $\mathcal{R}$. When generalizing to higher values of M, identifying $\mathcal{T}'$ is the challenge. For example, assume that $\mathcal{T}' = (12, 13, 31, 21)$. If the transmitted sequence is $\mathbf{x} = (3421, 1234, 2413, 2134)$ and a deletion error occurs in position 6, then the received sequence is $y = (3421, 1342, 4132, 134)$. Even though symbol 2 is deleted from the second subword, sequence 13 is still a valid start sequence. The error is thus not detected at the correct position and it is assumed that the error occurred in the last subword. The start sequences will thus need to be constructed in such a way that this cannot occur.

The subsets are constructed using the following steps:

1. Identify $\mathcal{T}'$. If M is even, $\mathcal{T}' = 12, 21, 34, 43, \ldots (M-1)M, M(M-1)$. If M is odd, $\mathcal{T}' = 12, 21, 34, 43, \ldots (M-2)(M-1), (M-1)(M-2)$. $\mathcal{R}'$ contains all other combinations not included in $\mathcal{T}'$.

   The number of possible start sequences in $\mathcal{T}'$ is

$$|\mathcal{T}'| = 2\left\lfloor \frac{M}{2} \right\rfloor, \tag{5.1}$$

   and the number of sequences in $\mathcal{R}'$ is

$$|\mathcal{R}'| = \frac{M!}{(M-2)!} - 2\left\lfloor \frac{M}{2} \right\rfloor. \tag{5.2}$$

2. Let $\mathcal{T}^*$ contain all possible permutations starting with the sequences in $\mathcal{T}'$. Every sequence in $\mathcal{T}'$ is combined with every possible permutation of the remaining $(M-2)$ symbols not yet used. The set $\mathcal{R}^*$ is constructed similarly from $\mathcal{R}'$.

   The number of sequences in $\mathcal{T}^*$ is

$$|\mathcal{T}^*| = 2(M-2)!\left\lfloor \frac{M}{2} \right\rfloor \tag{5.3}$$

   and the number of sequences in $\mathcal{R}^*$ is

$$|\mathcal{R}^*| = (M-2)!\left(\frac{M!}{(M-2)!} - 2\left\lfloor \frac{M}{2} \right\rfloor\right). \tag{5.4}$$

3. The permutations in $\mathcal{T}^*$ and $\mathcal{R}^*$ are now grouped together into subsets, where the minimum Hamming distance of each subset is $d_{\min} = 4$. This can be done in various ways. A computer search was done to complete the grouping.

4. Not all subsets will be used during encoding and decoding. The length of the binary sequences, v which will be mapped to $\mathcal{T}^*$ is calculated as follows:

$$v = \lfloor \log_2 |\mathcal{T}^*| \rfloor. \tag{5.5}$$

The number of permutations which will be used from $\mathcal{T}^*$ is thus 2^v. The number of subsets from $\mathcal{T}^*$ which will be used is then:

$$L_T = \left\lceil \frac{2^v}{4} \right\rceil. \tag{5.6}$$

Similarly, the length of the binary sequences, w which will be mapped to $\mathcal{R}^*$ is calculated as follows:

$$w = \lfloor \log_2 |\mathcal{R}^*| \rfloor. \tag{5.7}$$

The number of permutations which will be used from $\mathcal{R}^*$ is thus 2^w. The number of subsets from $\mathcal{R}^*$ which will be used is then:

$$L_R = \left\lceil \frac{2^w}{4} \right\rceil. \tag{5.8}$$

5. As in the previous section $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cdots \cup \mathcal{R}_{L_R}$, and $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_1 \cdots \cup \mathcal{T}_{L_T}$.

The encoding and decoding procedure will be the same as described in Section 5.2.

The number of iterations needed in Step 3 (grouping the permutations together in subsets), grows exponentially as the value of M increases. A random permutation is taken from the relevant set, either $\mathcal{T}^*$ or $\mathcal{R}^*$, and added to the subset $\mathcal{T}_i^*$ or $\mathcal{R}_i^*$. Then the rest of the permutations in the specific set is tested to determine whether it has a distance of at least 4 to the first permutation in subset $\mathcal{T}_i^*$ or $\mathcal{R}_i^*$. If it has a distance of at least 4, then the permutation is removed from the original set ($\mathcal{T}^*$ or $\mathcal{R}^*$), and added to subset ($\mathcal{T}_i^*$ or $\mathcal{R}_i^*$). If the subset contains 4 permutations, or no

more permutations with a distance of at least 4 is found, then the next subset is formed. This procedure is repeated until there is no more permutations in the original set ($\mathcal{T}^*$ or $\mathcal{R}^*$). Thus, the complexity grows exponentially as the value of M increases. However, this is acceptable since this step is only performed once during the initialization stage. (In computer science the complexity is given by Big O notation [95]. In this case it would be $O(M!^2)$.)

5.3.1 Example

Let $M = 5$, then $\mathcal{T}' = (12, 21, 34, 43)$ and $\mathcal{R}' = \{13, 14, 15, 23, 24, 25, 31, 32, 35, 41, 42, 45, 51, 52, 53, 54\}$. The length of the binary sequences which will be mapped to the permutations in $\mathcal{T}$ is $v = 4$ and thus $L_v = 16$. The length of the binary sequences which will be mapped to the permutations in $\mathcal{R}$ is $w = 6$ and thus $L_w = 64$. The subsets which will be used are:

$$\mathcal{T}_0 = \{34152, 12453, 43125, 21435\}$$

$$\mathcal{T}_1 = \{21534, 34251, 43215, 12354\}$$

$$\mathcal{T}_2 = \{21543, 34215, 43512, 12345\}$$

$$\mathcal{T}_3 = \{12435, 21453, 34125, 43251\}$$

$$\mathcal{R}_0 = \{52143, 13452, 23514, 45132\}$$

$$\mathcal{R}_1 = \{54312, 13524, 31425, 24135\}$$

$$\mathcal{R}_2 = \{53124, 52341, 54213, 41523\}$$

$$\mathcal{R}_3 = \{32154, 15423, 14235, 51324\}$$

$$\mathcal{R}_4 = \{13254, 51423, 25134, 42153\}$$

$$\mathcal{R}_5 = \{32415, 41325, 53421, 24531\}$$

$$\mathcal{R}_6 = \{41532, 25431, 52134, 13245\}$$

$$\mathcal{R}_7 = \{32514, 14532, 45213, 24315\}$$

$$\mathcal{R}_8 = \{54321, 15432, 23541, 51234\}$$

$$\mathcal{R}_9 = \{52413, 15342, 23451, 42531\}$$

$$\mathcal{R}_{10} = \{41235, 32451, 45321, 15243\}$$

$$\mathcal{R}_{11} = \{13542, 51432, 45123, 24351\}$$

$$\mathcal{R}_{12} = \{32541, 14253, 35412, 54132\}$$

$$\mathcal{R}_{13} = \{54231, 23415, 31542, 35124\}$$

$$\mathcal{R}_{14} = \{14523, 15234, 31245, 23154\}$$

$$\mathcal{R}_{15} = \{52314, 53142, 24153, 41352\}$$

5.3.2 Code Rate

The code rate is given by

$$R = \frac{(v+w)K}{2M(K+1)}. \tag{5.9}$$

This can be improved to

$$R = \frac{(v+w)K+4}{2M(K+1)} \tag{5.10}$$

by encoding two extra information bits into the choice of $\mathbf{x}_{2K+1}$ and two extra information bits into the choice of $\mathbf{x}_{2K+2}$. Figure 5.1 shows the code rates for different values of K and M. The improved code rate is used.

The error correction capabilities of the construction is the same as in Section 5.2.

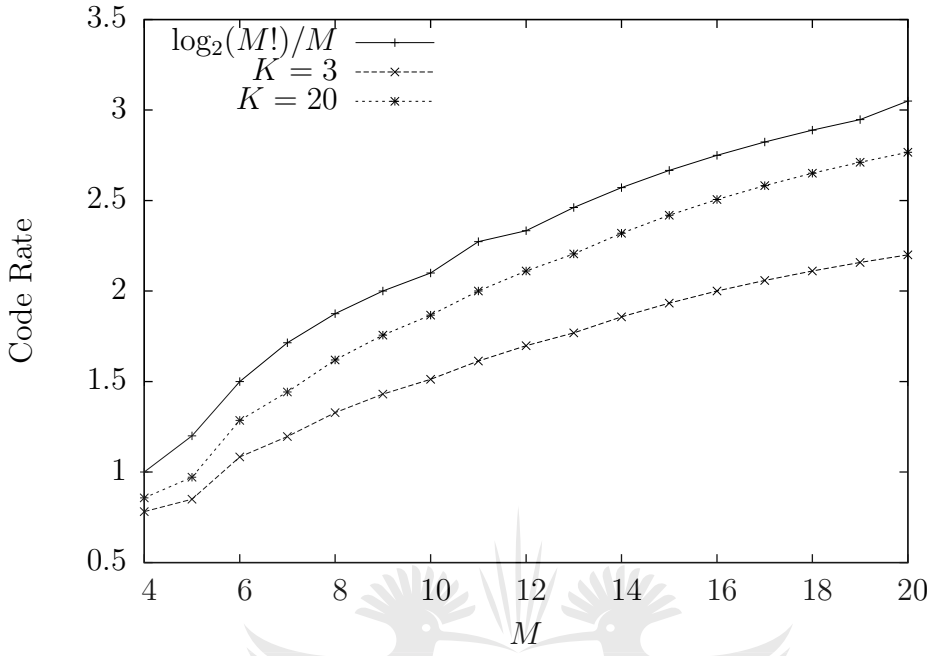

Figure 5.1: Construction II: Code rates for different values of M and K

5.4 Concatenated Codes to Correct Insertion Errors

Although the focus of the chapter is on deletion errors, the question of insertion errors may arise. The decoding algorithm of Section 5.2 is adapted to also be able to correct insertion errors.

In the previous section a codebook is constructed with deletion errors in mind. Every permutation in the $\mathcal{T}$ set starts with a specific start sequence. If a deletion error occurs in position j, every symbol after position j will shift one position to the left, i.e. every symbol will be shifted forward in time. The permutations from $\mathcal{T}$ is constructed in such a way that, no matter which symbol shifts into the head of a permutation from $\mathcal{T}$, it will not form a valid start sequence. Refer to Section 5.3 for more detail on how the permutations in $\mathcal{T}$ are constructed.

However, if an insertion error occurs in position k, every symbol from position k onwards will shift one position to the right, i.e. every symbol will be delayed in time. A symbol from a permutation belonging to set $\mathcal{R}$ will thus

shift into the head of the permutations from set $\mathcal{T}$. It is thus possible that the head sequence can still be valid, even after an insertion error occurred. For example, let the encoded sequence be $\mathbf{x} = (2134, 1423, 3142, 3214)$. If a symbol is inserted at the beginning of the transmitted sequence, every sequence will shift one position to the right and the received sequence is $\mathbf{x} = (1213, 4142, 3314, 23214)$. The head sequences 41 and 23 are valid head sequences for permutations from the set $\mathcal{T}$.

Thus, in order to be able to correct insertion errors, one will have to use the tail sequences from $\mathcal{T}$ instead of the head sequences. Even though the permutations in $\mathcal{T}$ were chosen to have unique head sequences, these permutations will also have unique tail sequences since we are working with permutations.

5.4.1 Decoding

The decoding procedure is given for $M = 4$. It has been shown in Section 5.3 that this can easily be generalized for values $M > 4$. The codebook given in Section 4.2 is used, where $\mathcal{T}_0 = \mathcal{R}_4$ and $\mathcal{T}_1 = \mathcal{R}_5$.

The occurrence of either an insertion error or a deletion error can be observed from the length of the received sequence $\mathbf{y}$. If no deletions occur, we receive the sequence $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{2K}, \mathbf{y}_{2K+1}, \mathbf{y}_{2K+2})$, where each $\mathbf{y}_j$ is a sequence of four symbols from $\{1, 2, 3, 4\}$. By applying the procedure from the previous chapter on the odd substring $(\mathbf{y}_1, \mathbf{y}_3, \ldots, \mathbf{y}_{2K-1}, \mathbf{y}_{2K+1})$ and the even substring $(\mathbf{y}_2, \mathbf{y}_4, \ldots, \mathbf{y}_{2K}, \mathbf{y}_{2K+2})$, one substitution error can be corrected and two substitution errors can be detected in each of the substrings.

If one deletion occurs (and no substitution errors), it can be corrected by the procedure given in Section 5.2, which exploits the fact that all sequences in $\mathcal{T}$ starts with 14, 41, 23 or 32, while none of the sequences in $\mathcal{R}$ starts with such a combination.

If one insertion occurs (and no substitution errors), it can be corrected by the procedure given below, which exploits the fact that all sequences in $\mathcal{T}$ ends with 14, 41, 23 or 32, while none of the sequences in $\mathcal{R}$ ends with such a combination.

1. Partition the received sequence $\mathbf{y}$ as $(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{2K}, \mathbf{y}_{2K+1}, \mathbf{y}_{2K+2})$, where $\mathbf{y}_j$ are sequences of symbols from $\{1, 2, 3, 4\}$, which are of length four, except the last one, which is of length five. Partition each $\mathbf{y}_j$ into two subsequences, i.e., $\mathbf{y}_j = (\mathbf{y}_j^{\text{head}}, \mathbf{y}_j^{\text{tail}})$, where each subsequence is of length two, except $\mathbf{y}_{2K+1}^{\text{tail}}$, which is of length three. Let $\mathbf{y}^{p,k}$ denote the sequences which are obtained from $\mathbf{y}$ by deleting the symbol at position $p \in \{0, 1, 2, 3\}$ of $\mathbf{y}_k$.

2. Find the smallest value of $e \in \{1, 2, \ldots, K+1\}$ such that $\mathbf{y}_{2e}^{\text{tail}}$ is not equal to 14, 41, 23 or 32.

3. If (i) $\mathbf{y}_{2e-2} \in \mathcal{T}$, (ii) $\mathbf{y}_{2e-1} \in \mathcal{R}$, and (iii) $\Sigma_{k=1}^{K+1} \psi(\mathbf{y}_{2k-1}^{0,2e}) \equiv 0 \pmod 4$, then set $\mathbf{x}' = \mathbf{y}^{p,2e}$, where p is chosen such that $\mathbf{y}_{2e}^{p,2e} \in \mathcal{T}$ and $\Sigma_{k=1}^{K=1} \xi(\mathbf{y}_{2k}^{p,2e}) \equiv 0 \pmod 2$, and go to Step 6.

4. If (i) $\mathbf{y}_{2e-2} \in \mathcal{T}$, (ii) $\Sigma_{k=1}^{K+1} \xi(\mathbf{y}_{2k}^{0,2e-1}) \equiv 0 \pmod 2$, and (iii) $\mathbf{y}_{2e-1} \notin \mathcal{R}$ or $\Sigma_{k=1}^{K+1} \psi(\mathbf{y}_{2k-1}^{1,0,2e}) \equiv 1, 2, 3 \pmod 4$, then set $\mathbf{x}' = \mathbf{y}^{p,2e-1}$, where p is chosen such that $\mathbf{y}_{2e-1}^{p,2e-1} \in \mathcal{R}$ and $\Sigma_{k=1}^{K+1} \psi(\mathbf{y}_{2k-1}^{p,2e-1}) \equiv 0 \pmod 4$, and go to Step 6.

5. Set $\mathbf{x}' = \mathbf{y}^{p,2e-2}$, where p is chosen such that $\mathbf{y}_{2e-2}^{p,2e-2} \in \mathcal{T}$ and $\Sigma_{k=1}^{K+1} \xi(\mathbf{y}_{2k}^{p,2e-2}) \equiv 0 \pmod 2$.

6. Set the decoder output as $\mathbf{u}' = (\mathbf{u}'_1, \mathbf{u}'_2, \ldots, \mathbf{u}'_K)$, where $\mathbf{u}'_k = (\phi^{-1}(\mathbf{x}'_{2k-1)}), \chi^{-1}(\mathbf{x}'_{2k}))$ for $k = 1, 2, \ldots, K$, and STOP.

In Step 2, the insertion in the transmitted code sequence $\mathbf{y}$ is determined to have occurred in $\mathbf{y}_{2e-2}^{\text{tail}}$, $\mathbf{y}_{2e-1}$ or $\mathbf{y}_{2e}$. It is possible that an insertion occurred in $\mathbf{y}_{2e-2}^{\text{tail}}$ and that $\mathbf{y}_{2e-2}^{\text{tail}}$ is still a valid end sequence, depending on the value of the inserted symbol. Then, in Steps 3-5, it is determined whether the deletion occurred in $\mathbf{y}_{2e}$ (if the three conditions in Step 3 are all true), in $\mathbf{y}_{2e-1}$ (if the three conditions in Step 4 are all true), or in $\mathbf{y}_{2e-2}$ (otherwise). Deletions are made accordingly. Finally, the binary information sequence is retrieved in Step 6.

5.4.2 Example

The same parameters and sequences are used as in Section 5.2.5: Let K=3 and let the information sequence be $\mathbf{u} = (0111, 000, 1000, 111, 1010, 101)$ and the encoded sequence be $\mathbf{x} = (3421, 1423, 1324, 2341, 2413, 4123, 1342, 1423)$. We consider three cases for the received sequence $\mathbf{y}$ and give the corresponding decoding results.

- If $\mathbf{y} = (3421, 1423, 1324, 1234, 1241, 3412, 3134, 21423)$, i.e. symbol 1 is inserted in position 13, then the decoder detects that an insertion occurred from the length of the sequence $\mathbf{y}$. Since $\mathbf{y}_2^{\text{tail}} = 23$ and $\mathbf{y}_4^{\text{tail}} = 34$, the insertion is determined to have occurred in $\mathbf{y}_2$, $\mathbf{y}_3$ or $\mathbf{y}_4$, i.e., $e = 2$ in Step 2 of the described decoding algorithm. Since all the conditions of Step 3 are satisfied, the insertion error occurred in y_4. For $p = 0$, $2341 \in \mathcal{T}$ and $\Sigma_{k=1}^{K=1} \xi(\mathbf{y}_{2k}^{0,2e}) \equiv 0 \pmod 2$.

- If $\mathbf{y} = (3421, 1423, 1323, 4234, 1241, 3412, 3134, 21423)$, i.e. symbol 3 is inserted in position 12, then the decoder detects that an insertion occurred from the length of the sequence $\mathbf{y}$. Since $\mathbf{y}_2^{\text{tail}} = 23$ and $\mathbf{y}_4^{\text{tail}} = 34$, the insertion is determined to have occurred in $\mathbf{y}_2$, $\mathbf{y}_3$ or $\mathbf{y}_4$, i.e., $e = 2$ in Step 2 of the described decoding algorithm. In Step 3 we observe that $\mathbf{y}_3$ is not in $\mathcal{R}$ and continue to Step 4. All the conditions of Step 4 are satisfied and an insertion has thus occurred in $\mathbf{y}_3$. It follows that $p = 3$ and thus the symbol at position 3 is deleted.

- If $\mathbf{y} = (3421, 1432, 3132, 4234, 1241, 3412, 3134, 21423)$, i.e. symbol 3 is inserted in position 7, then the decoder detects that an insertion occurred from the length of the sequence $\mathbf{y}$. Since $\mathbf{y}_2^{\text{tail}} = 32$ and $\mathbf{y}_4^{\text{tail}} = 34$, the insertion is determined to have occurred in $\mathbf{y}_2$, $\mathbf{y}_3$ or $\mathbf{y}_4$, i.e., $e = 2$ in Step 2 of the described decoding algorithm. (Note that due to the specific insertion, even though it occurred in $\mathbf{y}_2^{\text{tail}}$, the sequence is still a valid tail sequence.) In Step 3 we observe that $\mathbf{y}_3$ is not in $\mathcal{R}$ and continue to Step 4. In Step 4 we find that even though $\mathbf{y}_2 \in \mathcal{T}$, $\Sigma_{k=1}^{K+1} \xi(\mathbf{y}_{2k}^{0,2e-1})$ can not be calculated since $\mathbf{y}_4$ is not in $\mathcal{T}$. In Step 5, we find that the conditions are satisfied if $p = 2$.

5.4.3 Remarks

The $M = 4$ case is considered in the previous section. For larger values of M, if the codebooks are constructed as specified in Section 5.3, it will be possible to also correct an insertion error. However, the tail sequences which will be used will be of length $M - 2$, since the head sequences which are used has a length of 2. This will result in more computations needed to determine whether a tail sequence is valid. This is acceptable since the focus of the chapter is on deletions, and we assume that insertion errors are not that probable.

If insertion errors are more probable than deletion errors, the head and tail sequences can be swapped, so that the tail sequences are constructed as specified in Section 5.3. If deletion and insertion errors are equiprobable, one can always consider making the head and tail sequences more or less equal. This will not be covered in this thesis.

5.5 Concatenated Resynchronizable Codes

In Chapter 3, only one partition of the set $\mathcal{S}_{l_1}$ was used in combination with one partition from the set $\mathcal{S}_{l_2}$. When concatenation is used, more partitions can be used leading to higher cardinalities and code rates. However, the resynchronizable codes will still have the ability to resynchronize. This is similar to the previous sections in this chapter, where two substrings are used and permutations from the second substring can only start with a limited number of symbols. The resynchronizable concatenated codes will be compared to these codes to determine which is most suited.

5.5.1 Set Construction

Every codeword of length M is divided into two segments of length l_1 and l_2 respectively. Before encoding, the subsets need to be constructed. The construction procedure consists of the following steps:

1. Using the method in [57], divide the set $\mathcal{S}_{l_1}$ into l_1 partitions, $\mathcal{P}_1 = \mathcal{P}_{11}, \mathcal{P}_{12}, \ldots, \mathcal{P}_{1l_1}$. Each partition, $\mathcal{P}_{1i}$ contains $(l_1 - 1)!$ permutations capable of correcting one deletion error.

2. Using the method in [57], divide the set $\mathcal{S}_{l_2}$ into l_2 partitions, $\mathcal{P}_2 = \mathcal{P}_{21}, \mathcal{P}_{22}, \ldots, \mathcal{P}_{2l_2}$. Each partition, $\mathcal{P}_{2j}$ contains $(l_2 - 1)!$ permutations capable of correcting one deletion error.

3. Add l_1 to each symbol in $\mathcal{P}_{2j}$.

4. To form the subsets $\mathcal{R}_m$, the sequences in $\mathcal{P}_{1i}$ are combined with the sequences in $\mathcal{P}_{2j}$, where the number of possible subsets is given by $l_1 l_2$.

Table 5.1: Partitions for Segment 1

$\mathcal{P}_{11}$	$\mathcal{P}_{12}$	$\mathcal{P}_{13}$
123	132	213
321	231	312

Table 5.2: Partitions for Segment 2

$\mathcal{P}_{21}$	$\mathcal{P}_{22}$	$\mathcal{P}_{23}$
456	465	546
654	564	645

Example: Let $M = 6$, $l_1 = l_2 = 3$ and $K = 3$. Then, the partitions for $M = 3$ from [57] which will be used in the first segment is given in Table 5.1.

The partitions for the second segment of each codeword are the same partitions from Table 5.1, but l_1 is added to each symbol. The partitions are given in Table 5.2.

The sets which will thus be used in the concatenated codebook are:

$$\mathcal{R}_0 = \{123456, 123654, 321456, 321654\}$$

$$\mathcal{R}_1 = \{123465, 123564, 321465, 321564\}$$

$$\mathcal{R}_2 = \{123546, 123645, 321546, 321645\}$$

$$\mathcal{R}_3 = \{132456, 132456, 231654, 231654\}$$

$$\mathcal{R}_4 = \{132465, 132465, 231564, 231564\}$$

$$\mathcal{R}_5 = \{132546, 132546, 231645, 231645\}$$

$$\mathcal{R}_6 = \{213456, 312456, 213654, 312654\}$$

$$\mathcal{R}_7 = \{213465, 312465, 213564, 312564\}$$

$$\mathcal{R}_8 = \{213546, 312546, 213645, 312645\}$$

5.5.2 Encoding

The length of the binary codewords, v, which will be mapped to the permutations is

$$v = \lfloor \log_2[l_1 l_2 (l_1 - 1)!(l_2 - 1)!] \rfloor. \tag{5.11}$$

The number of permutations which will be used is thus 2^v. The minimum number of subsets which is needed, L, is

$$L = \left\lceil \frac{2^v}{M} \right\rceil. \tag{5.12}$$

If $2^v \mod M \equiv 0$, then all these subsets will contain M permutations, otherwise the last subset will not be complete and only contain $2^v \mod M$ permutations.

Let $\mathcal{B}_v$ be the set consisting of all binary sequences of length v and let K be a positive integer. For all permutation sequences $\mathbf{r} \in \mathcal{R}_i$, define $\psi(\mathbf{r}) = i$. Let $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_{L-1}$. Let ϕ be a one-to-one mapping from the set $\mathcal{B}_v$ to $\mathcal{R}$.

A source generates a sequence $\mathbf{u}$ of vK bits, which is partitioned into K sequences $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$, all from $\mathcal{B}_v$. Let $\mathbf{x}_k = \phi(\mathbf{u}_k)$ for all k and let $c = -\sum_{k=1}^{K} \psi(\mathbf{x}_k)$, where the summation is done modulo L. Then the encoder output is the code sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K, \mathbf{x}_{K+1})$, where the check sequence $\mathbf{x}_{K+1}$ is taken from $\mathcal{R}_c$. Note that $\sum_{k=1}^{K+1} \psi(\mathbf{x}_k) \equiv 0 \pmod{L}$.

In Chapter 3, the decoder uses a Subsequence Lookup Table to determine which subword belongs to which codeword. Since we are working with all the partitions, every subset, $\mathcal{R}_i$, will have a separate Subsequence Lookup Table. The decoder will thus have to determine which subset's Subsequence Lookup Table to use when correcting deletion errors. For every codeword in $\mathcal{R}_i$, the Subsequence Lookup Table will contain the subwords for cases if a single deletion occurs in the first segment, if a single deletion occurs in the second segment or if a deletion occurs in the first and in the second segment.

Example: The subsets from the previous example are used. The number of possible codewords is $N = l_1 l_2 (l_1 - 1)!(l_1 - 2)! = 36$. The length of the binary codewords which will be mapped to the permutations is $v = \lfloor \log_2 N \rfloor = 5$. The number of subsets which will be used is $L = \lceil \frac{2^v}{(l_1-1)(l_2-1)} \rceil = 8$, thus only 8 of the possible 9 subsets will be used. A Subsequence Lookup Table is constructed for each of the 8 subsets.

5.5.3 Decoding

Let the received sequence be $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K, \mathbf{y}_{K+1})$, where each $\mathbf{y}_i$ is divided into two segments: the first consisting of symbols from $\mathcal{L}_1$ and the second consisting of symbols from $\mathcal{L}_2$. The decoding procedure consists of the following steps:

1. Identify the boundaries of the subwords. A subword, $\mathbf{y}_k$, is defined as one or more consecutive symbols from $\mathcal{L}_1$ followed by one or more consecutive symbols from $\mathcal{L}_2$.

2. If all $\mathbf{y}_k$ are in $\mathcal{R}$ and $\sum_{k=1}^{K+1} \psi(\mathbf{y}_k) \equiv 0 \pmod{L}$, then set $\mathbf{x}' = \mathbf{y}$ and go to Step 5, else go to Step 3.

3. If there exists an e such that (i) $\mathbf{y}_e$ is not in $\mathcal{R}$, (ii) $\mathbf{y}_k$ is in $\mathcal{R}$ for all $k \neq e$, and (iii) $\mathbf{y}_e$ has length equal to $M-1$ or $M-2$, then proceed to Step 4. Else, proceed to Step 6.

4. Split $\mathbf{y}_e$ into two segments, i.e. $\mathbf{y}_e = (\mathbf{y}_e^{\text{head}}, \mathbf{y}_e^{\text{tail}})$, where the symbols from $\mathbf{y}_e^{\text{head}} \in \mathcal{L}_1$ and the symbols from $\mathbf{y}_e^{\text{tail}} \in \mathcal{L}_2$. If the length of $\mathbf{y}_e^{\text{head}} < (l_1 - 1)$ or the length of $\mathbf{y}_e^{\text{tail}} < (l_2 - 1)$, then proceed to Step 6. Otherwise, calculate $c = -\sum_{k=1}^{K+1} \psi(\mathbf{x}_k) \mod L$ for all $k \neq e$. Subword $\mathbf{y}_e$ thus belongs to $\mathcal{R}_c$. Let $\mathbf{x}' = \mathbf{y}$ where $\mathbf{x}'_e$ is determined by using the Subsequence Lookup Table for $\mathcal{R}_c$ and proceed to Step 5.

5. Set the decoder output as $\mathbf{u}' = (\mathbf{u}'_1, \mathbf{u}'_2, \ldots, \mathbf{u}'_K)$, where $\mathbf{u}'_k = \phi^{-1}(\mathbf{x}'_k)$ for $k = 1, 2, \ldots, K$ and STOP.

6. Detect that the correction capability of the codeword has been exceeded and STOP.

5.5.4 Example

This example continues from the example given in Section 5.5.2. Let $K = 3$ and the encoded sequence be $\mathbf{x} = (123465, 231564, 213546, 132546)$. Four cases are considered for the received sequence $\mathbf{y}$ and the corresponding decoding results are given.

- If $\mathbf{y} = (123465, 231562, 135461, 32546)$, i.e. an error occurred in position 12, then in Step 1 the boundaries are identified as $\mathbf{x} = (123465, 23156, 213546, 132546)$ and thus the subword 23156 is not in $\mathcal{R}$. In Step 3, $e = 2$, all other subwords are in $\mathcal{R}$ and $|y_e| = M - 1 = 5$, thus we proceed to Step 4. Since $|y_2^{\text{head}}| = l_1$ and $|y_2^{\text{tail}}| = l_2 - 1$, R_c is calculated to be 4. Using the Subsequence Lookup Table, 23156 is a subword of 231564 and thus $x' = y$ where $x_2 = 231564$.

- If $\mathbf{y} = (123465, 215621, 354613, 2546)$, i.e. an error occurred in positions 8 and 12, then in Step 1 the boundaries are identified as $\mathbf{x} = (123465, 2156, 213546, 132546)$ and thus the subword 2156 is not in $\mathcal{R}$. In Step 3, $e = 2$, all other subwords are in $\mathcal{R}$ and $|y_e| = M - 2 = 4$, thus we proceed to Step 4. Since $|y_2^{\text{head}}| = l_1 - 1$ and $|y_2^{\text{tail}}| = l_2 - 1$, R_c is calculated to be 4. Using the Subsequence Lookup Table, 2156 is a subword of 231564 and thus $x' = y$ where $x_2 = 231564$.

- If $\mathbf{y} = (123465, 231521, 354613, 2546)$, i.e. an error occurred in positions 11 and 12, then in Step 1 the boundaries are identified as $\mathbf{x} = (123465, 2315, 213546, 132546)$ and thus the subword 2315 is not in $\mathcal{R}$. In Step 3, $e = 2$, all other subwords are in $\mathcal{R}$ and $|y_e| = M - 2 = 4$, thus we proceed to Step 4. Since $|y_2^{\text{head}}| = l_1$ but $|y_2^{\text{tail}}| = l_2 - 2$, we proceed to Step 6 and conclude that the correction capability of the code has been exceeded.

- If $\mathbf{y} = (123465, 242135, 461325, 46)$, i.e. errors occurred in positions 8 to 11, then in Step 1 the boundaries are identified as $\mathbf{x} = (123465, 24, 213546, 132546)$ and thus the subword 24 is not in $\mathcal{R}$. In Step 3, $e = 2$, all other subwords are in $\mathcal{R}$ but $|y_e| = M - 4 = 2$, thus we proceed to Step 6 and conclude that the correction capability of the code has been exceeded.

5.5.5 Code Rate

The code rate is given by

$$R = \frac{\lfloor \log_2 l_1 l_2 (l_1 - 1)!(l_2 - 1)! \rfloor K}{M(K+1)} \text{ bits/symbol.} \tag{5.13}$$

This can be improved, by encoding $\log_2(l_1-1)!(l_2-1)!$ extra information bits into the choice of $\mathbf{x}_{K+1}$:

$$R = \frac{\lfloor \log_2 l_1 l_2 (l_1-1)!(l_2-1)! \rfloor K + \log_2(l_1-1)!(l_2-1)!}{M(K+1)} \text{ bits/symbol.} \tag{5.14}$$

Figure 5.2 shows a comparison between the concatenated resynchronizable code rates, presented in this section, and the resynchronizable code rates from Section 3.2. For the concatenated codes, $K = 3$. Similar to Section 3.3.3, the lines for $l_1 = 3$ and $l_1 = l_2$ are identical for low values of M and then diverge for larger values of M. For $M = 6$, $l_1 = l_2 = 3$, resulting in the same code rate initially for the $l_1 = 3$ and the $l_1 = l_2$ line. For $M = 7$, 8 and 9, the output of the term $\lfloor \log_2 l_1 l_2 (l_1-1)!(l_2-1)! \rfloor$ is the same for both the $l_1 = 3$ and the $l_1 = l_2$ line. Only after $M = 9$, different code rates are obtained.

Using the concatenated scheme improves the cardinality of the codebook and thus the code rate. The error correction capability, however, is decreased. In the construction from Section 3.2, one deletion error could be corrected per segment, for every codeword. If we concatenate the codewords, only one subword may contain errors, and then again only one deletion per segment.

Comparing it to Construction II from Section 5.3, with $K = 3$ for all scenarios, the resynchronizable concatenated code has a lower code rate (see Figure 5.3). The resynchronizable concatenated code can correct 2 deletion errors, if the errors occur in the same subword but different segments. Construction II can only correct 1 deletion error but, if no deletion errors occur, can correct substitution errors.

5.6 Conclusion

The construction presented in Chapter 4, referred to as Construction I, is adapted to correct either deletion or substitution errors. The new construction, called Construction II, makes use of two substrings. The substrings

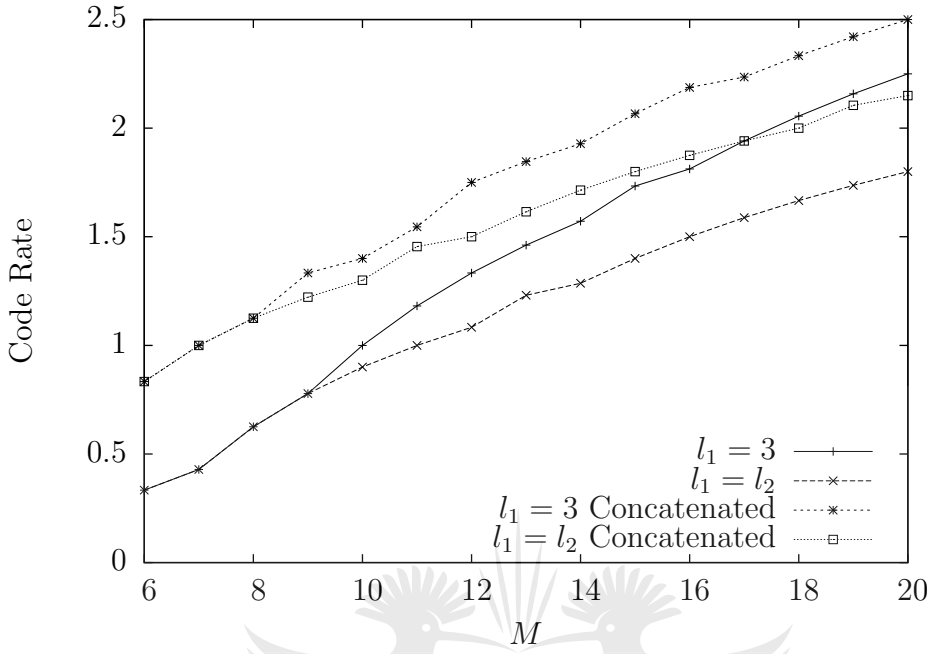

Figure 5.2: Comparison between Resynchronizable Concatenated Codes and Resynchronizable Codes.

contain permutations from different sets, either $\mathcal{R}$ or $\mathcal{T}$. The permutations in set $\mathcal{T}$ is constructed in such a way that each permutation starts with a specific sequence from a number of allowed start sequences. It is thus possible to detect the position of a deletion error by observing the positions of these special start sequences. Construction II is also generalized for all values of $M \geq 4$.

The decoding algorithm of Construction II can also be adapted to correct insertion errors. Since the codewords are permutations, the permutations with a specific start sequence will also have a specific end sequence. Instead of using the start sequences, the end sequences are used to determine the position of an insertion error. Construction II can thus correct a deletion error or an insertion error per codeword or a substitution per substring.

The resynchronizable codebooks from Chapter 3 can also be concatenated. It is thus not necessary to use two substrings since the structure of the codewords ensure that it is resynchronizable. This resynchronizable, concatenated scheme has a better code rate than the codes in Chapter 3 but a lower code rate than Construction II. However, it can be used to correct

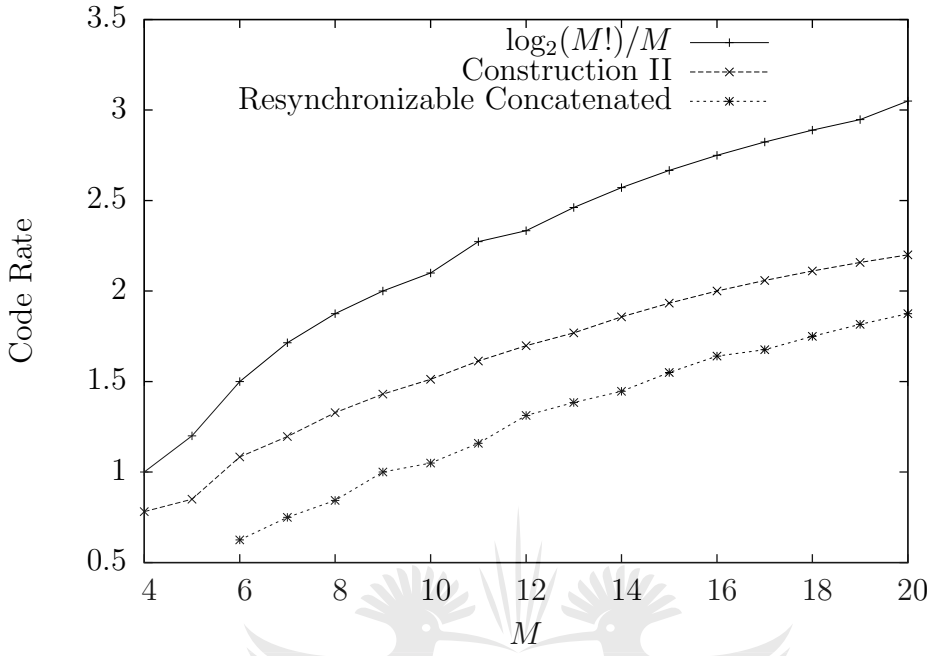

Figure 5.3: Comparison between Resynchronizable Concatenated Codes and Construction II

2 deletion errors, as long as these errors are in the same subword but in different segments.

Chapter 6

Concatenated Codes to Correct Transpositions Errors

6.1 Introduction

In this chapter, transposition errors are considered. In the previous chapters, the permutation property that every symbol occurs exactly once, was used to detect erroneous subwords. The parity permutation was then used to correct the error. However, if a transposition error occurs, the resulting subword is still a permutation. Detecting transposition errors are thus not as simple and more parity permutations are needed to do detection and correction.

A concatenation scheme is proposed based on Hamming codes. Additional parity permutations are added in accordance with the Hamming encoding scheme. The decoding algorithm then uses these parity permutations to detect the erroneous subword. The construction is firstly described for all values of M and a $(7, 4)$ Hamming code. The construction is then generalized for implementation with any (n, k) Hamming code to achieve better code rates. It is also shown that multiple adjacent transpositions or substitutions can be corrected.

The work presented in this chapter, with the exception of Section 6.4, has been published in [96].

6.2 Code Construction

In previous chapters, the permutation property of every symbol occurring only once was used to detect substitution errors. The parity permutation was only used to correct the already identified erroneous subword. However, if a transposition error occurs in a subword, the resulting subword is still a permutation. Hence, we cannot use the permutation property to detect errors. More parity permutations will thus have to be added to the codeword to assist with the detection and correction of the error. To illustrate the concept, a $(7, 4)$ codeword will be used based on Hamming codes. Thus, every codeword will consist of 4 permutations which have binary bits mapped to it, and 3 parity permutations. An example is shown in Figure 6.1.

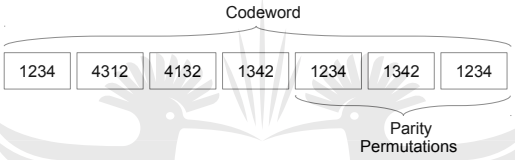

Figure 6.1: Concatenation of permutations to form a (7,4) codeword ($M = 4$)

Firstly, the set S_M is divided into subsets. For M symbols, $S_M = M!$, which is divided into $(M-1)!$ subsets, each containing M permutations with a Hamming distance of M. A permutation, with its $(M-1)$ circular shifts will form a subset.

The maximum length, v, of binary codewords which will be mapped to the permutations is

$$v = \lfloor \log_2 M! \rfloor. \tag{6.1}$$

The number of permutations of S_M which will be used is thus 2^v. The minimum number of subsets which is needed, L, is

$$L = \left\lceil \frac{2^v}{M} \right\rceil. \tag{6.2}$$

If $2^v \mod M \equiv 0$, then all these subsets will contain M permutations which are used, otherwise only $2^v \mod M$ permutations of the last subset will be used.

For example, let $M = 5$: binary codewords of length 6 will be mapped to the permutations. Only $2^6 = 64$ permutations are needed. Thus, of the $\frac{120}{5} = 24$ subsets only $L = \lceil \frac{64}{5} \rceil = 13$ will be used. Only 4 permutations of the last subset will be used. The possible subsets are:

$$\mathcal{R}_0 = \{12345, 23451, 34512, 45123, 51234\},$$

$$\mathcal{R}_1 = \{12354, 23541, 35412, 54123, 41235\},$$

$$\mathcal{R}_2 = \{12435, 24351, 43512, 35124, 51243\},$$

$$\mathcal{R}_3 = \{12453, 24531, 45312, 53124, 31245\},$$

$$\mathcal{R}_4 = \{12534, 25341, 53412, 34125, 41253\},$$

$$\mathcal{R}_5 = \{12543, 25431, 54312, 43125, 31254\},$$

$$\mathcal{R}_6 = \{13245, 32451, 24513, 45132, 51324\},$$

$$\mathcal{R}_7 = \{13254, 32541, 25413, 54132, 41325\},$$

$$\mathcal{R}_8 = \{13425, 34251, 42513, 25134, 51342\},$$

$$\mathcal{R}_9 = \{13452, 34521, 45213, 52134, 21345\},$$

$$\mathcal{R}_{10} = \{13524, 35241, 52413, 24135, 41352\},$$

$$\mathcal{R}_{11} = \{13542, 35421, 54213, 42135, 21354\},$$

$$\mathcal{R}_{12} = \{14235, 42351, 23514, 35142, 51423\}.$$

For all permutation sequences $\mathbf{r} \in \mathcal{R}_i$, define $\psi(\mathbf{r}) = i$. Let $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_{L-1}$. Let ϕ be a one-to-one mapping from the set $\mathcal{B}_v$ to $\mathcal{R}$.

6.2.1 Encoding

A source generates a sequence $\mathbf{u}$ of $(4v)$ bits, which is partitioned into 4 sequences $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ and $\mathbf{u}_4$, all from $\mathcal{B}_v$. Let $\mathbf{x}_l = \phi(\mathbf{u}_l)$ for all l and let

$$c_1 \equiv -(\psi(\mathbf{x}_1) + \psi(\mathbf{x}_2) + \psi(\mathbf{x}_3)) \mod L, \tag{6.3}$$

$$c_2 \equiv -(\psi(\mathbf{x}_1) + \psi(\mathbf{x}_3) + \psi(\mathbf{x}_4)) \mod L, \tag{6.4}$$

$$c_3 \equiv -(\psi(\mathbf{x}_1) + \psi(\mathbf{x}_2) + \psi(\mathbf{x}_4)) \mod L. \tag{6.5}$$

Then the encoder output is the code sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7)$, where the check sequence $\mathbf{x}_5$ is taken from $\mathcal{R}_{c_1}$, $\mathbf{x}_6$ is taken from $\mathcal{R}_{c_2}$, and $\mathbf{x}_7$ is taken from $\mathcal{R}_{c_3}$.

6.2.2 Decoding

Let the received sequence be $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_7)$, where each $\mathbf{y}_i$ is a sequence of M symbols from $\{1, 2, \ldots, M\}$. The decoding procedure consists of the following steps:

1. If at least one $\mathbf{y}_l$ is not in $\mathcal{R}$, then go to Step 2. Else calculate the syndrome values:

$$s_1 \equiv (\psi(\mathbf{y}_1) + \psi(\mathbf{y}_2) + \psi(\mathbf{y}_3) + \psi(\mathbf{y}_5)) \mod L, \tag{6.6}$$

$$s_2 \equiv (\psi(\mathbf{y}_1) + \psi(\mathbf{y}_3) + \psi(\mathbf{y}_4) + \psi(\mathbf{y}_6)) \mod L, \tag{6.7}$$

$$s_3 \equiv (\psi(\mathbf{y}_1) + \psi(\mathbf{y}_2) + \psi(\mathbf{y}_4) + \psi(\mathbf{y}_7)) \mod L. \tag{6.8}$$

   If $s_1 = 0$, $s_2 = 0$ and $s_3 = 0$, then set $\mathbf{a}' = \mathbf{y}$ and go to Step 5, else go to Step 3.

2. If there exists an e such that $\mathbf{y}_e$ is not in $\mathcal{R}$ and $\mathbf{y}_l$ is in $\mathcal{R}$ for all $l \neq e$, then go to Step 4. Else, go to Step 6.

3. Identify the erroneous subword, $\mathbf{y}_e$, by using Table 6.1. If $e = 5, 6, 7$, the error occurred in the parity permutations, set $\mathbf{a}' = \mathbf{y}$ and go to Step 5. Else, go to the next step.

4. For $j = 1, 2, \ldots, (M-1)$ set $\mathbf{a}^j$ as $\mathbf{y}$, with symbols at positions j and $(j+1)$ swapped in $\mathbf{y}_e$. Calculate s_1, s_2 and s_3. If $s_1 = 0$, $s_2 = 0$ and $s_3 = 0$, then set $\mathbf{a}' = \mathbf{a}^j$ for this j and go to Step 5. Else, go to Step 6.

5. Set the decoder output as $\mathbf{u}' = (\mathbf{u}'_1, \mathbf{u}'_2, \mathbf{u}'_3, \mathbf{u}'_4)$, where $\mathbf{u}'_l = \phi^{-1}(\mathbf{a}'_l)$ and STOP.

6. Detect that more than one error occurred and STOP.

Table 6.1: Error values

s_1	s_2	s_3	e
$= 0$	$= 0$	$\neq 0$	7
$= 0$	$\neq 0$	$= 0$	6
$= 0$	$\neq 0$	$\neq 0$	4
$\neq 0$	$= 0$	$= 0$	5
$\neq 0$	$= 0$	$\neq 0$	2
$\neq 0$	$\neq 0$	$= 0$	3
$\neq 0$	$\neq 0$	$\neq 0$	1

6.2.3 Code Rate

The code rate is given by:

$$R = \frac{4\lfloor \log_2 M! \rfloor}{7M} \text{ bits/symbol.} \tag{6.9}$$

Additionally, $\lfloor \log_2 M \rfloor$ additional information bits can be encoded into the choice of $\mathbf{x_5}$, $\mathbf{x_6}$ and $\mathbf{x_7}$. The code rate is then improved to:

$$R = \frac{4\lfloor \log_2 M! \rfloor + 3\lfloor \log_2 M \rfloor}{7M} \text{ bits/symbol.} \tag{6.10}$$

6.2.4 Example

Let $M = 4$ and the information sequence be $\mathbf{u} = (0000, 1001, 0111, 0001)$. The following subsets will be used:

$$\mathcal{R}_0 = \{1234, 2341, 3412, 4123\},$$

$$\mathcal{R}_1 = \{1243, 2431, 4312, 3124\},$$

$$\mathcal{R}_2 = \{1324, 3241, 2413, 4132\},$$

$$\mathcal{R}_3 = \{1342, 3421, 4213, 2134\}.$$

The algorithm in Section 4.3.4 is used to map the binary data to the permutations. Then the encoded sequence is $\mathbf{x} = (1234, 4132, 2431, 4123, 1243, 1342, 1324)$. Five cases are considered for the received sequence $\mathbf{y}$ and the corresponding decoding results are given.

- If $\mathbf{y} = \mathbf{x}$ (no errors), then we go from Step 1 to Step 5 and the decoding result is $\mathbf{u}' = \mathbf{u}$.

- If $\mathbf{y} = (1234, 1432, 2431, 4123, 1243, 1342, 1324)$, i.e. the symbol at position 5 was swapped with the symbol at position 6, then we find in Step 1 that $\mathbf{y}_2$ is not in $\mathcal{R}$. Thus, in Step 2 we find that $e = 2$ and proceed to Step 4. For $\mathbf{a}_2^1 = 4132$, we find $s_1 = 0$, $s_2 = 0$, $s_3 = 0$. For $\mathbf{a}_2^2 = 1342$, we find $s_1 \neq 0$ and $s_3 \neq 0$ and $\mathbf{a}_2^3 = 1423$ is not in $\mathcal{R}$. In conclusion, $\mathbf{a}' = \mathbf{a}^1 = (1234, 4132, 2431, 4123, 1243, 1342, 1324)$ and Step 5 gives $\mathbf{u}' = \mathbf{u}$. Hence, the error has been corrected.

- If $\mathbf{y} = (1234, 1432, 4231, 4123, 1243, 1342, 1324)$, i.e., the symbols at positions 5 and 6 were swapped as well as the symbols at positions 9 and 10, then we find that $\mathbf{y}_2$ and $\mathbf{y}_3$ are not in $\mathcal{R}$, and thus we go from Step 1 via Step 2 to Step 6, and the decoding result is the detection of (at least) two errors.

- If $\mathbf{y} = (1234, 4123, 2431, 4213, 1243, 1342, 1324)$, i.e., the symbols at positions 7 and 8 are swapped as well as symbols at positions 14 and 15, then we find in Step 1 that all subwords are in $\mathcal{R}$, but that $s_1 \neq 0$, $s_2 \neq 0$, $s_3 \neq 0$ and thus in Step 3 we find that $e = 1$ and proceed to Step 4. For $\mathbf{a}_1^1 = 2134$, we find $s_1 \neq 0$ and $s_2 \neq 0$. For $\mathbf{a}_1^2 = 1324$, we find $s_2 \neq 0$ and $s_3 \neq 0$. For $\mathbf{a}_1^3 = 1243$, we find $s_1 \neq 0$ and $s_3 \neq 0$. In conclusion, we end up in Step 6, and the decoding result is the detection of (at least) two errors.

- If $\mathbf{y} = (1234, 4132, 2431, 4132, 1243, 1342, 1324)$, i.e., the symbols at positions 15 and 16 are swapped, then we find in Step 1 that all subwords are in $\mathcal{R}$, but that $s_2 \neq 0$ and $s_3 \neq 0$. In Step 3 we find that $e = 4$ and proceed to Step 4. $\mathbf{a}_4^1 = 1432$, which is not in $\mathcal{R}$. For $\mathbf{a}_4^2 = 4312$, we find $s_2 \neq 0$ and $s_3 \neq 0$. For $\mathbf{a}_4^3 = 4123$, we find $s_1 = 0$, $s_2 = 0$, $s_3 = 0$. Thus, $\mathbf{a}' = \mathbf{a}^3 = (1234, 4132, 2431, 4123, 1243, 1342, 1324)$ and Step 5 gives $\mathbf{u}' = \mathbf{u}$.

6.3 Generalizing the Outer Hamming Code

In the previous section, a $(7, 4)$ Hamming code was used to illustrate the code construction. However, any (n, k) Hamming code can be used. Using a Hamming code with a better code rate will also improve the code rate of the construction in the previous section. On the other hand, the outer code is still only a single error correction code. So even if more subwords are used per codeword, only one erroneous subword per codeword can be identified and corrected. When generalizing the Hamming code, the parity permutations will be added between the other permutations, and not at the end of the codeword as in the previous section.

The set $\mathcal{S}_M$ will be partitioned as in the previous section. The mapping from the binary subwords to the permutation subwords will also be identical. An (n, k) Hamming code will be used as outer code, thus the codeword will consist of n permutation subwords of which k contain information. Let m be the number of parity subwords, then $n = 2^m - 1$ and $k = n - m$.

6.3.1 Encoding

A source generates a sequence $\mathbf{u}$ of (kv) bits, which is partitioned into k sequences $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$, all from $\mathcal{B}_v$. Let $\mathbf{z}_j = \phi(\mathbf{u}_j)$.

Let i and j be positive integers and let $\sigma(j)$ represent the transformation of the decimal representation of j to the binary representation of j. The function $\sigma(j) \wedge \sigma(i)$ is the bitwise AND operation of the two binary numbers. For example, let $j = 1$ and $i = 3$, then $\sigma(j) \wedge \sigma(i) = 01$.

1. The parity permutations are inserted into the sequence $\mathbf{x}$ at the positions where i is a power of 2. Firstly, the permutations which are not parity permutations are inserted into $\mathbf{x}$. Let $j = 1$ and $i = 1$. If i is a power of 2, then $i = i + 1$, else $\mathbf{x}_i = \mathbf{z}_j$, $j = j + 1$ and $i = i + 1$. Repeat while $i \leq n$.

2. Next, the subset number of each parity permutation is calculated. Let $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m)$ be the set of parity permutations, $j = 1, 2, \ldots, m$ and $\mathbf{x}_{2^{j-1}} = \mathbf{p}_j$. For i from 1 to n for each $\mathbf{p}_j$ and $c_j = 0$: If $2^{j-1} \neq i$ and $\sigma(2^{j-1}) \wedge \sigma(i) \neq \mathbf{0}$, then

$$c_j = -(c_j + \psi(\mathbf{x}_i)) \mod L. \tag{6.11}$$

   The parity permutation $\mathbf{p}_j$ is then taken from $\mathcal{R}_{c_j}$.

The encoder output is the code sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$.

6.3.2 Decoding

Let the received sequence be $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$, where each $\mathbf{y}_i$ is a sequence of M symbols from $\{1, 2, \ldots, M\}$. The decoding procedure consists of the following steps:

1. If at least one $\mathbf{y}_l$ is not in $\mathcal{R}$, then go to Step 2. Else, let $j = 1, 2, \ldots, m$. For i from 1 to n for each $\mathbf{p}_j$ and $s_j = 0$: if $\sigma(2^{j-1}) \wedge \sigma(i) \neq \mathbf{0}$, then

$$s_j = s_j + \psi(\mathbf{y}_i) \mod L. \tag{6.12}$$

   If $s_j \neq 0$, then set $s_j = 1$.

   If $s_j = 0$ for all values of j, then set $\mathbf{a}' = \mathbf{y}$ and go to Step 5, else go to Step 3.

2. If there exists an e such that $\mathbf{y}_e$ is not in $\mathcal{R}$ and $\mathbf{y}_l$ is in $\mathcal{R}$ for all $l \neq e$, then go to Step 4. Else, go to Step 6.

3. A syndrome $\mathbf{s}$ can now be constructed where s_1 is the least significant bit and s_m the most significant bit. The decimal value of $\mathbf{s}$ is the position of the erroneous subword $\mathbf{y}_e$. If e is a power of 2, the error occurred in a parity permutation, set $\mathbf{a}' = \mathbf{y}$ and go to Step 5. Else, go to the next step.

4. For $j = 1, 2, \ldots, (M-1)$ set $\mathbf{a}^j$ as $\mathbf{y}$, with symbols at positions j and $(j+1)$ swapped in $\mathbf{y}_e$. Calculate $\mathbf{s}$. If $\mathbf{s} = 0$ then set $\mathbf{a}' = \mathbf{x}^j$ for this j and go to Step 5. Else, go to Step 6.
5. Remove the parity permutations from $\mathbf{a}'$ and set the decoder output as $\mathbf{u}' = (\mathbf{u}'_1, \mathbf{u}'_2, \ldots, \mathbf{u}'_k)$, where $\mathbf{u}'_j = \phi^{-1}(\mathbf{a}'_j)$ and STOP.
6. Detect that more than one error occurred and STOP.

6.3.3 Code Rate

The code rate is given by:

$$R = \frac{k\lfloor \log_2 M! \rfloor}{nM} \text{ bits/symbol.} \tag{6.13}$$

Additionally, $\lfloor \log_2 M \rfloor$ additional information bits can be encoded into the choice of the parity permutations. The code rate is then improved to:

$$R = \frac{k\lfloor \log_2 M! \rfloor + m\lfloor \log_2 M \rfloor}{nM} \text{ bits/symbol.} \tag{6.14}$$

6.3.4 Example

Let $M = 4$ and the information sequence be $\mathbf{u} = (0000, 1001, 0111, 0001, 0000, 1001, 0111, 0001, 1111, 1010, 1001)$. The same subsets will be used as in the previous example. Let $m = 4$, $n = 15$ and $k = 11$. The algorithm in Chapter 4 is used to map the binary data to the permutations. Calculating the sets for the parity permutations: $c_1 = 2$, $c_2 = 0$, $c_3 = 2$ and $c_4 = 2$. Then the encoded sequence is $\mathbf{x} = (\mathbf{1324}, \mathbf{1234}, 1234, \mathbf{1324}, 4132, 2431, 4123, \mathbf{1324}, 1234, 4132, 2431, 4123, 3421, 2413, 4132)$, where the parity permutations are shown in bold.

Only one case is considered since it is similar to the scenarios in the previous example. If $\mathbf{y} = (1324, 1234, 1243, 1324, 4132, 2431, 4123, 1324, 1234, 4132, 2431, 4123, 3421, 2413, 4132)$, i.e., the symbols at positions 11 and 12 are swapped, then we find in Step 1 that all subwords are in $\mathcal{R}$, but that $s_1 \neq 0$ and $s_2 \neq 0$ and thus in Step 3 we find that the syndrome is 0011 and $e = 3$. We proceed to Step 4. Both $\mathbf{a}_1^1 = 2143$ and $\mathbf{a}_1^2 = 1423$ are not in $\mathcal{R}$. For $\mathbf{a}_1^3 = 1234$, the syndrome value $\mathbf{s} = \mathbf{0}$ and is thus the correct permutation.

6.4 Correcting Multiple Transposition and Substitution Errors

The distance between permutations in a subset is M. For higher values of M, the increased distance can be used to correct more transposition errors in an erroneous subword or to correct substitution errors. By adapting the algorithm in Section 6.3.2, combinations of transpositions and substitutions can be corrected.

An adjacent transposition error can be modelled as two substitution errors. For example, let (12345) $\rightarrow$ (12435), thus symbol 3 and 4 are transposed. This is equivalent to symbol 3 being substituted with symbol 4 and symbol 4 being substituted with symbol 3, thus two substitution errors. It is known that if $2t + 1 \leq d$, where t is the number of substitution errors and d the Hamming distance, then the substitution errors can be corrected. Thus, if the Hamming distance is M, then $\lfloor \frac{M-1}{2} \rfloor$ substitution errors can be corrected or $\lfloor \frac{M-1}{4} \rfloor$ adjacent transposition errors.

The same decoding steps are followed as in Section 6.3.2, except Step 4 is replaced by the following step:

4. Choose any $s_j = 1$. Let $i = 1, 2, \ldots, m$ and $x' = 0$: if $\sigma(2^{i-1}) \wedge \sigma(j) \neq \mathbf{0}$, then

$$x' = -(x' + \psi(\mathbf{y}_i)) \mod L. \tag{6.15}$$

Thus, $\psi(\mathbf{y}_e)$ should be x'. Calculate the Hamming distance between $\mathbf{y}_e$ and all the permutations in subset x'. Set $\mathbf{a}' = \mathbf{y}$ where $\mathbf{a}_e$ is the permutation in $\mathcal{R}$ with the smallest Hamming distance to $\mathbf{y}_e$.

6.5 Comparison to Previous Work

More parity permutations are added compared to the construction in Chapter 4, to be able to correct adjacent transposition errors. Figure 6.2 shows the code rates for different constructions and different values of M and different outer codes (Equation 6.14 is used). The line marked "Concatenated" is the

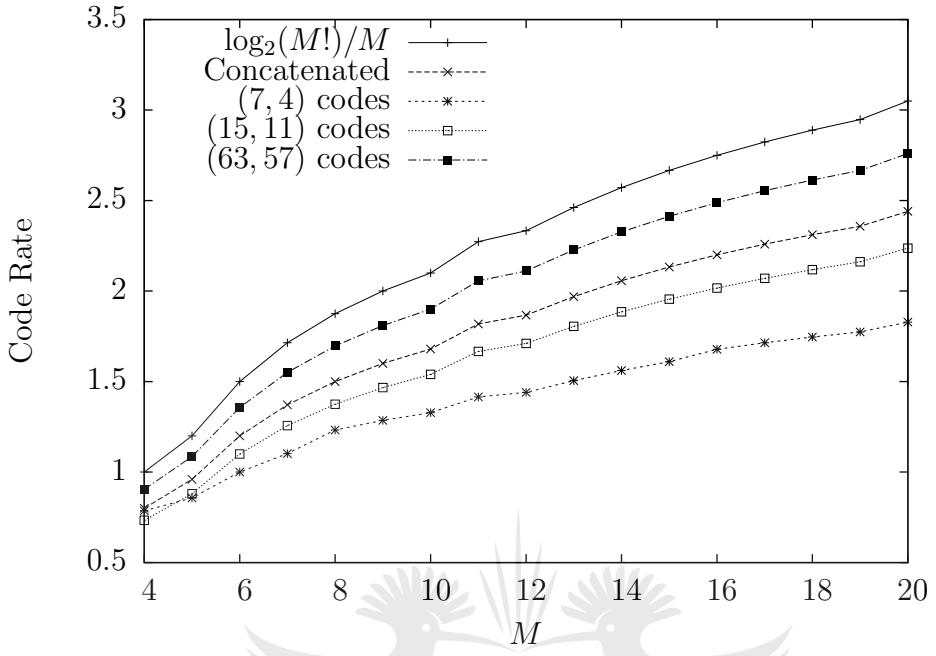

Figure 6.2: Code rates for different codes

construction presented in Chapter 4 with a value of $K = 4$, thus the same number as information permutations as in the $(7, 4)$ Hamming code case. For every $K = 4$ information permutations, one parity permutation is added.

However, it is important to note that the constructions in Figure 6.2 have different properties with regards to their error correction capabilities. Codes with the optimal code rate, $\log_2(M!)/M$, do not have any error-correction capabilities. The concatenated code with $K = 4$, can correct one substitution error in every $K + 1 = 5$ permutations or detect two substitution errors. The codes presented in this chapter can correct a single adjacent transposition error per codeword. Other work in literature, for example in [35] and [90], can correct errors in every permutation. The concatenated coding scheme is thus suited for environments with low error probabilities which benefits from higher code rates.

6.6 Conclusion

The construction in Chapter 4 is adapted to correct single adjacent transposition errors. A number of permutations is concatenated and parity per-

mutations are added to form a codeword. A construction is presented with an outer code based on $(7, 4)$ Hamming codes. The outer code can identify which permutation is erroneous. The permutation property in combination with the set construction are then used to correct the error. This construction is then generalized for any (n, k) Hamming code. These constructions make use of a larger subset of S_M, thus leading to higher cardinalities and higher code rates. The code rates can be further increased by using Hamming codes with higher code rates as outer codes.

The disadvantage of this construction is that only one transposition error is corrected per codeword. An adaptation to the decoding algorithm is presented which can correct multiple transpositions and substitutions, however, only if the errors are limited to one subword. Using a different outer code can increase the error correction capability of the codeword. Then, the number of erroneous subwords which can be detected depends on the outer code. A number of transposition and substitution errors can then be corrected per erroneous subword.

Chapter 7

Conclusion

7.1 Overview

The main focus of this thesis is taking a segmented or concatenated approach to construct longer codewords in order to improve the code rate and cardinalities of the codebooks. Concatenated permutations can also be viewed as a subset of multipermutations. Multipermutations, as discussed in Chapter 2, are permutations where symbols appear a fixed number of times in every codeword. In the case of concatenated permutations, if a codeword consists of K permutations, then every symbol will appear exactly K times in the codeword. However, an additional restriction is that, if the codeword is divided into blocks of length M, every symbol can occur only once per block.

Resynchronizable permutation codes were presented first in Chapter 3. The permutations were constructed using two segments. Certain symbols could only appear in the first segment and the others in the last segment. Furthermore, every segment was constructed to be able to correct one deletion error. This construction enables the decoder to detect if deletion errors occurred and to correct them. Since substitution errors are usually more probable than deletion errors, the effect of substitution errors were also taken into account. The decoding algorithm was adapted to also detect substitution errors and, for certain error patterns, correct the substitution errors by deleting the erroneous symbols and then correct the deletion error. Even though these codes are able to not only resynchronize after an error occurred, but also to correct

the error, the code rates are quite low. These codes are revisited in Chapter 5, where it is shown that the code rate can be increased by concatenating the codes.

The idea of concatenating permutation codes was presented next. Instead of using a single permutation as a codeword, permutations are concatenated. Additional parity permutations are added for error-correction. Every permutation can thus be viewed as a symbol in the complete codeword. The complete set of permutations are divided into smaller subsets. Permutations in a single subset have a minimum Hamming distance of 4. Binary data is mapped onto the permutations. Initially, only one parity permutation was added. It was shown that these codewords will have a minimum distance of 4 and thus one substitution error can be corrected per codeword.

Next, the possibility of deletions was considered. The complete set of permutations was firstly divided into two sets, namely $\mathcal{R}$ and $\mathcal{T}$. The permutations in set $\mathcal{T}$ was constructed so that every permutation starts with one of the specifically reserved start sequences. None of the permutations in $\mathcal{R}$ is allowed to start with these reserved start sequences. The codeword is then constructed by using two substrings which are intertwined. All the permutations in one substring are from $\mathcal{R}$ and all the permutations from the other substring are from $\mathcal{T}$. Two parity permutations are added, one for each substring. It is thus possible to correct a substitution error per substring. If a deletion error occurs, the position of these reserved start sequences of the second substring can be used to determine the position of the deletion and a correction can be made.

It was shown that since permutation codes are used, if the codes in $\mathcal{T}$ are constructed to have a specifically reserved start sequence, then it will also have a specifically reserved end sequence. These end sequences can be used when an insertion error occurred to detect the position of the error and to make a correction. A deletion, insertion or substitution error can thus be corrected.

In order to also be able to correct adjacent transposition errors, an outer code was implemented where the parity permutations are added in accordance with the specific outer code which is used. A Hamming code is implemented for illustration purposes. It is then possible to correct multiple adjacent transposition and substitution errors, as long as the errors are limited to one subword. Implementing outer codes with stronger error-correction capabilities would be beneficial since more erroneous subwords can be identified and then adjacent transposition and substitution errors can be corrected in each of the erroneous subwords.

7.2 Further Research

Throughout this thesis, the Hamming distance has been used to correct various errors. Other distance measures can be implemented to correct errors. For example, if the subsets are constructed based on the Kendall τ distance or the Ulam distance, it may be possible to correct more transposition errors and even translocation errors. Since existing error-correcting codes are used as outer codes, the outer codes will still make use of Hamming distances. Additional benefits may be achieved by using one distance measure in the inner code and Hamming distance in the outer code.

Instead of using existing error-correction codes as the outer codes, one can also investigate other codes as the outer codes based on other distance measures. Using distance measures like the Lee distance or Levenshtein distance might make permutations attractive for use in other applications.

References

[1] I. T. Union. (2014) The world in 2014: ICT facts and figures. [Online]. Available: http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf

[2] D. E. Knuth, *The art of computer programming, Volume 4A, Combinatorial Algorithms, Part 1*, 2nd ed., Addison-Wesley, Ed., 2011.

[3] M. Deza and S. A. Vanstone, "Bounds on permutation arrays," *Journal of Statistical Planning and Inference*, vol. 2, no. 2, pp. 197–209, 1978.

[4] A. J. H. Vinck, "Coded modulation for powerline communications," *AEÜ International Journal of Electronics and Communications*, vol. 54, no. 1, pp. 45–49, Jan. 2000.

[5] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, June 2009.

[6] Z. Wang, A. Jiang, and J. Bruck, "On the capacity of bounded rank modulation for flash memories," in *Proceedings of the IEEE International Symposium on Information Theory*, Seoul, Korea, June 2009, pp. 1234–1238.

[7] E. En Gad, M. Langeberg, M. Schwartz, and J. Bruck, "Constant-weight Gray codes for local rank modulation," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7431–7442, July 2011.

[8] ——, "Generalized Gray codes for local rank modulation," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6664–6673, June 2013.

[9] M. Horovitz and T. Etzion, "Local rank modulation for flash memories," in *Proceedings of the IEEE Information Theory Workshop*, Hobart, Tasmania, Nov. 2014, pp. 606–610.

[10] M. Deza and H. Huang, "Metrics on permutations, a survey," *Journal of Combinatorics, Information and System Sciences*, vol. 23, pp. 173–185, 1998.

[11] T. Kløve, "Classification of permutation codes of length 6 and minimum distance 5," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, Nov. 2000, pp. 465–468.

[12] I. Janiszczak and R. Staszewski. An improved bound for permutation arrays of length 10. Institute for Experimental Mathematics, University of Duisburg-Essen, Germany. [Online]. Available: http://www.iem.uni-due.de/preprints/IJRS.pdf

[13] R. Montemanni, J. Barta, and D. H. Smith, "Permutation codes: a new upper bound for m(7,5)," in *Proceedings of the International conference on Informatics and Advanced Computing*, Bangkok, Thailand, Dec. 2014, pp. 1–4.

[14] H. C. Ferreira and A. J. H. Vinck, "Interference cancellation with permutation trellis codes," in *Proceedings of the IEEE Vehicular Technology Conference*, Boston, USA, Sept. 2000, pp. 2401–2407.

[15] T. G. Swart, I. de Beer, H. C. Ferreira, and A. J. H. Vinck, "Simulation results for permutation trellis codes using M-ary FSK," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Vancouver, Canada, April 2005, pp. 317–321.

[16] T. G. Swart, A. J. H. Vinck, and H. C. Ferreira, "Convolutional code search for good permutation trellis codes using M-ary FSK," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Pisa, Italy, March 2007, pp. 441–446.

[17] J.-C. Chang, R.-J. Chen, T. Kløve, and S.-C. Tsai, "Distance-preserving mappings from binary vectors to permutations," *IEEE Transactions on Information Theory*, vol. 49, no. 4, pp. 1054–1059, June 2003.

[18] J.-C. Chang, "Distance-increasing mappings from binary vectors to permutations," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 359–363, Jan. 2005.

[19] K. Lee, "New distance-preserving mappings of odd length," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2539–2543, Oct. 2004.

[20] T. G. Swart and H. C. Ferreira, "A generalized upper bound and a multilevel construction for distance-preserving mappings," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3685–3695, Aug. 2006.

[21] J.-S. Lin, J.-C. Chang, R.-J. Chen, and T. Kløve, "Distance-preserving and distance-increasing mappings from ternary vectors to permutations," *IEEE Transactions on Information Theory*, vol. 54, no. 3, pp. 1334–1339, March 2008.

[22] T.-T. Lin, S.-C. Tsai, and H.-L. Wu, "Simple distance-preserving mappings from ternary vectors to permutations," *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 3251–3256, July 2008.

[23] H. Chadwick and I. Reed, "The equivalence of rank permutation codes to a new class of binary codes," *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 640–641, Sept. 1970.

[24] H. C. Ferreira and A. J. H. Vinck, "Interference cancellation with permutation trellis codes," in *Proceedings of the IEEE Vehicular Technology Conference*, Boston, MA, USA, Sept. 2000, pp. 2401–2407.

[25] H. C. Ferreira, D. A. Wright, and A. L. Nel, "Hamming distance preserving mappings and trellis codes with constrained binary symbols," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 1098–1103, Sept. 1989.

[26] C. Ding, F.-W. Fu, T. Kløve, and V. K.-W. Wei, "Construction of permutation arrays," *IEEE Transactions on Information Theory*, vol. 48, no. 4, pp. 977–980, April 2002.

[27] F.-W. Fu and T. Kløve, "Two constructions of permutation arrays," *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 881–883, May 2004.

[28] C. J. Colbourn, T. Kløve, and A. C. H. Ling, "Permutation arrays for powerline communication and mutually orthogonal latin squares," *IEEE Transactions on Information Theory*, vol. 50, no. 6, pp. 1289–1291, June 2004.

[29] C. J. Colbourn and J. H. Dinitz, "Mutually orthogonal latin squares: A brief survey of constructions," *Journal of Statistical Planning and Inference*, vol. 95, no. 1-2, pp. 9–48, May 2001.

[30] W. Chu, C. J. Colbourn, and P. Dukes, "Constructions for permutation codes in powerline communications," *Designs, Codes and Cryptography*, vol. 32, no. 1-3, pp. 51–64, May 2004.

[31] M. G. Kendall, "A new measure of rank correlation," *Biometrica*, vol. 30, no. 12, pp. 81–93, June 1938.

[32] S. Buzaglo and T. Etzion, "Bounds on the size of permutation codes with the Kendall τ-metric," *IEEE Transactions on Information Theory*, vol. 61, no. 6, pp. 3241–3250, April 2015.

[33] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3158–3165, July 2010.

[34] F. Farnoud and O. Milenkovic, "Error-correction in flash memories via codes in the Ulam metric," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3003–3020, Jan. 2013.

[35] ——, "Multipermutation codes in the Ulam metric for nonvolatile memories," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 919–932, May 2014.

[36] M.-Z. Shieh and S.-C. Tsai, "Decoding frequency permutation arrays under Chebyshev distance," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5730–5737, Nov. 2010.

[37] T. Kløve, T.-T. Lin, and W.-G. Tzeng, "Permutation arrays under the Chebyshev distance," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2611–2617, June 2010.

[38] M.-Z. Shieh and S.-C. Tsai, "Computing the ball size of frequency permutations under Chebyshev distance," in *Proceedings of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, July 2011, pp. 2100–2104.

[39] A. Jiang, M. Schwartz, and J. Bruck, "Error-correcting codes for rank modulation," in *Proceedings of the IEEE International Symposium on Information Theory*, Toronto, Canada, July 2008, pp. 1736–1740.

[40] Y. M. Chee and V. K. Vu, "Breakpoint analysis and permutation codes in generalized Kendall tau and Cayley metrics," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2959–2963.

[41] L. Su, F. Farnoud, and O. Milenkovic, "Similarity distances between permutations," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2267–2271.

[42] A. Mazumdar, A. Barg, and G. Zemor, "Constructions of rank modulation codes," *IEEE Transactions on Information Theory*, vol. 56, no. 2, pp. 1018–1029, Oct. 2012.

[43] Z. Hongchao, M. Schwartz, A. Jiang, and J. Bruck, "Systematic error-correcting codes for rank modulation," *IEEE Transactions on Information Theory*, vol. 61, no. 1, pp. 17–32, Oct. 2015.

[44] S. Lin and D. J. Costello Jnr., *Error control coding*, 2nd ed., P. E. International, Ed., 2004.

[45] S. Buzaglo, E. Yaakobi, T. Etzion, and J. Bruck, "Systematic codes for rank modulation," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2386–2390.

[46] A. Jiang, M. Schwartz, and J. Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2112–2120, May 2010.

[47] F. F. Sellers, "Bit loss and gain correction code," *IEEE Transactions on Information Theory*, vol. 8, no. 1, pp. 35–38, Jan. 1962.

[48] E. N. Gilbert, "Synchronization of binary messages," *IEEE Transactions on Information Theory*, vol. 6, no. 4, pp. 470–477, Sept. 1960.

[49] T. Shongwe, T. G. Swart, H. C. Ferreira, and T. van Trung, "Good synchronization sequences for permutation codes," *IEEE Transactions on Communications*, vol. 60, no. 5, pp. 1204–1208, March 2012.

[50] M. C. Davey and D. J. C. MacKay, "Reliable communication over channels with insertions, deletions and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687–698, Feb. 2001.

[51] R. Heymann and H. C. Ferreira, "Regaining synchronization using neural networks and watermarks," in *Proceedings of the IEEE International Symposium on Information Theory and its Applications*, Auckland, New Zealand, Dec. 2008, pp. 1–6.

[52] S. W. Golomb, B. Gordon, and L. R. Welch, "Comma-free codes," *Canadian Journal of Mathematics*, vol. 10, no. 2, pp. 202–209, 1958.

[53] H. Morita, A. J. van Wijngaarden, and A. J. H. Vinck, "On the construction of maximal prefix-synchronized codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 2158–2166, Nov. 1996.

[54] W. L. Eastman, "On the construction of comma-free codes," *IEEE Transactions on Information Theory*, vol. 11, no. 2, pp. 263–267, Apr. 1965.

[55] R. A. Scholtz, "Codes with synchronization capability," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 135–142, Apr. 1966.

[56] C. V. Ramamoorthy and D. W. Tufts, "Reinforced prefixed comma-free codes," *IEEE Transactions on Information Theory*, vol. 13, no. 3, pp. 366–371, July 1967.

[57] V. I. Levenshtein, "On perfect codes in deletion and insertion metric," *Discrete Mathematics and its Applications*, vol. 2, no. 3, pp. 241–258, 1992.

[58] P. A. H. Bours, "On the construction of perfect deletion-correcting codes using design theory," *Designs, Codes and Cryptography*, vol. 6, no. 1, pp. 5–20, July 1995.

[59] L. Cheng, T. G. Swart, and H. C. Ferreira, "Synchronization using insertion/deletion correcting permutation codes," in *Proceedings of the IEEE International Symposium on Information Theory and its Applications*, Jeju Island, Korea, Apr. 2008, pp. 135–140.

[60] ——, "Re-synchronization of permutation codes with Viterbi-like decoding," in *Proceedings of the IEEE International Symposium on Information Theory and its Applications*, Dresden, Germany, March. 2009, pp. 36–40.

[61] R. Heymann and H. C. Ferreira, "Using tree structures to resynchronize permutation codes," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Rio de Janeiro, Brazil, March 2010, pp. 108–113.

[62] R. Heymann, T. G. Swart, and H. C. Ferreira, "Correcting adjacent errors using permutation code trees," in *Proceedings of the IEEE Africon*, Livingston, Zambia, Sept. 2011, pp. 1–5.

[63] R. Gabrys, E. Yaakobi, F. Farnoud, and J. Bruck, "Codes correcting erasures and deletions for rank modulation," in *IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2759–2763.

[64] R. Gabrys, E. Yaakobi, F. Farnoud, F. Sala, L. Dolocek, and J. Bruck, "Single-deletion-correcting codes over permutations," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2764–2768.

[65] F. Sala, R. Gabrys, and L. Dolocek, "Deletions in multipermutations," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2769 – 2773.

[66] F. G. Stremler, *Introduction to communication systems*, 3rd ed., Addison-Wesley, Ed., 1992.

[67] M. C. Bali and C. Rebai, "Optimum receiver of coded M-FSK modulation for power line communications," in *Proceedings of the IEEE Symposium on Computers and Communications*, Fungal, Portugal, June 2014, pp. 1–6.

[68] H. C. Ferreira, A. J. H. Vinck, T. G. Swart, and I. de Beer, "Permutation trellis codes," *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 1782–1789, Nov. 2005.

[69] D. J. J. Versfeld, A. J. H. Vinck, and H. C. Ferreira, "Reed-Solomon coding to enhance the reliability of M-FSK in a power line environment," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Vancouver, Canada, April 2005, pp. 100–104.

[70] D. J. J. Versfeld, A. J. H. Vinck, J. N. Ridley, and H. C. Ferreira, "Constructing coset codes with optimal same-symbol weight for detecting narrowband interference in M-FSK systems," *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6347–6353, Dec. 2010.

[71] B. Rajkumarsingh and K. Z. Heetun, "Turbo codes and FSK in power line communication," in *Proceedings of the IEEE Africon*, Pointe-Aux-Piments, Mauritius, Sept. 2013, pp. 1–5.

[72] Y. M. Chee, H. M. Kiah, P. Purkayastha, and C. Wang, "Importance of symbol equity in coded modulation for power line communications," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4381–4390, Sept. 2013.

[73] Y. M. Chee, H. M. Kiah, A. C. H. Ling, and C. Wang, "Optimal equitable symbol weight codes for power line communications," in *Proceedings of the IEEE International Symposium on Information Theory*, Cambridge, USA, July 2012, pp. 666–670.

[74] H. P. Alliance. (2001) Homeplug 1.0.1 specification. [Online]. Available: http://read.pudn.com/downloads114/ebook/479147/HOMEPLUG.pdf

[75] Y.-H. Kim, K.-H. Kim, H.-M. Oh, K.-H. Kim, and S.-C. Kim, "Mitigation of effect of impulsive noise for OFDM systems over power line channels," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Jeju City, Jeju Island, May 2008, pp. 386–390.

[76] H.-M. Oh, Y.-J. Park, J.-J. Lee, and K.-C. Whang, "Mitigation of performance degradation by impulsive noise in LDPC coded OFDM system," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Orlando, USA, March 2006, pp. 331–336.

[77] C. Hsu, N. Wang, W.-Y. Chan, and P. Jain, "Improving Homeplug power line communications with LDPC coded OFDM," in *Proceedings of the 28th Annual International Telecommunications Energy Conference*, Providence, Rhode Island, USA, Sept. 2006, pp. 1–7.

[78] A. Mengi and A. J. H. Vinck, "Impulsive noise error correction in 16-OFDM for narrowband power line communication," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Dresden, Germany, March 2009, pp. 31–35.

[79] V. N. Papilaya, T. Shongwe, A. Vinck, and H. C. Ferreira, "Selected subcarriers QPSK-OFDM transmission schemes to combat frequency disturbances," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Beijing, China, March 2012, pp. 200–205.

[80] H. H. Nguyen and T. Q. Bui, "Bit-interleaved coded OFDM with iterative decoding in impulsive noise," *IEEE Transactions on Power Delivery*, vol. 23, no. 2, pp. 640–649, April 2008.

[81] K. S. Al-Mawali and Z. M. Hussain, "Performance of bit-interleaved coded OFDM in power line communications with impulsive noise," in *Proceedings of the International Conference on Advanced Technologies for Communications*, Hai Pong, Vietnam, Oct. 2009, pp. 49–53.

[82] G. Ren, S. Qiao, H. Zhao, C. Li, and Y. Hei, "Mitigation of periodic impulsive noise in OFDM-based power-line communications," *IEEE Transactions on Power Delivery*, vol. 28, no. 2, pp. 825–834, Jan. 2013.

[83] L. Cheng, T. G. Swart, and H. C. Ferreira, "Adaptive rateless permutation coding scheme for OFDM-based PLC," in *Proceedings of the IEEE International Symposium on Powerline Communications and its Applications*, Johannesburg, South Africa, March 2013, pp. 242–246.

[84] K. Ogunyanda, A. D. Familua, T. G. Swart, and H. C. Ferreira, "Evaluation and implementation of cyclic permutation coding for power line communications," in *Proceedings of the IEEE 6th International Conference on Adaptive Science and Technology*, Ota, Nigeria, Oct. 2014, pp. 1–7.

[85] T. Wadayama and M. Hagiwara, "LP-decodable permutation codes based on linearly constrained permutation matrices," *IEEE Transactions on Information Theory*, vol. 58, no. 8, pp. 5454–5470, 2012.

[86] A. Berman and Y. Birk, "Constrained flash memory programming," in *Proceedings of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, July 2011, pp. 2128–2132.

[87] F. Sala and L. Dolocek, "Constrained rank modulation schemes," in *Proceedings of the IEEE International Symposium on Information Theory*, Sevilla, Spain, Sept. 2013, pp. 479–483.

[88] S. Buzaglo and Y. Yaakobi, "Constrained codes for rank modulation," in *Proceedings of the IEEE International Symposium on Information Theory*, Honolulu, USA, June 2014, pp. 2396–2400.

[89] D. Slepian, "Permutation modulation," *Proceedings of the IEEE*, vol. 53, no. 3, pp. 228–236, March 1965.

[90] S. Buzaglo, E. Yaakobi, T. Etzion, and J. Bruck, "Error-correcting codes for multipermutations," in *Proceedings of the IEEE International Symposium on Information Theory*, Istanbul, Turkey, July 2013, pp. 724–728.

[91] X. Liu and S. C. Draper, "LP-decodable multipermutation codes," in *Proceedings of the 52nd Annual Allerton Conference on Communication, Control and Computing*, Monticello, USA, Sept. 2014, pp. 828–835.

[92] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Transactions on Information Theory*, vol. 28, no. 1, pp. 55–67, Jan. 1982.

[93] R. Heymann, H. C. Ferreira, and T. G. Swart, "Combined permutation codes for synchronization," in *Proceedings of the IEEE International Symposium on Information Theory and Its Applications*, Hawaii, USA, Oct. 2012, pp. 230–234.

[94] R. Heymann, J. H. Weber, T. G. Swart, and H. C. Ferreira, "Concatenated permutation block codes based on set partitioning for substitution and deletion error-control," in *Proceedings of the IEEE Information Theory Workshop*, Sevilla, Spain, Sept. 2013, pp. 1–5.

[95] D. E. Knuth, *The art of computer programming, Volume 1, Fundamental Algorithms*, 3rd ed., Addison-Wesley, Ed., 1997.

[96] R. Heymann, J. H. Weber, T. G. Swart, and H. C. Ferreira, "Concatenated permutation block codes for correcting single transposition errors," in *Proceedings of the IEEE Information Theory Workshop*, Hobart, Australia, Nov. 2014, pp. 576–580.