Copyright and Citation Considerations for this Thesis/Dissertation

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

- NonCommercial — You may not use the material for commercial purposes.

- ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

How to cite this thesis
CODING FOR THE CORRECTION OF SYNCHRONIZATION ERRORS

by

ALBERTUS STEPHANUS JACOBUS HELBERG

A thesis submitted as partial fulfilment of the requirements for the degree

R A U

DOCTOR OF ENGINEERING in

ELECTRICAL AND ELECTRONIC ENGINEERING in the

FACULTY OF ENGINEERING at the

RAND AFRIKAANS UNIVERSITY

SUPERVISOR : Prof H C Ferreira

November 1993
"Every man is a valuable member of society who by his observations, research and experiments procures knowledge for men. It is in his knowledge that man has found his greatness and his happiness. ... No ignorance is probably without loss to him, no error without evil."

James Smithson, Founder of the Smithsonian Institution
(From the wall of the Smithsonian Institution, Washington, DC, USA)

DEDICATION

I dedicate this work to my Lord Jesus Christ, who knew about timing problems long before telecommunications was invented:

"There is a time for everything, and a season for every activity under heaven."

"For there is a proper time and procedure for every matter"

Ecc. 3:1 and 8:6, NIV

ACKNOWLEDGEMENT

It is with gratitude that I acknowledge the many colleagues and friends who accompanied me through the period of my studies. I especially wish to thank my supervisor, prof. H. C. Ferreira, for the many hours, (some of them late at night), he spent with my work. I am also indebted to Prof. Dr. Ir. A. J. Han Vinck for the insight he provided during my studies at the Institute for Experimental Mathematics in Essen, Germany. Last, but not least, I thank my wife for her loving acceptance and support during some trying moments.

Albert
## CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>viii</td>
</tr>
</tbody>
</table>

### 1 INTRODUCTION
1.1 Problem statement  
1.2 Organization of thesis

### 2 BACKGROUND
2.1 Aspects of synchronization
   2.1.1 Carrier and clock synchronization
   2.1.2 Frame and word synchronization
   2.1.3 Network synchronization
2.2 Coding for synchronization
2.3 Some commonly used coding definitions
2.4 Synchronizable codes
2.5 Codes for the correction of insertions or deletions
CONTENTS

2.6 Number theoretic constructions of synchronization error correcting codes 2.9
2.7 Research direction 2.10

3 THE SINGLE INSERTION/DELETION CORRECTING CODES DESCRIBED BY LEVENSHTEIN
3.1 Preliminaries 3.1
3.2 Levenshtein codes 3.3
3.3 Cardinality of Levenshtein codes 3.5
3.4 New properties of the Levenshtein codes.
   3.4.1 Weight spectra 3.7
   3.4.2 Hamming distance properties of Levenshtein codes 3.9

4. CONSTRAINED SUBCODES OF THE LEVENSHTEIN CODES
4.1 \((d, k)\) constrained subcodes 4.1
4.2 Additive error correcting subcodes 4.3
4.3 Spectral shaping subcodes
   4.3.1 DC free Levenshtein subcodes 4.4
   4.3.2 Higher order dc constrained Levenshtein subcodes 4.5
   4.3.3 Minimum bandwidth subcodes 4.7
   4.3.4 Combined spectral shaping subcodes 4.7
4.4 Decoding
   4.4.1 Decoding strategy 4.11
   4.4.2 Second stage decoding
      4.4.2.1 Additive error 4.13
      4.4.2.2 Synchronization error 4.13
   4.4.3 Third stage decoding 4.14
   4.4.4 An \(n = 12\) code example 4.15
4.5 Semi systematic spectral shaping
   Levenshtein subcodes 4.17
4.6 Summary 4.17
CONTENTS

5 DOUBLE ADJACENT SYNCHRONIZATION ERROR CORRECTING CODES

5.1 The double adjacent synchronization error correcting
(a = 2) Levenshtein subcodes 5.1

5.2 A double adjacent synchronization error
code partition 5.3

5.3 Corroborating results 5.4

5.3.1 Example 5.5

5.3.2 Example 5.6

6 MULTIPLE SYNCHRONIZATION ERROR CORRECTING CODES

6.1 Number theoretic double synchronization error correcting
(s = 2) codes 6.1

6.1.1 Properties of number theoretic s = 2 codes 6.1

6.2 Bounds on the cardinality of number theoretic
s = 2 correcting codes

6.2.1 Upper Bound 6.7

6.3 A number theoretic s = 2 correcting code 6.9

6.4 General s - insertion/deletion correcting
number theoretic codes 6.13

6.4.1 Weight/distance structure of s - insertion/deletion
correcting number theoretic codes 6.15

6.5 Bounds on the synchronization error capability

6.5.1 Upper bound 6.16

6.5.2 Lower bound 6.16

6.6 A general number theoretic s - insertion/deletion
correcting construction 6.18

6.7 Corroborating results 6.21

6.7.1 Example 6.21

7 CONCLUSIONS AND OPEN QUESTIONS

7.1 Overview of new contribution 7.1

7.2 Future research and open questions 7.3

7.3 Conclusion 7.6
CONTENTS

REFERENCES

APPENDICES

APPENDIX A  SEMI SYSTEMATIC SPECTRAL SHAPING CODES
APPENDIX B  PROOFS OF DOUBLE ADJACENT INSERTION/DELETION CORRECTING CODE CLASSES
APPENDIX C  TEST PROGRAM FOR $a = 2$ INSERTION/DELETION CORRECTING CODES
APPENDIX D  TEST PROGRAM FOR GENERAL $s$ - INSERTION/DELETION CORRECTING CODES
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE No.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Information bits vs. code word length for $s = 1$ Levenshtein codes</td>
<td>3.6</td>
</tr>
<tr>
<td>3.2</td>
<td>Weight/distance relationships for Levenshtein codes</td>
<td>3.11</td>
</tr>
<tr>
<td>4.1</td>
<td>Information bits vs. code word length for $d$-constrained Levenshtein subcodes</td>
<td>4.2</td>
</tr>
<tr>
<td>4.2</td>
<td>The number of information bits vs. code word length for additive error correcting Levenshtein subcodes</td>
<td>4.3</td>
</tr>
<tr>
<td>4.3</td>
<td>The number of information bits vs. code word length of the balanced Levenshtein subcodes</td>
<td>4.5</td>
</tr>
<tr>
<td>4.4</td>
<td>Normalized spectrum of $R = 3/8$ code example</td>
<td>4.8</td>
</tr>
<tr>
<td>4.5</td>
<td>Power spectrum of a rate $R = 5/12$ code</td>
<td>4.9</td>
</tr>
<tr>
<td>4.6</td>
<td>Information symbols vs. word length of the combined spectral shaping subcodes</td>
<td>4.10</td>
</tr>
<tr>
<td>5.1</td>
<td>Code rates for $\alpha = 2$ Levenshtein subcodes</td>
<td>5.2</td>
</tr>
<tr>
<td>5.2</td>
<td>Cardinalities of double adjacent deletion/insertion correcting codes</td>
<td>5.7</td>
</tr>
<tr>
<td>FIGURE No.</td>
<td>TITLE</td>
<td>PAGE</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6.1</td>
<td>Schematic diagram showing the relationship between Hamming distance and weight for number theoretic $s = 2$ correcting codes</td>
<td>6.6</td>
</tr>
<tr>
<td>6.2</td>
<td>Bounds on the cardinality of number theoretic $s = 2$ correcting codes</td>
<td>6.8</td>
</tr>
<tr>
<td>6.3</td>
<td>A comparison of $s = 2$ code cardinalities and $s = 2$ bounds</td>
<td>6.12</td>
</tr>
<tr>
<td>6.4</td>
<td>Weight-distance structure of number theoretic $s$ - correcting codes</td>
<td>6.17</td>
</tr>
<tr>
<td>6.5</td>
<td>A comparison between rate and bounds for a $s = 3$ code</td>
<td>6.19</td>
</tr>
<tr>
<td>6.6</td>
<td>A comparison between rate and bounds for a $s = 4$ code</td>
<td>6.20</td>
</tr>
<tr>
<td>6.7</td>
<td>A comparison between rate and bounds for a $s = 5$ code</td>
<td>6.20</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE No</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Code book for a rate ( R = \frac{3}{8} ) code</td>
<td>4.8</td>
</tr>
<tr>
<td>4.2</td>
<td>A rate ( R = \frac{5}{12} ) code book</td>
<td>4.16</td>
</tr>
<tr>
<td>4.3</td>
<td>Three possible decoders of the code book in table 4.2</td>
<td>4.17</td>
</tr>
<tr>
<td>6.1</td>
<td>Code rates of number theoretic double synchronization error correcting codes</td>
<td>6.12</td>
</tr>
<tr>
<td>7.1</td>
<td>A (7, 4) Hamming code</td>
<td>7.4</td>
</tr>
<tr>
<td>7.2</td>
<td>A (7, 3) ( s = 1 ) insertion/deletion correcting code</td>
<td>7.4</td>
</tr>
<tr>
<td>7.3</td>
<td>A (7, 2) ( s = 2 ) insertion/deletion correcting code</td>
<td>7.5</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>Integer representing the residue in a modulus operation</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>Number of adjacent insertions or deletions</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>Integer representing the residue in a modulus operation</td>
<td></td>
</tr>
<tr>
<td>$C$</td>
<td>A code book</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>Minimum runlength constraint</td>
<td></td>
</tr>
<tr>
<td>$d_{\text{min}}$</td>
<td>Minimum Hamming distance of a code book</td>
<td></td>
</tr>
<tr>
<td>$d(x, y)$</td>
<td>Hamming distance between code words $x$ and $y$</td>
<td></td>
</tr>
<tr>
<td>$\delta(x, y)$</td>
<td>Levenshtein distance between code words $x$ and $y$</td>
<td></td>
</tr>
<tr>
<td>$l, l(\cdot)$</td>
<td>Length of a sequence</td>
<td></td>
</tr>
<tr>
<td>$M$</td>
<td>Parameter determining the separation between words of a synchronizable code</td>
<td></td>
</tr>
<tr>
<td>$m$</td>
<td>Number of source bits</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>Number of code bits</td>
<td></td>
</tr>
<tr>
<td>$N(\cdot)$</td>
<td>The number of elements</td>
<td></td>
</tr>
<tr>
<td>$p$</td>
<td>Subword of a code word</td>
<td></td>
</tr>
<tr>
<td>$q(x, y)$</td>
<td>Length of the largest common subword of $x$ and $y$</td>
<td></td>
</tr>
<tr>
<td>$q$</td>
<td>Size of symbol alphabet, $q = 2$ for binary codes</td>
<td></td>
</tr>
<tr>
<td>$R$</td>
<td>Code rate, $R = m/n$</td>
<td></td>
</tr>
</tbody>
</table>
## LIST OF SYMBOLS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>Number of random insertions or deletions</td>
</tr>
<tr>
<td>$t$</td>
<td>Number of additive errors</td>
</tr>
<tr>
<td>$u$</td>
<td>Operand in a modulus operation, e.g. $\sigma \equiv a \mod(u)$</td>
</tr>
<tr>
<td>$v$</td>
<td>A vector of integers $v_i$ for use in a code word moment</td>
</tr>
<tr>
<td>$w, w(\cdot)$</td>
<td>The Hamming weight of a sequence</td>
</tr>
<tr>
<td>$x, y, z$</td>
<td>A code word of length $n$</td>
</tr>
</tbody>
</table>
CHAPTER ONE

INTRODUCTION

1.1 Problem statement

In the ideal communication system no noise is present and error free communication is possible. In practice, however, several factors influence the correctness of the communication. One of the most important of these factors is the synchronization of the message. Synchronization techniques form an integral part of data communication systems and without synchronization no comprehensible message can be received.

An example of a communication system in which synchronization errors occur is the plesiochronous communication network which is used in many telephone networks [49, 50]. A common problem with the use of the multiplexers in such a network is that output pulses may occur that do not contain valid data, due to minor discrepancies in the clock frequencies of the incoming signals. These inserted bits are termed "justification bits" and their presence is signaled over the link by the justification control bits or stuffing control bits, which are included in the frame [49 and 50]. Synchronization of the network is dependent, among other factors, on the correct decoding of the stuffing control bits.

Synchronization at the receiver can also be lost when the frame markers are not recognizable due to errors on the channel. This could be due to jitter or wander of the digital signal. Another major cause for loss of synchronism on a PCM system is the insertion or deletion of
INTRODUCTION

1.2 Organization of thesis.

In chapter 2, an overview of coding techniques for synchronization is given. Some of the work in this field is discussed and a few definitions for use in coding is given. From this literature study, a choice on the research direction is made.

Chapter 3 deals with the number theoretic single insertion/deletion correcting codes described by Levenshtein. These codes are a subclass of the class of asymmetric error correcting code constructions of Varshamov and will be termed Levenshtein codes to avoid any ambiguity. This term also implies the use of the codes as single insertion/deletion correcting codes. Numerous new properties of the Levenshtein codes are presented which are used in later chapters.

A number of constrained subcodes of the Levenshtein codes are described in chapter 4. The most interesting classes of subcodes are the subcodes which combine several different constraints on the power spectral density of the code. These codes are investigated extensively and a decoding algorithm is given for a subclass of codes that combine Levenshtein codes with higher order dc free and minimum bandwidth spectral constraints.
In chapter 5, the first results of an attempt to extend the synchronization error correcting capability of number theoretic codes are given, namely codes that can correct two adjacent deletions or insertions. These constructions improve on some of the known constructions with regard to rate or complexity. A comparison of the rates of the known codes is presented.

A general number theoretic construction method for correcting any number of insertions and/or deletions is presented in chapter 6. This construction is developed by first describing the properties of double synchronization error correcting codes and then developing a number theoretic construction method for these codes. The properties of the double synchronization error correcting codes are generalized for any number of synchronization errors and a general construction is derived from these properties. The code rates of these codes are compared to the known bounds.

Finally, in chapter 7 an overview of the results of this study is presented. The degree in which the aim of the investigation was fulfilled is discussed and several open questions are stated for possible investigation in the future. The work is concluded with references and appendices.
2.1 Aspects of synchronization

The term "synchronization" is derived from the words "syn," meaning "together," and "chronous," meaning "time." Synchronization is then the process of aligning the time scales of at least two periodic processes that are separated in space. Synchronization is a fundamental problem in digital communication systems and in general there must be distinguished between several different types of synchronization. The following are some of the levels of synchronization found in digital communications:

2.1.1 Carrier and clock synchronization

Carrier synchronization is required in receivers that use bandpass signaling with a carrier wave and coherent detection. Proper detection of a signal requires a local carrier reference that agrees closely in frequency and phase with the received pulses. Several carrier synchronization techniques are known and the use of a specific synchronizer depends on the type of signal, e.g. a phase-locked loop (PLL) is used for signals that have a discrete spectral line at the carrier frequency.

It is also necessary to know when one data symbol ends and another begins, and the term "clock synchronization" [46] or "bit synchronization" [47 and 48], describes this process. In
many systems, the local clock must be derived from the incoming signal, which often does not
it in an explicit form. In baseband communication systems, the clock can also be recovered
through the use of a PLL.

2.1.2 Frame and word synchronization
If coding is part of the waveform design, then the decoder cannot operate unless the recovered
symbols can be separated into the proper groups. In the case where block coding is used, the
received symbols must be separated into words and word synchronization must be maintained.
When a TDM (Time Division Multiplex) system is used, the received multiplexed data must
be sorted and directed to the appropriate output channel. This process is called frame
synchronization. Closely related to frame synchronization is content synchronization in which
the content of a frame is synchronized, i.e. the information within specific fields within a
frame is distinguished from other fields. Thus it is important to know when the bits in a
frame are control bits, parity bits, synchronization bits or data bits.

2.1.3 Network synchronization
The synchronization of access to the communications medium, also termed network
synchronization, is especially important when there are several users of the communication
medium. This is an increasing concern in the control of local area networks (LAN's),
multipoint broadcasting and switched digital communications networks. In such systems it is
necessary to control which point may transmit and which point may receive at a given instant.

There may be certain synchronization aspects of different communications schemes that are
not covered by the three classes of synchronization described above, or may go under different
names in different systems. In such a way, the synchronization of waveforms can be linked to
carrier synchronization, and byte synchronization can be linked to bit or symbol
synchronization. It should be noted that if a synchronization problem exists at any of the
different synchronization levels, that this problem usually propagates right through the higher
levels. When considering the synchronization aspects of a communications system, all
synchronization levels must be considered as important. For further study of the overall
synchronization problem, the reader is referred to a special issue of the IEEE Transactions on
Communications [46]. Some practical applications of synchronization techniques and
protocols can be found in [47] and [48] respectively.
2.2 Coding for synchronization

There has been extensive research done in the field of coding for use in synchronization. This mainly concerns the setup and detection of synchronization. We are however interested in using coding techniques for the correction of frame and word synchronization errors. In this line there have been three directions of thought:

a) coding such that synchronization errors cause only local disruptions. By this it is meant that the codes recover synchronization, but the errors caused during the limited period that synchronization was lost, are not corrected. These codes are called synchronization recovery codes or self synchronizing codes,

b) codes that modify known additive error correcting codes to correct synchronization and the errors that occurred during loss of synchronization,

c) codes that are constructed by number theoretic designs such that they correct synchronization errors and thus recover synchronization.

It is important at this stage that we define the concept of a synchronization error for use in frame and word synchronization.

**Definition 2.1**

A synchronization error is said to have occurred when the length of the code word has been affected. Such an error manifests itself in the bit stream as the deletion or the insertion of a binary one or zero.

It should be noted that the term synchronization error is also used to describe framing errors. The following definition of slip describes a class of framing errors that occur most often:

**Definition 2.2**

Slip is said to have occurred when the decoder does not frame the correct code word but rather the beginning and end of two adjacent code words.

Note that this definition means that synchronization has been lost, hence slip is also referred to as a synchronization error.
In the definitions above, the cause of the error is not of importance, but only in the effect it has on the bit stream. In this way the insertion or deletion of bits in a frame causes the following frames to slip and vice versa. If the frame "slips" in either direction, such slip errors can always be modeled as the insertion or deletion of bits in the frame. In the following discussions only binary codes are considered, although the definitions above are easily modified for multilevel codes.

2.3 Some commonly used coding definitions.

Due to the diversity of the codes that shall be presented, it is also necessary to define additive errors, the Hamming distance metric and the relation it has to the additive error - correcting capability of a code. The following definitions are well known and can be found in one form or another in any good text book on error - correcting coding such as [43] or [44].

Definition 2.3
An additive error occurs when the value of the binary bit in a code word is changed, thus a binary one is changed to a binary zero and vice versa. This type of error is known as an additive error as it can be modeled by adding a binary one modulo 2 to the bit in the error position.

Definition 2.4
The weight \( w(x) \) of a code word \( x \in C \) is defined as the number of binary ones in the word.

Definition 2.5
The Hamming distance \( d(x, y) \) between any two binary words \( x \) and \( y \) in a code \( C \) is defined as the number of positions in which \( x \) and \( y \) differ. The Hamming distance is easily determined by adding the two words \( x \) and \( y \) modulo 2 and determining the weight of the sum, thus \( d(x, y) = w(x \oplus y) \)

In definition 2.6 a relationship between the additive error - correcting capability and the minimum Hamming distance of a code is given.

Definition 2.6
Let \( d_{\text{min}} \) be the minimum Hamming distance between any two code words of a code \( C \). Then the code can correct \( t \) additive errors if and only if \( d_{\text{min}} \geq 2t + 1 \)
A measure of the efficiency of a code is the code rate $R$, which is defined as follows:

**Definition 2.7**

The rate, $R$, of a code is defined as the ratio of the number of information bits, $m$, to the number of code bits, $n$, onto which the information bits are mapped. Thus $R = m/n$. The rate $R$ represents the average information content per symbol.

In the next sections we briefly describe some of the work done in the three synchronization error correcting fields mentioned earlier. This is not a complete overview of all available literature on synchronization but rather a selection of works in order to briefly describe certain aspects of synchronization correction coding.

### 2.4 Synchronizable codes.

The basic idea behind synchronizable codes is to make word synchronization impervious to synchronization errors. This is done by constructing code words in such a way that the tail ends of the code words form a separation between the code words. Inspection of the bitstream thus reveals the beginning of the following code word. A formal definition (after [1, 2]) follows.

**Definition 2.8**

A finite code is called synchronizable if, and only if, there exists an integer $M$ such that the knowledge of the last $M$ letters of any message suffices to determine a separation of code words.

A similar but stronger condition on the code words yields the comma-free codes. These codes are subsets of the synchronizable codes with the condition that no overlap between code words yields another valid code word. Thus a valid code word can be uniquely identified in the bit stream.

The use of synchronizable codes is obvious. When a synchronization error occurs in a code word the length of the code word is changed. Thus the receiver does not know where the next code word begins. By using a synchronizable code, the receiver only searches for the next recognizable code word and synchronization is recovered. The code words that were corrupted are discarded.
In [1] Eastman and Even present a method for the construction of synchronizable block codes along with the proof that this construction is maximal in the sense that it contains as many code words as possible. An expression is also given for the cardinality of the code book. The authors then describe a construction of phase-shift-keying synchronizable codes based on the maximal construction presented. In [2] Eastman gives a construction technique for comma-free codes that is also maximal in the sense of cardinality for odd word length and any alphabet size.

The next logical step after the development of the synchronizable and comma-free codes was to incorporate error-correcting capabilities into the code. In [3] Stiffler proposes a method whereby cyclic additive error-correcting group code dictionaries are made comma-free without adding to the redundancy of the code or altering the error-correcting capability of the code under certain conditions. These codes are able to retain synchronization and correct a certain number of additive errors. It should be noted that with the use of these codes it might not always be possible to distinguish between a synchronous word with additive errors and a word that was corrupted by synchronization errors. By observing several words it is possible to make an increasingly accurate decision as to which case actually occurred.

Several methods for incorporating synchronizability into binary error-correcting codes have been proposed. In [4, 5, and 6] three different techniques are presented to change binary cyclic codes so that they are synchronizable. The technique presented by Levy in [4] involves changing the parity bits of the binary cyclic code to affect the change while the technique presented by Seguin in [5] achieves synchronizability by shifting the bit pattern of the binary cyclic code in such a way that the shifted versions have a maximum number of leading and trailing binary zeros. In [6] the properties of a binary cyclic code which is able to correct both additive errors and synchronization errors are presented by Shiva and Seguin, along with a decoding method. An approach to alter a binary Reed-Solomon code to achieve synchronizability is presented in [7] by Inaba and Kasahara. In [8] Hatcher constructed variable length codes that are synchronizable and additive error correcting. The technique presented by Tavares and Fukada in [9] involves the addition of a binary vector to a cyclic code to obtain a coset code that is able to correct additive errors and recover synchronization. The idea of synchronization recovery through the use of the additive error-correcting property of cyclic codes is extended to cyclic burst-error-correcting codes by Tong in [10].
In the light of the definitions 2.1 and 2.2, the work in [1 - 10] can be classified as follows:

The codes in [1 and 2] are able to recover synchronization (thus are slip correcting) but are unable to correct the code words in which the synchronization errors occurred. The codes described in [3, 4 and 8] are also slip correcting and are able to correct additive errors in correctly synchronized code words as well, but are unable to correct the words in which synchronization errors occurred. These codes do not correct slip and additive errors simultaneously. The codes that are able to correct slip (i.e. recover synchronization) and correct additive errors simultaneously can be found in [5, 6, 7, 9 and 10]. It is important to note that none of the above codes explicitly considers the case of a synchronization error as a change in the length of the bit stream but instead corrects the framing. If, however, the slip was caused by the deletion or insertion of bits, the code word in which the deletions or insertions occurred is not decoded correctly, unless additional additive error correcting capabilities are built into the code. The following work has been done to consider the case of bit insertions or deletions and possible correction thereof.

2.5 Codes for the correction of insertions or deletions.

In this section we briefly summarize codes that are designed to correct synchronization errors (i.e. insertions and/or deletions). Some of these codes also have additive error correcting capabilities. The importance of these codes are that they are capable of extracting the complete message from the bit stream.

A model for the synchronization error channel is proposed by Ullman in [11]. From this model a lower and upper bound on the redundancy necessary to correct errors at a given error rate is derived. A coding technique that does not require special synchronization sequences between words (similar to synchronizable and comma-free codes) is presented by Calabi and Hartnett in [12]. This technique is capable of correcting at most a single synchronization error in every three consecutive code words or one additive error under the same conditions. If the restriction on the minimum number of consecutive words in which the synchronization error may occur is relaxed, the additive error correcting capability of the code is increased but not the synchronization error correcting capability. The decoder must wait for the reception of at most two of the three words before correctly decoding the first word.
Another approach to the correction of synchronization errors is to place a special synchronization sequence (also referred to as a *comma*), between the code words. In [13] a block code that is able to correct a single deletion or insertion is presented by Sellers. The coding method involves inserting a comma into a burst error correcting code at periodic intervals. The comma locates the approximate position of the synchronization error and at this location a bit is inserted or deleted to correct the length of the bit stream. The burst-error-correcting code then corrects the erroneous bits between where the error occurred and where the correction took place. In [14] Scholtz constructed codes using a suffix construction procedure. The construction technique and the mechanization of the synchronizers are explained in detail. Note that this code does not correct synchronization errors but recovers synchronization, i.e. the codes are synchronizable. The method is described here because the suffix construction technique is similar to placing commas between code words. In [15] a block code which corrects a single synchronization error per code word is presented by Ullman. This code has at most three bits more redundancy than that of an optimal code for this class of errors (c.f. [11]). A discussion of frame synchronization techniques and comparisons to techniques such as comma-free coding is presented by Scholtz in [16]. An interesting point from the comparison between codes that utilize commas and comma-free codes is that the comma-free codes are more efficient, i.e. they require less redundancy to achieve synchronizability. The troubles of achieving synchronization is aptly summarized in [16] when the author gives the following quotation:

*The time is out of joint: O cursed spite that ever I was born to set it right!*  
*(*Hamlet*, I, 5)*.

Two approaches that stand alone are given in [17 and 18]. The first is the concept of an information lossless sequential machine in which synchronization can be recovered by using as input to the machine a synchronizing input sequence which always causes the machine to return to a starting state. The approach in [18] is to use sequential decoding of a convolutional code to determine the channel characteristics of channels which cause synchronization errors or to determine an efficient decoding method when these errors occur.

We next consider synchronization error correcting codes that are constructed using a number theoretic approach. These codes are special cases of codes that were originally developed to correct asymmetrical additive errors, i.e. the probability of a binary one becoming a binary zero is much higher than the probability of a binary zero becoming a binary one, or vice versa.
For more information regarding these codes we refer the reader to a paper by Constantin [19]. A study of certain block codes, including those presented in [19], is done by Delsarte and Piret in [20]. The main results of the above study are concerned with the additive error - correcting capability of the codes and the enumeration thereof.

2.6 Number theoretic constructions of synchronization error correcting codes.

During the 1960's Levenshtein [21, 22] noticed that certain number theoretic constructions presented by Varshamov and Tenengolts for codes that correct single asymmetrical errors (c.f. [23]) can be modified to correct single insertions and deletions. In [21] the characteristics and conditions of binary codes that can correct insertions, deletions and additive errors are given. For a code to be able to perform these tasks it was assumed that the limits, i.e. the beginning and end of the code word being decoded, is known. Thus a third symbol, a "space," is assumed between code words. It was also noted in [21] that commas could be used without disturbing the synchronization error correcting capability of the code. In [22] Levenshtein presents perfect codes that are capable of correcting single deletions. These codes are perfect in the sense that all possible subwords can be found from all the valid code words. This construction technique is not restricted to binary codes although only the decoder for the binary case is given. Further work by Levenshtein includes a binary code correcting the deletion of one or two adjacent bits [26]. Other advances in this field are now discussed.

In [24] Tenengolts described a class of codes correcting a deletion as well as an additive error in the preceding bit. These codes also allow two adjacent asymmetric additive errors to be corrected. The same author later extends this work to nonbinary codes that are capable of correcting a single deletion or insertion of the nonbinary symbol [25]. In [27] Tanaka and Casai constructed block codes that are capable of simultaneously correcting $s$ or fewer synchronization errors in $t$ consecutive words for any $t \geq 2s + 1$, and $s$ or fewer additive errors in each of $t - 1$ code words under the condition that there exists at least a single word without errors among the $t$ consecutive words under consideration. The $t$ in the above paper should not be confused with the commonly used symbol for the additive error correcting ability of a code. The codes from [27] are based on some of the results in [21]. More recently, Hollman [8] established a relation between the insertion and deletion correcting capability of a code and its Levenshtein distance metric described by Levenshtein in [21]. In [29 and 30], Bours presented generalized bounds on the cardinality of codes, not necessarily binary, that are
capable of correcting insertions and deletions, as well as subcodes of Levenshtein codes [21] that are able to correct two adjacent deletions or one deletion. Work regarding the connection between Levenshtein codes and Steiner systems is presented by Van Trung in [31].

2.7 Research direction.

From this investigation into the main thoughts surrounding synchronization error correcting codes, a direction for further research must be pursued. The idea of a comma is to determine the boundaries of the code word and from there on to correct the synchronization and/or any additive errors that might have occurred. This is essentially also true for the comma-free and synchronizable codes due to the fact that code word boundaries are still defined. The number theoretic codes on the other hand assume that the boundaries of the codes are known and work from there on to correct the synchronization errors and/or additive errors that have occurred in that word. To achieve practical codes (i.e. good code rates, low complexity, ease of implementation, etc.), it is probably necessary to combine number theoretic code characteristics with comma or comma-free characteristics. The greatest promise seems to lie in the number theoretic codes which have elegant structures and good rates. Unfortunately, the synchronization error correcting capabilities of these codes are still very low, only one random synchronization error or two adjacent synchronization errors can be corrected. We therefore investigate the Levenshtein codes from [21] in chapter 3 to determine their characteristics which hopefully could lead to better codes.
3.1 Preliminaries

In this chapter we present some of the known properties of synchronization error correcting codes as well as the construction techniques presented by Levenshtein for the correction of deletions, insertions and reversals. We also present some new properties of the Levenshtein codes.

In the mid sixties, Levenshtein [21 and 22] determined certain properties to which codes must comply to be able to correct synchronization errors. The most important of these properties is the concept of Levenshtein distance and the relation of this distance to error correcting capability. We give the main results. For the proofs of the lemmas we refer the reader to the literature cited.

The first lemma (from [21]) describes the relationship between the deletion (or insertion) synchronization error correcting capability and the combined synchronization error correcting capability of a code.
Lemma 3.1
Any code $C$ that can correct $s$ deletions (or insertions) can correct a combination of $s$ deletions and insertions. The following example clarifies the concept of a combination of $s$ errors:

Example
Consider the two code words, $000$ and $111$. These code words are capable of correcting $s = 2$ insertions or deletions. Consider all the possible words that can be formed of each code word after a single insertion and a single deletion:

$111, 110, 101, 011$
$000, 001, 010, 100$

This example shows that the resulting words after a single insertion followed by a single deletion are unique to each code word. It is thus possible for a $s$ insertion/deletion correcting code to correct up to $s$ errors consisting of insertions and/or deletions.

The following lemma describes the Levenshtein distance metric and relates the synchronization error correcting capability of a code to its minimum Levenshtein distance. Due to notational differences in references [21, 22] and [28], the notation used in [28] is adopted.

Lemma 3.2
Consider a function $\delta(x,y)$ defined on pairs of binary words and equal to the minimum number of deletions and insertions necessary to transform the word $x$ into the word $y$. This function is known as the Levenshtein distance metric. Then a code $C$ can correct $s$ deletions and insertions if and only if

$$\delta(x, y) > 2s$$

(3.1)

for any two different words $x$ and $y$ in $C$. 
Due to lemma 3.1, it is sufficient to consider only deletions when investigating synchronization error correcting codes as every code that can correct $s$ deletions can also correct $s$ insertions. The following definitions from [21, 22 and 28] come in handy when working with synchronization error correcting codes:

**Definition 3.1**
A word $x$ has a subword $p$ if $p$ can be obtained from $x$ by an arbitrary number of deletions.

**Definition 3.2**
The length of the largest common subword $Q(x, y)$ for words $x$ and $y$ in code $C$ is defined as:

$$Q(x, y) = \max \{ |p| : p \text{ is a common subword of } x \text{ and } y \} \quad (3.2)$$

Definition 3.2 can now be used to calculate the Levenshtein distance $\delta(x, y)$ as follows:

$$\delta(x, y) = |x| + |y| - 2Q(x, y) \quad (3.3)$$

The following property is useful for constructing practical codes due to the fact that it allows the use of "commas" (i.e. binary sequences separating valid code words that allow the determination of the beginning and end of the code words).

For any prefix $\alpha$ and any suffix $\beta$ and two code words $x, y \in C$, the following relations are valid:

$$Q(\alpha x \beta, \alpha y \beta) = Q(x, y),$$

and

$$\delta(\alpha x \beta, \alpha y \beta) = \delta(x, y). \quad (3.4)$$

### 3.2 Levenshtein codes

In 1965 Levenshtein [21] noted that a certain number theoretic construction for the correction of single asymmetrical errors of the type "0" to "1" proposed by Varshamov and Tenengolts [23] can also correct single synchronization errors. This technique partitions all possible
binary sequences of length \( n \) into distinct subsets by associating a certain value to the position or index of the ones in the sequence. Using these indices, the code word moment is computed and used to construct the code as follows:

**First class of codes**

Let \( n \) be the length of the binary sequence \( x \), where \( x = x_1 x_2 \ldots x_n \). Then single synchronization error correcting codes are found by only including in the code book code words which comply with

\[
\sum_{i=1}^{n} i \cdot x_i \equiv a \mod (n+1),
\]

(3.5)

for a fixed \( a \), where \( 0 \leq a \leq n \). Thus the \( 2^n \) possible binary words are partitioned into \( (n + 1) \) different \( s = 1 \) correcting code books, where each code book is denoted by an integer \( a \).

Levenshtein also extended the error correcting capabilities of these codes to include additive error correcting capabilities.

**Second class of codes**

Let \( n \) be the length of the binary sequence \( x \), where \( x = x_1 x_2 \ldots x_n \). Then codes correcting either single synchronization errors or single additive errors are found using the following construction:

\[
\sum_{i=1}^{n} i \cdot x_i \equiv a \mod (2n),
\]

(3.6)

for a fixed \( a \), where \( 0 \leq a \leq 2n - 1 \). Thus each of the \( 2n \) codes has \( s = t = 1 \).

It is important to note the Levenshtein codes retain synchronization by correcting any single insertions or deletions that occur in a code word. However, if more than a single error should occur in a code word, synchronization is irretrievably lost. One method of enabling the decoder to recover synchronization, is through the use of markers. The use of markers in conjunction with Levenshtein codes is proposed by Clarke, Helberg and Ferreira in [45].
Markers are predetermined binary words used for the alignment of transmitted sequences. Note that no data is mapped onto a marker. From equation (3.4) it is seen that the use of markers does not reduce the insertion/deletion correcting capability of a code. The markers fulfill a dual purpose, namely the detection of an error as well as differentiating between the different types of errors (insertion, deletion or additive error.) These markers must comply to certain rules which enable the decoder to recognise the type of error when it occurs while still uniquely recognizing the marker. If this were not so, it would not be possible to find the beginning and end of a code word. If the error correcting capability of the code is exceeded, these markers will enable the decoder to recognise subsequent uncorrupted markers and resynchronize.

3.3 Cardinality of Levenshtein codes.

The cardinality of the class of Levenshtein codes which is described by (3.5), was investigated by several authors, and the reader is referred to references [19 - 22] for the proofs of the properties that we present. Levenshtein presented a lower bound for the cardinality of the $s = 1$ codes as defined in equation (3.5) as follows:

**Property 3.1**

Let $n$ be the length of the code words in code $C$ and let $|C|$ denote the cardinality of code $C$, i.e the number of code words in code $C$. Then

$$|C| \geq 2^n / (n + 1).$$  \hspace{1cm} (3.7)

As $n$ approaches infinity, the cardinality of code $C$ can be approximated by:

$$|C| \sim 2^n / n, \hspace{0.5cm} n \rightarrow \infty.$$  \hspace{1cm} (3.8)

In 1967 Ginzburg [33] determined the cardinality of the binary Varshamov - Tenengol'ts codes and thus in effect the cardinality of the Levenshtein codes. Ginzburg further determined that the code corresponding to $a = 0$ in the partition has the largest cardinality for any value of $n$. In the same vein he noted that the code corresponding to $a = 1$ always has the lowest cardinality. Note that these values of the cardinality are not unique, i.e. other codes corresponding to different values of $a$ may possibly have the same cardinality.
Levenshtein [21] also presented bounds for the $s = t = I$ construction described in equation (3.6), as follows:

$$2^{n-I}/n \leq |C| \leq 2^{n}/(n + 1).$$

(3.9)

The codes described in equation (3.5) were constructed and the cardinality determined for word lengths $n \leq 23$. Using this information, the rates of the codes were determined. (c.f. definition 2.7). The number of information bits vs. code word length of Levenshtein's first class of codes with $a = 0$ for short word lengths is presented in figure 3.1.

![Figure 3.1: Information bits vs. code word length for $s = 1$ Levenshtein codes.](image)

We next investigate some other properties of the Levenshtein codes that have not been published before.
3.4 New properties of the Levenshtein codes.

3.4.1 Weight spectra.
In this section we confine our interest in the Levenshtein codes to codes corresponding to \( a = 0 \) as these codes always have the highest rate. We first investigate the weight spectra of these codes, where the weight spectra is defined as the distribution of the number of code words, \( N(w) \), of weight \( w \) in the code \( C \). The following two examples clarify the concept:

\[
\begin{array}{cccccccccccc}
\ h \ n = 10 & w & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
N(w) & 1 & 0 & 5 & 10 & 20 & 22 & 20 & 10 & 5 & 0 & 1 \\
\h n = 11 & w & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
N(w) & 1 & 0 & 5 & 14 & 28 & 38 & 38 & 28 & 15 & 4 & 1 & 0 \\
\end{array}
\]

As illustrated above, it was found that the weight spectra is asymmetrical for odd code word lengths \( n \) but is symmetrical around \( n/2 \) for \( n \) even. The weight spectra are later used to find the cardinality of the dc free constrained subcodes of the Levenshtein codes. We now present some propositions describing the weight spectra of Levenshtein's first class of codes:

**Proposition 3.1**
The value \( N(w) \) for \( w = 2 \) is given by:

\[
N(2) = \left\lfloor \frac{n}{2} \right\rfloor 
\]  

(3.10)

**Proof**
For \( w = 2 \) and \( a = 0 \) the sum \( \sum_{i=1}^{n} i \cdot x_i \) must equal \( (n + 1) \) because

\[
0 < \sum_{i=1}^{n} i \cdot x_i \leq n + (n - 1) \\
= 2n - 1 \\
< 2(n + 1) 
\]  

(3.11)
i.e. the code word moment of any word of weight 2 (denoted by $\sigma_2$) is at most $2n - 1$. This is less than $2(n + 1)$, and since $a \equiv 0 \pmod{(n + 1)}$, this code word moment must be an integer multiple of $(n + 1)$ larger than 0, which leaves only the value $(n + 1)$ for $\sigma_2$. Furthermore, the only way a code word of weight $w = 2$ and code word moment $\sigma_2 = (n + 1)$ can be found is as follows:

$$\sigma_2 = (n - i) + (i + i) = (n + 1)$$ for $i = 0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor - 1$.

There are thus $\left\lfloor \frac{n}{2} \right\rfloor - 1 + 1$ possible values for $i$, thus proving the proposition.

The following proposition describes the occurrence of code words of even length $n$ and explains why the weight spectra is symmetrical for $n$ even and $a = 0$.

**Proposition 3.2**

For codes of even length, the code words occur in complementary pairs if $a = 0$.

**Proof**

Let $n = 2z$. Consider complimentary code words, $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$.

Let $x \in C$ and $a = 0$. Then

$$\sum_{i=1}^{n} i \cdot x_i + \sum_{i=1}^{n} i \cdot y_i = \sum_{i=1}^{n} i$$

$$= \frac{n(n + 1)}{2}$$

$$= z(n + 1)$$

$$\equiv 0 \pmod{(n + 1)}.$$  \hspace{1cm} (3.12)

But

$$\sum_{i=1}^{n} i \cdot x_i \equiv 0 \pmod{(n + 1)}, \quad (x \in C \text{ and } a = 0).$$
THE SINGLE INSERTION/DELETION CORRECTING CODES ...  \hspace{1cm} \textit{CHAPTER 3}

Thus

\[ \sum_{i=1}^{n} i \cdot y_i = 0 \mod (n+1), \text{ and } y \in C, \text{ proving the proposition.} \]

\textbf{Corollary}

The weight spectrum is symmetrical for \( n \) even and \( a = 0 \).

3.4.2 \hspace{0.5cm} \textit{Hamming distance properties of Levenshtein codes.}

In this section we describe a relationship between the weight of a Levenshtein code word and the Hamming distance between code words of different weights. A number of propositions describing this relationship were proven and are now presented:

\textit{Proposition 3.4}

A Levenshtein code \( C \) has only one code word of either weight \( w = 0 \) or weight \( w = 1 \), i.e. code words of weight \( w = 0 \) and \( w = 1 \) do not occur in the same Levenshtein code \( C \) and if one of them occurs in code \( C \), there is only one code word of that weight.

\textit{Proof}

Consider two code words of length \( n \) and weights \( w = 0 \) and \( w = 1 \) respectively. Then a common subword of length \( n - 1 \) can always be found by deleting the \( x_i = 1 \) bit in the \( w = 1 \) code word and an arbitrary zero in the \( w = 0 \) code word. The same holds for two words of weight \( w = 1 \). It is not possible to determine which word was transmitted if this common subword is received, thus a single deletion (or insertion) cannot be corrected if words of these weights are in the same code book.

\textit{Proposition 3.5}

Levenshtein codes have a minimum Hamming distance of at least two between code words.
Proof
Consider two binary code words that have a Hamming distance of one. By deleting that binary symbol in each word that builds the distance, we find that the same subword can always be formed from both code words. This is not possible for a valid Levenshtein code, thus they are not valid code words. Thus the minimum distance must be at least two.

Proposition 3.6
Code words of a valid Levenshtein code have a minimum Hamming distance of at least four if they have the same weight.

Proof
In general, different binary sequences of the same weight have a Hamming distance of at least two. Consider any two binary words of the same weight of which \( w - 1 \) binary ones correspond. Such words have a Hamming distance of 2. For these two code words to be in the same code book, they must have the same code word moments. The binary "ones" that correspond in each word, have the same contribution to the code word moment, thus the contribution of the binary "ones" that do not correspond should also be the same. These binary "ones" are not at the same position and therefore cannot have the same contribution to the code word moment. It is thus impossible for the two words to have the same code word moment and therefore to be in the same code book.

From the above it is clear that the constant weight Levenshtein code words must have a minimum Hamming distance \( d_{\text{min}} > 2 \). The Hamming distances of constant weight code words occur in multiples of 2, thus \( d_{\text{min}} \geq 4 \).

Note however that proposition 3.6 is not valid for all single synchronization error correcting codes. Consider the following two binary words and all corresponding subwords after a single deletion:

\[
egin{align*}
011100 & : \quad 01110, 01100, 11100. \\
001101 & : \quad 00110, 00111, 00101, 01101.
\end{align*}
\]

As can be seen none of the subwords are the same, thus these code words can correct a single deletion or insertion. These words have the same weight but only have a Hamming distance of two.
Proposition 3.7
Levenshtein code words that differ in one unit of weight have a minimum Hamming distance of at least three.

Proof
Consider a code word of weight \( w = u \) and another of weight \( w = u - 1 \). Let \( u - 1 \) binary ones of these code words correspond in position. The Hamming distance between these words is one. We have already proven that the minimum Hamming distance of Levenshtein codes must be greater than one, thus these code words cannot be in the same code book. Next consider code words of which \( u - 2 \) binary ones correspond. These words have a Hamming distance of three. Thus code words that differ in one unit of weight must have a Hamming distance of at least three.

Figure 3.2: Weight/distance relationships for Levenshtein codes.
Propositions 3.4 to 3.7 can schematically be represented as in figure 3.2 which shows all the weight/distance properties at a glance. This figure will be used to construct similar weight/distance relationships for new more powerful insertion/deletion correcting codes. These relationships will be used to derive new bounds on the cardinality and insertion/deletion correcting ability of multiple insertion/deletion correcting codes. These figures will also be used to indicate possible construction methods for these codes.

In the next chapter we present results pertaining to the constrained subcodes of the Levenshtein codes.
CHAPTER
FOUR

CONSTRAINED SUBCODES OF THE LEVENSHTEIN CODES

In this chapter we present results pertaining to some of the subcodes of the Levenshtein codes. These subcodes combine several properties such as additive error correction, runlength constraints and charge constraints with the single synchronization error correcting property of the Levenshtein codes, hence the title of the chapter.

4.1 \((d, k)\) constrained subcodes.

Runlength constrained codes have been used extensively in both the magnetic and optic recording media [33, 34]. In runlength constrained codes, certain constraints are placed on the minimum and maximum length of runs of like symbols. A related class of codes are the \((d, k)\) codes, where the \(d\) - parameter denotes the minimum number of binary zeros between ones and the \(k\) - parameter the maximum number of binary zeros between ones, using the NRZI representation. As recording systems employ clock extraction for synchronization, it may be considered to relax the \(k\) constraint, which is necessary for accurate clock extraction [33], and compensating for this relaxation and possible synchronization errors by using a code that combines synchronization error correcting capabilities with a \(d\) - constraint. The foremost question is if these codes are as effective regarding rate as the known codes. We therefore investigate the rate of these combined codes first before commencing.
The results of a computer search for $d = 1$ and $d = 2$ concatenable subcodes of the first class of Levenshtein block codes as described in equation (3.5), are shown in figure 4.1. As can be seen, the rates are very low, although there is a steady increase in rate versus code word length. Also shown is the result of a computer search for $d = 1$ subcodes of the Levenshtein codes which have a minimum Hamming distance of 3. Such codes are able to correct an additive error or a synchronization error in addition to having a minimum runlength constraint. This result shows that very little rate loss is experienced when placing the additional Hamming distance constraint on the subcode. We conclude that the rates of these subcodes do not compare favourably to conventional $(d, k)$ codes, but that further investigation of the additive error correcting capabilities of the Levenshtein codes is justified.

![Figure 4.1: The number of information bits vs. code word length for $d$-constrained Levenshtein subcodes](image)
4.2 Additive error correcting subcodes.

In the previous section it was noticed that little rate loss is experienced by placing a Hamming distance constraint on the code words. In this section the \((d, k)\) constraints are discarded and the cardinality of Levenshtein subcodes that have a Hamming distance of \(d_{\text{min}} = 3\) and 5 is investigated. These codes are able to correct a single or double additive error respectively. The cardinalities of the \(d_{\text{min}} = 3\) subcode is compared with the second class of Levenshtein codes, as described in (3.6). In figure 4.2 the results of an exhaustive computer search are shown. It is found that the \(d_{\text{min}} = 5\) codes have very low rates and are not suitable for practical use. However, the \(d_{\text{min}} = 3\) codes have reasonable rates which increase with code word length and a rate \(R = 4/8\) is achieved. It is also found that these subcodes have exactly the same rates as Levenshtein's second class of codes, although the code books differ.

![Figure 4.2: The number of information bits vs. code word length for additive error correcting Levenshtein subcodes.](image-url)
4.3 Spectral shaping subcodes.

4.3.1 Dc free Levenshtein subcodes.

There are many transmission systems used presently that require the Running Digital Sum [35] of the transmitted sequence to be bounded [33]. In twisted pair systems where ac coupling is used, the transmitted sequence must have no dc component as this could result in saturation of transformers. Some magnetic recording systems also require a constraint on the accumulated charge at the recording head. All these transmission systems employ some form of synchronization and it is thus important to investigate the dc free subcodes of the Levenshtein codes.

A formal description of the Running Digital Sum (RDS) for binary block codes of length \( n \) is given below:

\[
RDS_n = \sum_{i=1}^{n} y_i = 0, \quad y_i \in \{-1,+1\},
\]

(4.1)

where \( y = \{y_1,y_2,y_3,...,y_n\} \) is a binary code word of length \( n \).

The first subcodes we consider are the balanced subcodes of the first class of Levenshtein codes. These subcodes conform to equation (3.5) with \( a = 0 \) as well as equation (4.1). The use of equation (4.1) implies that the number of binary zeros and ones in a code word are the same, thus the weight is \( w = n/2 \). This furthermore implies that only even length code words can be used in a binary block code. Another property of these balanced subcodes is that they have \( d_{\min} = 4 \). This is due to the Hamming distance/weight structure of the Levenshtein codes as discussed in chapter 3, section 3.4.2. The number of information bits vs. code word length of these block codes are presented in figure 4.3.

It is interesting to note that a rate \( R = 9/16 \) dc free, \( d_{\min} = 4 \) code is easily achieved by discarding all words that do not comply with equations (3.5) and (4.1). A similar code with a second order spectral null at 0 Hz was found by Immink [34, p245]. The new code improves
on a rate $R = 8/16$, $d_{\text{min}} = 4$ block code presented by Ferreira in 1984 [36] and a rate $R = 9/16$, $d_{\text{min}} = 4$ code presented by Blaum in 1988 [37]. This new code has the same properties as the cited codes and in addition can correct the deletion/insertion of a symbol. Furthermore, the construction of the new code is much simpler and it is easier to decode.

![Figure 4.3: The number of information bits vs. code word length of the balanced Levenshtein subcodes.](image)

### 4.3.2 Higher order dc constrained Levenshtein subcodes.

In this section we present a class of codes that form a subset of the Levenshtein codes. These codes can be used in systems that require a high suppression of the dc component, such as some magnetic and optic recording systems. In the construction of the codes, we constrain the binary code word in such a way that the second derivative of the power spectrum at $0$ Hz is zero. This is also termed as a first order null at dc.[35]

In magnetic and optical recording systems it is preferable to have a dc free code due to the physical properties of the recording media as well as the nature of the recording system. As an example, in optical recording a dc free code is used to reduce interaction between the data written on the track and the servo system employed [35]. By using a code that has a first order null at dc, a much higher suppression of the dc component is achieved.
The subcodes considered here have a minimum Hamming distance $d_{\text{min}} = 4$ and are able to correct a single additive error or detect three additive errors and hence a single peak shift error.

The new codes form a subset of the Levenshtein codes as described in equation (3.5). These codes are single synchronization error correcting codes, i.e. the loss or gain of a single binary symbol due to synchronization errors can be corrected. In the interest of completeness we restate equation (3.5) in (4.2):

$$ \sum_{i=1}^{n} ix_i = a \mod (n+1), \quad x_i \in \{0, 1\}. \quad (4.2) $$

where $\bar{x} = (x_1, x_2, \ldots, x_n)$ is a binary word, $n$ is the length of $\bar{x}$ and $a$ is an integer corresponding to the code partition.

In section 3.4.2 the investigation of the Levenshtein codes revealed relationships between the Hamming distance and weight of the Levenshtein codes. Of special interest is the observation that the balanced subcodes (i.e, having a weight of $n/2$) of a Levenshtein code have a minimum Hamming distance $d_{\text{min}} = 4$, thus these codes are able to correct a single additive error or detect three additive errors.

Relations describing the spectral shaping of binary codes can be found in [35]. From [38] a first order spectral null at 0 Hz, (i.e. the second derivative of the power spectrum at 0 Hz is zero), is found when a balanced code complies to the following criterion:

$$ \sum_{i=1}^{n} iy_i = 0, \quad y_i \in \{-1, +1\}. \quad (4.3) $$

where the variables $i$ and $n$ are defined as in (4.2) and furthermore $n$ is a multiple of 4. When investigating the new subcodes in the rest of this section, we assume a direct mapping of $\{0, 1\}$ onto $\{-1, +1\}$ using the function $y = 2x - 1$. 

Notice that equation (4.2) with $a = 0$ and equation (4.3) are similar. Investigation of this similarity [39], lead to the discovery that the first order spectral null codes of Immink [38] are contained in the balanced Levenshtein subcodes. Thus, the dc$^2$ constrained codes of Immink are able to correct the insertion or deletion of a single symbol.

4.3.3 Minimum bandwidth subcodes

Another property that can be incorporated into the subcodes is a spectral null at half the normalized frequency, also termed the Nyquist frequency. This is also called a minimum bandwidth property because only half of the bandwidth need be used to transmit the coded message without errors in a noiseless system. This allows a doubling of the possible code symbol transmission rate for a certain fixed bandwidth. The condition from [40], known as the Running Alternate Sum (RAS), for this property is stated in equation (4.4):

$$\sum_{i=1}^{n} (-1)^{i} y_i = 0, \quad (4.4)$$

where all the variables are the same as in (4.3).

4.3.4 Combined subcodes.

It is now possible to combine several of the properties mentioned above into a single subcode of the Levenshtein codes. For reasons to be stated later, it is chosen that both the minimum bandwidth property and the second order spectral null at dc be incorporated into a single code. We now present an example of such a code.

Example:

We consider the set of all binary words of length 8 and which have a weight of $n/2 = 4$. There are 70 such words. From this set, words are rejected that do not comply to equations (4.2), (4.3) and (4.4). The surviving set of words is the valid code which combines the properties of the individual codes.

For the case of $n = 8$ we find 8 valid code words, shown in Table 4.1. It is thus possible to map 3 information bits onto the 8 code bits, giving an information rate of $R = 3/8$. 
TABLE 4.1
CODE BOOK FOR A RATE $R = 3/8$ CODE

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111100</td>
<td>10010110</td>
</tr>
<tr>
<td>01011010</td>
<td>10011001</td>
</tr>
<tr>
<td>01100110</td>
<td>10100101</td>
</tr>
<tr>
<td>01101001</td>
<td>11000011</td>
</tr>
</tbody>
</table>

The power spectrum for the $R = 3/8$ code was calculated using a method described by Immink [57], and is shown in figure 4.4. Notice that the power spectrum has a "dead band" at the Nyquist frequency and that the dc and very low frequencies are completely suppressed.

Figure 4.4 Normalized power spectrum of $R = 3/8$ code example.
In general the number of surviving code words is not a power of two. If this is the case, the code words are discarded in complementary pairs (to eliminate spectral line components), until the number of code words is a power of two. Figure 4.5 shows the normalized power spectra of a rate $R = 5/12$ code. This is not the only possible code for this rate. The code book was chosen from all possible words to facilitate the decoding described in the next section.

![Normalized Power Spectrum](image.png)

**Figure 4.5** Power spectrum of a rate $R = 5/12$ code.

In figure 4.6 we present the rates of the combined spectral shaping subcodes for $n \leq 24$. As can be seen, the rate increases with an increase of code word length. Rate $R = 8/16$ is achieved for code word length 16 which makes it attractive when the present 8-bit (or multiples thereof) computer architecture is considered.

The new class of codes is extremely suitable to replace some of the recording codes currently used such as the Hedeman codes [41]. The Hedeman codes have dc free and minimum bandwidth spectral properties. The new class of codes presented here has better dc suppression than the Hedeman codes and in addition also offers single additive error correcting or triple additive error detecting capabilities, which implies that a single peak shift can also be detected, along with the property that the loss or gain of a single bit due to synchronization errors can be corrected.
Chapter 4

At this stage some clarifying remarks are necessary. It must be stressed that the above mentioned subcodes of the Levenshtein codes do no simultaneously correct a single synchronization error and an additive error. Furthermore, due to the assumption that the beginning and the end of a code word is known to the decoder, the rates presented do not reflect the influence of any scheme whereby this achieved. In recent work [45], a scheme is presented whereby the beginning and end of a code word is determined. This scheme adds a constant number of redundant bits to the code words. For the combined spectral shaping subcodes presented in section 4.3.4, 4 bits need to be added for words of length $n > 4$. Good overall code rates are however still possible for large code word lengths. At present decoding consists of an error detector which classifies the type of error (deletion, insertion, additive error or no error.) A separate decoding algorithm is then applied for each of the error types and a lookup table is used in the final stage of decoding.

Figure 4.6 Information symbols vs. word length of the combined codes.
4.4 Decoding.

4.4.1 Decoding strategy.
In this section we discuss the decoding of the combined spectral shaping Levenshtein subcodes. These codes abound in structure and several elegant decoding methods are possible. The first such method that can be used is the original Levenshtein decoder as described in [21]. This decoder functions under the assumption that the beginning and end of the code word is known, and thus the length of the code word is known. This information is obtained from the first stage of decoding where a framing technique is used to determine the code word boundaries. From the length of the code word it can be determined if the synchronization error occurred in the form of a insertion (longer than transmitted word) or a deletion (shorter than transmitted word). The algorithm in [21] then describes a method whereby the deletion or insertion of a binary zero or one is corrected. The result is the original transmitted word which is then decoded using a lookup table or another method such as combinatorial logic.

Further structure is built into the Levenshtein codes by adding the spectral shaping constraints. The following properties are of importance.

Property 4.1
The code word moment in the unipolar representation of the spectral shaping code words is equal to \((n + 1)n/4\) or:

\[
\sum_{i=1}^{n} ix_i = (n + 1)n/4, \quad x_i \in \{0, 1\}. \tag{4.5}
\]

This property was proven by Immink in [35] and later it was shown [39] that the dc^2 constrained codes of Immink are contained in the balanced Levenshtein subcodes.

Property 4.2
The spectral shaping code books are fully complementary, i.e. if a word \(x\) is a valid code word, then the complement, \(x'\), is also a valid code word.
Proof
Consider the following:

\[
\sum_{i=1}^{n} ix_i + \sum_{i=1}^{n} ix'_i = \sum_{i=1}^{n} i = (n + 1)n/2
\]

But, from (4.5)

\[
\sum_{i=1}^{n} ix_i = (n + 1)n/4,
\]

Thus

\[
\sum_{i=1}^{n} ix'_i = (n + 1)n/2 - (n + 1)n/4 = (n + 1)n/4,
\]

which indicates that \( x' \) is also a valid code word.

Property 4.3
When a code book consisting of complementary code words is subjected to the RAS constraint, the result is still a code book consisting of complementary code words.

Proof
Consider a code word \( y \) in bipolar form. The complement of this code word is \(-y\). If

\[
\sum_{i=1}^{n} (-1)^i y_i = 0, \text{ then}
\]

\[
\sum_{i=1}^{n} (-1)^i (-y_i) = - \sum_{i=1}^{n} (-1)^i y_i = 0
\]

(4.7)
These properties are now used to restore the received word to the original transmitted word in the case of deletions or insertions. Once again it is assumed that the length of the received code word is known and that this information is used to determine if an insertion, deletion or an additive error occurred. The second stage of decoding (thus after the error detection) is done as described in the following section.

4.4.2 Second stage decoding.

4.4.2.1 Additive error.
When the received word is not a valid code word but of the correct length an additive error occurred. In this case the weighted sum is influenced by the single additive error as follows:

\[ \sum_{i=1}^{n} iz_i = (n + 1)n/4 \pm i, \]  

(4.8)

where \( i \) is the position where the additive error occurred and \( z \) the received code sequence. The following syndrome is calculated:

\[ |S| = |n/4\cdot(n + 1) - \sum_{i=1}^{n} iz_i|, \]  

(4.9)

The bit at position \( |S| \) is then complemented to restore the code word.

4.4.2.2 Synchronization error.
If the code word length is \( l = (n + 1) \), an insertion occurred, else if \( l = (n - 1) \), a deletion occurred. The weight of a valid code word must be \( w = n/2 \), thus the number of binary ones is the same as the number of binary zeros. This is used to determine if a one or a zero was inserted or deleted as follows:
If \( l = (n + 1) \) and \( w = n/2 \), then a binary zero was inserted, else a binary one was inserted.

If \( l = (n - 1) \) and \( w = n/2 \), then a binary zero was deleted, else a binary one was deleted.

The following syndrome is calculated for the cases of a deleted or inserted zero:

\[
|S| = |n/4(n + 1) - \sum_{i=1}^{n} i\bar{z}_i|,
\]  

(4.10)

In the case of a deleted zero, \( (S > 0) \), a zero is inserted directly after \( S \) most significant ones. Similarly, in the case of an inserted zero, \( (S < 0) \), a zero is deleted directly after \( S \) most significant ones.

The cases of an inserted or deleted one, are easily handled if it is remembered that the words occur in complementary pairs. If the erroneous received word is complemented, the inserted or deleted one in the received word becomes an inserted or deleted zero in the complemented word. This is then corrected as shown previously and the corrected word is then complemented to restore the transmitted word. Note that this method only holds true for the case when the words occur in complementary pairs. Thus, when setting up a code book with \( 2^k \) code words, it is important to remember to discard redundant words in complementary pairs as well.

4.4.3 Third stage decoding

In the third stage of decoding, the reconstructed code words from stage two are decoded to yield the transmitted data bits. To achieve this, a lookup table or some form of combinatorial logic can be used. In this section we investigate the spectral shaping Levenshtein subcodes as discussed in section 4.3.4 for a more elegant decoder.

Close scrutiny of the rate \( R = 2/4 \) and \( R = 3/8 \) codes reveals that these codes can be made systematic, i.e. the data bits are explicitly stated in certain bit positions. Thus, to decode these words it is only necessary to extract these systematic bits. The question arises whether the spectral shaping Levenshtein subcodes are all systematic, and the answer is given in the following statement:
Statement 4.1
The spectral shaping Levenshtein subcodes of section 4.3.4 are not systematic in general.

Proof
From figure 4.6 it is seen that there exist codes that have a rate $R > 0.5$. For such a code to be systematic there must exist a code word that has a weight $w > n/2$ that could correspond to the maximum weight (all ones) data word of length $l > n/2$. This however contradicts the property that the multipurpose charge constrained Levenshtein subcodes are balanced and hence have weight $w = n/2$. Thus, these codes are not systematic.

4.4.4 An $n = 12$ code example.
The 12-bit codes were investigated next and a semi systematic decoding method was found, where some of the code bits are systematic and the rest can be decoded with simple combinatorial logic.

There are 48 code words that comply to the conditions imposed in section 4.3.4 on the spectral shaping Levenshtein subcodes. The maximum rate that can be achieved, is $R = 5/12$. Thus only $2^5 = 32$ words are needed and the rest must be discarded, being careful to keep the code complementary. There are $\binom{24}{8}$ different codes that can be constructed from these words.

We would like to incorporate the $n = 8$ code words into part of the $n = 12$ code words in such a way that the 3 systematic positions of the 8 bit code are preserved. The remaining four bits must contribute the portion of the code word moment that is not contributed by the 8 bits. It is thus necessary to calculate the code word moments of both the 12 and 8 bit code words, as shown below:

For $n = 12$,
$$
\sum_{i=1}^{12} ix_i = (12 + 1)12/4 = 39,
$$
and for $n = 8$,
$$
\sum_{i=1}^{8} ix_i = (8 + 1)8/4 = 18.
$$
It is now observed that if the 8 bit code words are positioned at the beginning of the 12 bits, the remaining 4 code bits must contribute 21 to the code word moment to achieve the required 39. Furthermore, these 4 bits must have a weight of only 2. There are only two such words, namely, 1001 and 0110. Similarly, it is found for the case of the 8 bit words at the end of the 12 bit words that the remaining 4 bits must contribute 5 to the existing code word moment of 34, and that if the 8 bit words are split into two 4 bit groups, one at the beginning and one at the end of the 12 bit code, the middle 4 bits must contribute 13 to the existing code word moment of 26. In both these last cases it was found that the only 4 bit words able to achieve this are 1001 and 0110. Thus the 32 code words chosen from the 48 available words all contain either 1001 or 0110 in at least one of the 4 bit subdivisions of the 12 bit word. The resulting code book is given in Table 4.2.

| TABLE 4.2 |
| A RATE $R = 5/12$ CODE BOOK |

| 001101101100 | 001110011100 | 0011111000110 | 0011111001001 |
| 010101101010 | 010110011010 | 010110100110 | 010110101001 |
| 011000111100 | 011001011010 | 011001100110 | 011001101001 |
| 011010010110 | 011010100110 | 011010101001 | 011010101010 |
| 100100111100 | 100101011010 | 100101100110 | 100101101001 |
| 100110010110 | 100110100110 | 100110101001 | 100110101010 |
| 101000111010 | 101001011100 | 101001100110 | 101001101001 |
| 110000111010 | 110001011100 | 110001100111 | 110001101011 |

Notice that each 4 bit subdivision of this code is balanced. With the aid of a program that searches for systematic positions, it was found that the code in Table 4.2 has several combinations of four systematic positions, namely positions:

1, 4, 6, 7  
2, 3, 5, 8  
2, 3, 9, 12

Using this information it was found that there at least three possible decoders where the last data bit (bit #5) is given by the addition modulo 2 of two code bits. The complete decoders are given in Table 4.3.
4.5 Semi systematic combined codes

It is possible to construct a class of semi systematic spectral shaping subcodes of the Levenshtein codes by following a similar approach as with the code example in the previous section. The basic idea behind the construction of these codes is to concatenate 4-bit balanced segments in such a way that the resulting code words have all the spectral properties discussed in section 4.3.4, namely a second order zero at 0 Hz and a spectral zero at the Nyquist frequency. The rates that are achieved by this class of codes is given by the following expression:

\[ R = \frac{1}{2} - \frac{1}{n}, \]  

(4.11)

where \( n \) is the length of the code words in the code book. A full discussion of the construction of these codes can be found in Appendix A.

4.6 Summary

The main result of this chapter is the construction of a class of spectral shaping subcodes of Levenshtein's first class of codes. These codes can be used in multiple applications and have a semi systematic decoder which is very easy to implement. The rates of the new codes approach \( R = \frac{1}{2} \) which is the theoretic maximum for systematic balanced block codes. Although this rate appears to be quite low, it must be remembered that the minimum bandwidth constraint on these codes compensate for this deficiency by enabling transmission at twice the rate of a similar bandwidth limited code which does not have \( \text{RAS} = 0 \). Using this technique, timing becomes very important and any jitter could cause the spurious insertion or
deletion of unwanted bits in the recovered code sequence. Such errors can be corrected by these codes because they are subcodes of the single insertion/deletion correcting codes described by Levenshtein. Furthermore, it was found that these codes have a minimum distance $d_{\text{min}} = 4$, which allows for the correction of a single additive error or the detection of three additive errors (and thus also a single peak shift error).

In the next chapter, the investigation of the Levenshtein codes is extended to the case of multiple synchronization errors.
In this chapter codes are presented that are able to detect and correct the deletion or insertion of two adjacent binary bits in a code word of length \( n \). We introduce the symbol \( \alpha \) to denote adjacent insertion or deletion errors. The classes of codes presented include subcodes of the first class of Levenshtein codes as well as two new partition constructions. The \( \alpha = 2 \) subcodes of the Levenshtein codes are presented first.

5.1 The double adjacent synchronization error correcting \((\alpha = 2)\) Levenshtein subcodes.

The following construction technique is used to construct \( \alpha = 2 \) Levenshtein subcodes:

Two length \( n/2 \) Levenshtein codes are interleaved and the resulting code word must also comply to (3.5). It is easily shown that these codes are able to correct a single synchronization error (due to equation (3.5)) or two adjacent synchronization errors.

**Theorem 5.1**

The interleaving of two Levenshtein code words of length \( n/2 \) in such a way that the resulting word is also a Levenshtein code word of length \( n \), results in a code word that is able to correct two adjacent insertions or deletions as well as a single insertion or deletion.
Proof

Consider two valid Levenshtein code words $x$ and $y$. The interleaved code word, $c$, consists of the bits of $x$ in the even numbered positions and the bits of $y$ in the odd numbered positions.

Thus, \[ c = x_1y_1x_2y_2...x_{n/2}y_{n/2}. \]

If two adjacent deletions occur, one bit in both $x$ and $y$ is deleted. By de-interleaving the received word, both the received $x$ and $y$ can be decoded using the Levenshtein algorithm. The same holds true for two adjacent insertions, which proves the theorem. It should be noted that when an insertion and a deletion occur next to each other, the effect is the same as an additive error, (in the worst case). The minimum Hamming distance of these codes is $d_{\text{min}} = 2$. These codes are not able to correct a single additive error and by inference, the case of a deletion and an insertion error next to each other, but are able to detect such an error. The codes are able to correct a single insertion or deletion because they are subcodes of the first class of Levenshtein codes. Once again it is assumed that the beginning and end of a code word is known.

It should be noted that double adjacent insertion/deletion correcting ($\alpha = 2$) subcodes of the Levenshtein codes are known, [29]. The construction shown above has a much simpler proof than the construction in [29]. In fig. 5.1 we compare the rates of the new construction with the construction presented in [29].

![Figure 5.1: Code rates for $\alpha = 2$ Levenshtein subcodes.](image-url)
5.2 A double adjacent synchronization error code partition.

In this section a partition of all binary sequences of length \( n \) is presented. Each one of these partitions forms a code that is able to correct a single deletion or two adjacent deletions.

**Theorem 5.2**

The following equation describes a class of codes which are able to correct two adjacent deletions:

\[
\sum_{i=1}^{n} i x_i + \sum_{i=1}^{n-1} (n-i)x_{i+1} = a \mod (3n),
\]

where \( x_i \in \{0, 1\} \) and \( n \) is the length of the code word \( x \), and \( a \) is an integer corresponding to the partition of the space of integers. The proof of the error correcting capabilities of the code is presented in Appendix B, section B.1. and consists of showing that if a binary word \( x \) from a partition is subjected to any two adjacent deletions at positions \( i \) and \( i+1 \) and then to any two insertions at positions \( j+1 \) and \( j+2 \), then the resulting word \( y \) is not in the same partition as \( x \) for all values of \( i \) and \( j \). Thus, the codes have a Levenshtein distance \( \ell_a(x,y) > 2 \), where \( \ell_a(x,y) \) denotes the Levenshtein distance for adjacent errors.

**Theorem 5.3**

Every partition described by (5.1) can detect two adjacent additive errors.

By considering all possible cases of two adjacent additive errors which change a word \( x \) into a word \( y \), it is proven in appendix B that the resulting word \( y \) is not in the same partition as the original word \( x \). It is however possible that the same resulting word is obtained from two different original words. This implies that the error can be detected but not corrected. It should be noted that the case of a single insertion and a single deletion in adjacent positions can result at the worst in a single additive error. Because the code described in (5.1) has a minimum distance \( d_{min} = 2 \), this error can be detected but not corrected. Thus, the codes described by (5.1) can correct two adjacent insertions or two adjacent deletions, but not a single insertion and a single deletion at adjacent positions.
By careful observation of the values which $\sigma_x - \sigma_y$ can assume in the proof of theorem 5.3 given in appendix B, section B.1, it is found that the following equation also describes partitions of the $2^n$ words that are able to correct either two adjacent deletions or two adjacent insertions, but not a combination of adjacent insertions and deletions:

**Theorem 5.4**

Every partition described by (5.2) can correct two adjacent insertions or two adjacent deletions.

$$\sum_{i=1}^{n} ix_i + \sum_{i=1}^{n-1} (n-i)x_i x_{i+1} = a \mod (2n+1), \quad (5.2)$$

The proof is similar to that of equation (5.1), but because the maximum value of $\sigma_x - \sigma_y > (2n+1)$, it is necessary to consider all the possible values which $\sigma_x - \sigma_y$ can assume.

Because of the length and tedious nature of the proof, it is also presented in appendix B. From appendix B it is found that $\sigma_x - \sigma_y \neq 0 \mod (2n+1)$. These codes have the same error correcting capabilities as the codes described in (5.1), but have larger cardinalities. A comparison of the codes described in this chapter is given in figure 5.2. These cardinalities were determined by selecting the partition with the largest cardinality for a certain word length $n$. A computer was used to construct all the partitions for every construction and to count the words per partition. From figure 5.2 the significant difference between the code rates of the interleaved codes and the partition codes can be seen. Note that rate $R = 1/2$ is already achieved at $n = 8$ for the second class of $a = 2 \ (\mod (2n+1))$ codes. An interesting topic for further investigation would be the determination of an analytical description of the cardinality of these codes.

Using a completely different approach, Levenshtein [26] constructed asymptotically optimal codes that are able to correct a single insertion/deletion or two adjacent insertions/deletions. These codes were constructed by first defining two functions on the "runs" of like symbols in a code word and then applying a modulus operator to partition the state space into code books. After this procedure, it is necessary to append a binary zero to each code word. When comparing this class of codes to the class of codes described by theorem 5.4, it is seen that the new class of codes are easier to construct. Furthermore, due to the appended binary zero, the rates of the two classes of codes approach the same value for large code word lengths.
5.3 Corroborating results

Due to the complex and tedious nature of the proofs of the theorems presented in section 5.2, it is preferable to corroborate the results using another method, if possible. Another way of testing the construction, is to generate the code books and to subject these code books to exhaustive tests to such an extent that the possibility of error becomes negligible. To this end, all the code books of lengths $n = 4$ to $n = 10$ were subjected to an exhaustive subword test. This test entails obtaining all possible subwords of every code word after $a = 2$ deletions and comparing the subwords of every code word to the subwords of every other code word. A valid code is a code where no subwords of a code word is also the subword of another code word. The following example illustrates such a test on a $R = 4/8, n = 8$ code obtained from the second class of $a = 2, (\mod (2n + 1))$, codes:

5.3.1 Example
Consider the following code book with 16 code words, along with the respective subwords after $a = 2$ adjacent deletions:

\begin{verbatim}
00101000 001010 001000 101000
01000100 010001 010000 010100 000100
11100100 111001 111000 111100 110100 100100
01101100 011011 011010 011000 011100 001100 101100
10000100 100000 100010 000010
11000110 110001 110000 110010 110110 100110 000110
01001110 010011 010010 010110 011110 001110
11101110 111011 111010 111110 101110
00000001 000000 000001
01100101 011001 011101 010101 000101 100101
11110101 111101 111010
00101101 001011 001001 001101 011010 101101
10101011 101010 101011
10000111 100001 100011 100111 000111
11001111 110011 110111 111111 101111 001111
01011111 010111 011111
\end{verbatim}
This code is a valid double adjacent deletion correcting code because every subword is unique to only one specific code word.

The same test was applied to the other 16 $n = 8$ code books, as well as to all the code books for $n = 4$ to $n = 10$. In every case the code books successfully passed the test outlined above. We conclude that these results corroborate the claims of theorem 5.4. A similar test was applied to the class of codes described in theorem 5.2, with the same results. The program used for these tests can be found in Appendix C, along with a short description and instructions for use.

Another interesting exhaustive test that was applied to the first and second class of double adjacent deletion correcting codes is a test for random insertion/deletion correcting capabilities. As above, all the possible subwords after $s$ random deletions in a code word are found. Every subword found in this way must be unique to a code word for a code to be able of correcting the $s$ insertions or deletions. Using this exhaustive test, it was found that the first and second class of double adjacent insertion/deletion correcting codes are also able to correct a single insertion or deletion, i.e. they are $s = 1$ correcting codes. These codes are not subcodes of the Levenshtein codes and at present constitute the only other number theoretic class of codes that are able to correct a single insertion or deletion. This exhaustive insertion/deletion correcting test is illustrated on the same code used in example 5.3.1.

5.3.2 Example
Consider the following rate $R = 4/8 \ a = 2$ correcting code, along with all possible subwords after a single deletion:

```
00101000 0010100 0010000 0011000 0001000 0101000
01000100 0100010 0100000 0100100 0000100 1000100
11100100 1110010 1110000 1110100 1100100
01101100 0110110 0110100 0111100 0101100 1101100
10000010 1000001 1000000 1000000 0000010 0000010
11000110 1100011 1100010 1100110 1000110
01001110 0100111 0100110 0101110 0001110 1001110
11101110 1110111 1110110 1111110 1101110
```
Note, once again, that the subwords are unique to only one code word. The code is thus able to correct a single insertion or deletion.

This test was applied to all the code books for \( n = 4 \) to \( n = 10 \). In every case the code books successfully passed the test outlined above. The program used for these tests can be found in Appendix D, along with a short description and instructions for use.

In the next chapter the structure of codes that are able to correct more than one random synchronization error is investigated. From this knowledge bounds on the cardinality of such codes are derived and a general construction technique is presented.

\[
\begin{align*}
00000001 & \quad 0000000 0000001 \\
01100101 & \quad 0110010 0110011 0110001 0110101 0100101 1100101 \\
11110101 & \quad 1111010 1111011 1111001 1111101 1111011 1111011 \\
00101101 & \quad 0010110 0010111 0010101 0011101 0001101 0101101 \\
10101011 & \quad 1010101 1010111 1010011 1011011 1001011 1101011 0101011 \\
10000111 & \quad 1000011 1000111 0000111 \\
11001111 & \quad 1100111 1101111 1001111 \\
01011111 & \quad 0101111 0111111 0011111 1011111
\end{align*}
\]

**Figure 5.2** Cardinalities of double adjacent deletion/insertion correcting codes.
In this chapter we present several results pertaining to codes that are capable of correcting more than one synchronization error. The investigation of the Hamming weight distance structure of the Levenshtein codes is extended to the case of double synchronization error correcting codes. From this structure, bounds on the cardinality of the codes are derived which improve on the known bounds. A construction method is given for such codes. Using a similar approach, the structure of multiple insertion/deletion correcting codes are investigated and from which bounds and a general construction technique are derived.

6.1 Number theoretic double synchronization error correcting (s = 2) codes.

In this section we extend our investigation of the properties of the Levenshtein codes to cover the case of two random synchronization errors per word. These new properties are then used to determine new upper and lower bounds on the cardinality of codes that have these properties. Some useful guidelines to the construction of such codes are then used to construct s = 2 codes.

6.1.1 Properties of number theoretic s = 2 codes.

In this section it is assumed that a code is a valid s = 2 correcting code. Then the following propositions describe certain characteristics of such a code.
**Proposition 6.1**
There can only be one code word of weight $w = 1$ in a valid $s = 2$ code book.

**Proof**
If there could be two valid code words of weight $w = 1$, then it is always possible that a deletion in each word of the binary ones result in the same (all zero) subword for each word. This is not possible in a valid $s = 2$ or $s = 1$ code according to the Levenshtein criterion.

**Inverse**
There can only be one valid code word of weight $w = n - 1$ per code book.

**Proof**
As in proposition 6.1 with the roles of the binary ones and zeros interchanged.

**Proposition 6.2**
There can only be one valid code word with weight $w = 2$ in a code book.

**Proof**
If there could be two valid code words of weight $w = 2$, then it is always possible that 2 deletions in each word on the binary ones result in the same (all zero) subword for each word. This is not possible in a valid $s = 2$ code according to the Levenshtein criterion.

**Inverse**
There can only be one valid code word of weight $w = n - 2$ per code book.

**Proof**
As in proposition 6.2 with the roles of the binary ones and zeros interchanged.

**Proposition 6.3**
If a code book has as lowest weight code word, the word of weight $w = 0$ (i.e. the all zero code word), then the code word of next lowest weight must have $w > 2$. 
Proof
If the next lowest weight word has \( w = 2 \) then it is always possible that 2 deletions may result in a subword of weight \( w = 0 \). The weight \( w = 0 \) word also has a subword of weight \( w = 0 \) after 2 deletions, thus the Levenshtein metric \( \varphi(x,y) < 4 \). Thus the next lowest weight word must have a weight \( w > 2 \) in a valid \( s = 2 \) code.

Inverse
If the highest weight code word has weight \( w = n \) then the next highest weight code word that is valid has a weight \( w < n - 2 \).

Proof
As in proposition 6.3 with the roles of the binary ones and zeros interchanged.

Proposition 6.4
Words of weights \( w = 0, 1, 2 \) do not occur together in a valid \( s = 2 \) code book.

Proof
a) From proposition 6.1 it is clear that words of weight \( w = 1 \) or \( 2 \) do not occur with the weight \( w = 0 \) word in a valid \( s = 2 \) code.
b) It is always possible to delete the binary one in the weight \( w = 1 \) word along with an arbitrary zero to obtain the all zero subword. By deleting both the binary ones in the weight \( w = 2 \) word the all zero subword is always obtained. The subwords after \( s = 2 \) deletions and/or insertions must differ in a valid \( s = 2 \) code, thus words of weight \( w = 1,2 \) cannot occur together in a valid \( s = 2 \) code.

Inverse
Words of weight \( w = (n - 1, 2, 3) \) do not occur together in a valid \( s = 2 \) code.

Proof
As in proposition 6.4 with the roles of the binary zeros and ones interchanged.

Proposition 6.5
The minimum Hamming distance between words of a valid \( s = 2 \) code is 3.
Proof
If the Hamming distance between any two words were equal to 2, then it is always possible to obtain the same subword from these two words by $s = 2$ deletions of the binary symbols in the positions where Hamming distance is built in each word. This is not possible in a valid $s = 2$ code, thus the minimum Hamming distance between any two code words in a valid $s = 2$ code is 3.

We next consider a specific class of insertion/deletion correcting codes which we prefer to call number theoretic insertion/deletion correcting codes, similar to the $s = 1$ code proposed by Levenshtein. These codes are partitions of the $n$ space of binary sequences into distinct subsets by associating a certain value to the indices of the binary ones in the sequence. When considering the $s = 1$ codes, it is found that these values are unique because only one word of weight $w = 1$ is allowed per partition, (c.f. chapter 3). The total code word moment is then unique to a valid $s = 1$ code partition. The following propositions describe some of the properties of $s = 2$ number theoretic codes. These properties are similar to the properties of the $s = 1$ Levenshtein codes as described in chapter 3 and are better understood if viewed in conjunction with section 3.4 of chapter 3.

Proposition 6.6
Any two words of the same weight in a valid $s = 2$ code have a minimum Hamming distance of 6.

Proof
It is known that the Hamming distance of constant weight code words is a multiple of 2. Consider any two different constant weight code words of which $w - 1$ binary ones correspond in position. The minimum Hamming distance between the code words is built by the binary one in each word that does not correspond in position to any binary one in the other word, thus the Hamming distance between such words is 2. In proposition 6.5 it was proved that the minimum Hamming distance between code words of a valid $s = 2$ code is three, therefore the above words are not valid $s = 2$ code words.

Consider any two different constant weight code words of which $w - 2$ ones correspond in position. These code words have a Hamming distance of 4. According to the number theoretic construction, the code word moment (modulo $u$) of such words is the same. By replacing the binary ones which correspond, by binary zeros, we find that we have two $w = 2$, words of which the weighted sum (modulo $u$) is still the same. This is in violation of
proposition 6.2, thus such words are not valid $s=2$ code words. When considering the case for which $w-3$ ones do not correspond, we find that the Hamming distance is 6. By replacing the corresponding binary ones with zeros, we find that there are two $w=3$ words that have the same weighted sum (modulo $u$). This is permissible, thereby proving the proposition.

**Proposition 6.7**
Valid number theoretic $s=2$ code words that differ in one unit if weight have a minimum Hamming distance of 5.

**Proof**
Consider any word of weight $w=x$ and any other word of weight $w=x+1$ such that $x$ binary ones correspond in position. The Hamming distance between these words is 1 and thus they cannot be valid $s=2$ correcting code words according to proposition 6.5. Consider any word of weight $w=x$ and any other word of weight $w=x+1$ such that $x-1$ binary ones correspond in position. The Hamming distance of these words is 3 thus they could be permissible according to proposition 6.5. Consider the contribution to the code word moment of the bits that do not correspond. These two code words must have the same code word moment to be in the same partition, thus the positions where the Hamming distance is built contribute the same amount to the code word moment. Thus the word of weight $w=x$ has two elements which contribute the same amount to the code word moment as the single element of the word of weight $w=x+1$. It is then possible that two other words, of weight $w=2$ and $w=1$ respectively, which have binary ones at these positions, are in the same partition. This, however, violates proposition 6.4, proving proposition 6.7.

**Proposition 6.8**
Valid number theoretic $s=2$ code words that differ in two units of weight have a minimum Hamming distance of 4.

**Proof**
Consider any binary word of weight $w=x$ and any other binary word of weight $w=x+2$ such that $x$ binary ones of each word correspond in position. These words have a Hamming distance of 2 and are thus not valid $s=2$ code words according to proposition 6.5. Next consider any binary word of weight $w=x$ and any other binary word of weight $w=x+2$ such
that \( x - 1 \) binary ones of each word correspond in position. These words have a Hamming distance of 4 and are thus permissible code words according to proposition 6.5.

The properties discussed above are easily summarized as shown in figure 6.1.

---

Figure 6.1 Schematic diagram showing the relationship between Hamming distance and weight for number theoretic \( s = 2 \) correcting codes.
6.2 Bounds on the cardinality of number theoretic $s = 2$ correcting codes.

The properties if the $s = 2$ number theoretic codes as described by propositions 6.1 to 6.8 and fig. 6.1 are now used to derive improved upper bounds on the cardinality of the number theoretic $s = 2$ correcting codes. First we discuss the upper bound.

6.2.1 Upper bound

Close scrutiny of fig. 6.1 reveals that an upper bound on the cardinality can be found by enumerating the number of constant weight code words that have a certain minimum distance. Work of this type is extensively discussed in [42] and several tables as well as bounds are given. The following upper bound on the cardinality is then found:

$$|C| < 2 + \sum_{w=3}^{n-3} (n/w \cdot A(n-1,w-1))$$  \hspace{1cm} (6.1)

where $w$ is the weight of the code word of length $n$ and $A(n,d,w)$ is the number of code words of weight $w$, which differ from each other in at least $d$ positions (i.e. have a minimum Hamming distance of $d$). The expression in (6.1) is due partly to Johnsson [32] who derived a recursive upper bound for constant weight codes with a certain minimum Hamming distance. Notice that the bound in (6.1) is found by adding the number of all constant weight code words that have a minimum Hamming distance of $6$. Due to proposition 6.4, code words of weights $3$ up to $(n - 3)$ are included in the code book, in addition to 2 words of weight less than $2 + 1$ and more than $n - 3$ respectively. General bounds (thus not only for the case of number theoretic $s = 2$ codes) on the cardinality of $s$ - codes were found by Bours [30], from which the upper bound for $s = 2$ codes is given in (6.2):

$$|C| \leq 2^n \sum_{j=0}^{1} \binom{n}{j}$$  \hspace{1cm} (6.2)

where $n$ is the length of the code word. From figure 6.1 it is also noted that an upper bound on the cardinality of $d_{\text{min}} = 5$ code books, is also an upper bound on the cardinality of $s = 2$ code books. Using this approach, it is possible to use a Hamming bound for $d_{\text{min}} = 5$ codes to determine an upper bound on the cardinality of $s = 2$ code books:
A lower bound on the cardinality of number theoretic $s = 2$ codes was found by Bours [30] and is given in (6.4):

$$|C| \geq 2^n / \sum_{j=0}^{2} \binom{n}{j}.$$  

(6.4)

The new bounds from fig. 6.1 are compared to bounds (6.2) and (6.4) in fig. 6.2. From this figure it is seen that the new upper bounds are much tighter than the known bounds.

Figure 6.2 Bounds on the cardinality of number theoretic $s = 2$ correcting codes.
6.3 A number theoretic $s = 2$ correcting code.

With the use of propositions 6.1 to 6.8, we show that it is possible to construct a number theoretic code that can correct up to two deletions and/or insertions. The number theoretic construction has the following form:

$$\sum_{i=1}^{n} v_i x_i \equiv a \mod (u), \quad (6.5)$$

where $x = (x_1, x_2, ..., x_n), x_i \in \{0, 1\}; n$ is the length of the code word $x$, $a$ is an arbitrary integer and $v$ is a vector containing the weights by which each element of $x$ is weighted in the code word moment. It now remains to be determined what the value of $u$ is, as well as $v$. The value $v$ is determined by considering proposition 6.4. According to this proposition words of weight $w = 0, 1, 2$ do not occur together in a $s = 2$ code. This means that $v$ must comply to the following rules:

**Rule 6.1**
The elements of $v$ must be unique, i.e. an element may occur only once in $v$.

**Proof**
If the same element occurs at positions $i$ and $j$, then there exists two words of weight $w = 1$, where the binary ones are at positions $i$ and $j$, of which the code word moments are the same. These words are then in the same code book. This contravenes proposition 6.1, thus an element may occur only once. Note that this case extends to where an element mod $(u)$ is the same as another element.

**Rule 6.2**
The sum of two elements may not be the same as the sum of two other elements mod$(u)$, or be the same as a single element mod$(u)$ or be equal to zero mod$(u)$.

**Proof**
a) If $v_i + v_j = v_l + v_m$, then there exists two words of weight $w = 2$ with binary ones at $i, j$ and $l, m$ respectively, of which the weighted sum is the same. These words are then in the same code book, which contravenes proposition 6.2.
b) If \( v_i + v_j = v_i' \), then there exists two words of weight \( w = 2 \) and weight \( w = 1 \) respectively, with binary ones at \( i, j \) and \( l \) respectively. The weighted sum of these words is the same. These words are then in the same code book, which contravenes proposition 6.4.

c) If \( v_i + v_j \equiv 0 \mod(u) \) then there exists a word of weight \( w = 2 \) with binary ones at \( i, j \) of which the weighted sum is the same (zero) as the all zero word. These words are then in the same code book, which contravenes proposition 6.4.

Diligent search for such a vector \( v \) eventually revealed that a modified Fibonacci sequence of appropriate length complies to the above rules. It should be noted that the choice of vector \( v \) and the value of \( u \) is related. The search is simplified by specifying that \( u \) is greater than the maximum valued element of \( v \).

The following general construction for number theoretic \( s = 2 \) correcting codes is proposed using the information above.

**Theorem 6.1**

Every partition of the \( 2^n \) binary words according to the partition (6.6), is capable of correcting up to two random insertions or deletions consisting of deletions and insertions of symbols.

\[
\sum_{i=1}^{n} v_i x_i \equiv a \mod(u),
\]

(6.6)

where

\[
v_i = v_{i-1} + v_{i-2} + 1,
\]

\[v_1 = 1, \ v_2 = 2,\]

and

\[u = v_{n-1} + v_n + 1.\]

The proof consists of considering all possible combinations of deletions and proving that the resulting subword is can only be formed from one of the code words. The case for the insertions is proven by recalling that Levenshtein [22] proved that if a code can correct \( s \) deletions, it can also correct \( s \) deletions and insertions.
Proof
Consider any code word that complies to (6.6). A deletion at position $i$ will diminish the code word moment in (6.6) by at most $v_n$. This is seen as follows:

Consider a code word consisting of binary ones. If a deletion occurs at position $i$, the resulting subword will be the sequence of length $n - 1$ consisting of binary ones. The difference between the code word moment of this subword and the original code word will then be the value of $v_n$. Any other code word has a lower weight than the all ones code word and thus the code word moment cannot differ by more than $v_n$ in the case of a single deletion.

Similarly, if two deletions occur at any position $i$ and $j$ respectively, the resulting code word moment differs at most by $(v_n + v_{n-1})$ from the original code word moment. This is demonstrated by the fact that two deletions in the maximum weight sequence (all binary ones) diminishes the length by two, and the code word moment sum is diminished by the value of the two highest valued elements of $v$, namely $(v_n + v_{n-1})$. In both the cases of single or double deletions, the difference in the code word moment between the original code word and the subword is less than the value of $u$, where $u = v_n + v_{n-1} + 1$. This means that, due to the constraints upon $v$, (rules 6.1 and 6.2), single or double deletions do not cause the code word moment of any subword to be the same as the code word moment of any other subword. This proves that any two deletions in any two code words in one partition, do not result in the same subwords. The code is able to decode any sequence after $s = 2$ deletions, and thus also any combination of two synchronization errors consisting of deletions and/or insertions.

In table 6.1 the rates of the double synchronization error correcting codes described in (6.6) are given for short code word lengths. The code rates in table 6.1 are then compared to the bounds derived by Bours [30] and the new bounds described in section 6.2. The results of this comparison is shown in figure 6.3.
### TABLE 6.1

CODE RATES OF NUMBER THEORETIC DOUBLE SYNCHRONIZATION ERROR CORRECTING CODES.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Residue ((a))</th>
<th>Rate ( R )</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>1/3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1/4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1/5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1/6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>2/7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>2/8</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>2/9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>66</td>
<td>3/10</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>65</td>
<td>3/11</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>65</td>
<td>3/12</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>297</td>
<td>3/13</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>297</td>
<td>4/14</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>1283</td>
<td>4/15</td>
<td>30</td>
</tr>
</tbody>
</table>

**Figure 6.3** A comparison of \( s = 2 \) code cardinalities and \( s = 2 \) bounds
6.4 General $s$ - insertion/deletion correcting number theoretic codes.

Using the principles described in section 6.1 and 6.3, it is now possible to construct number theoretic codes that are able to correct any number of synchronization errors. The same approach as previous is followed, by first considering the weight/distance relationships of number theoretic $s$ - insertion/deletion correcting codes and then extending these relationships to achieve a general number theoretic $s$ - insertion/deletion correcting code construction.

6.4.1 Weight/distance structure of $s$ - insertion/deletion correcting number theoretic codes.

The following propositions describe certain characteristics of number theoretic codes that are able to correct up to $s$ synchronization errors.

*Proposition 6.9*

Code words of weights $w = 0, 1, 2, \ldots, s$ do not occur together in a valid $s$ - correcting code.

*Proof*

Because it is possible that up to $s$ deletions may occur, any two words of weight $w < s + 1$ may have the same all zero subword by deleting the binary ones of the code words. This means that the code words are not uniquely decodable and cannot correct $s$ synchronization errors. Thus these words do not occur together in a valid $s$ - insertion/deletion correcting code book.

*Proposition 6.10*

The minimum Hamming distance of a $s$ - insertion/deletion correcting code is $d_{\text{min}} = s + 1$.

*Proof*

If two words have a minimum distance $d_{\text{min}} < s + 1$, then it is possible that $s$ deletions occur at the positions in each word where the Hamming distance is built, resulting in the same all zero subword. These words are thus not decodable and cannot occur together in a valid $s$ - insertion/deletion correcting code.

It should be noted that propositions 6.9 and 6.10 are valid for all $s$ - insertion/deletion correcting codes. The following proposition is only valid for number theoretic $s$ - correcting codes which have the following form:
\[
\sum_{i=1}^{n} v_i x_i \equiv a \mod (u),
\]

where \( v \) is a vector containing the weights by which each position of the code sequence is weighted and \( u \) is an integer to be determined from the values in \( v \). The integer \( a, a \in \{0, 1, ..., u - 1\} \), corresponds to each code partition.

**Proposition 6.11**

The Hamming distance between valid \( s \) - correcting code words which differ in weight \( \Delta w = i, i = \{0, 1, ..., s\} \) have a minimum Hamming distance of \( d_{\text{min}} = 2(s + 1) - i \).

**Proof**

Consider two valid \( s \) - correcting code words which differ in weight \( \Delta w = i, i = \{0, 1, ..., s\} \). Let \( x \) be an integer denoting the number of positions in which the binary ones in each word correspond. Let \( w_1 \) denote the weight of the minimum weight word in the code book and let \( w_2 \) denote the weight of the maximum weight word in the code book. The Hamming distance between these two words is then calculated by:

\[
d_{\text{min}} = (w_2 - x) + (w_1 - x)
= w_2 + w_1 - 2x
= w_2 + w_2 - \Delta w - 2x
= 2(w_2 - x) - \Delta w
\]

where \( (w_2 - x) \) signifies the contribution of those elements of the maximum weight word that do not correspond to the code word moment as described in (6.7). Thus, by ignoring the \( x \) binary ones that correspond in each word, it is found that the remaining binary ones must have a certain Hamming distance. According to proposition 6.10, this Hamming distance must be at least \( s + 1 \). Because \( \Delta w = i \), equation (6.7) can now be rewritten as follows:

\[
d_{\text{min}} \geq 2(s + 1) - i,
\]
which proves proposition 6.11. The weight-distance relationships of $s$-correcting codes can be summarized as shown in figure 6.4.

Figure 6.4: Weight-distance structure of number theoretic $s$-correcting codes.
6.5 Bounds on the synchronization error correcting capability.

Using propositions 6.9 to 6.11, it is possible to derive upper and lower bounds on the synchronization error correcting capability of any code book. The upper bounds are investigated first.

6.5.1 Upper bound
From proposition 6.10 it is found that $d_{\text{min}} \geq s + 1$. Then an upper bound on the $s$ - insertion/deletion correcting capability of any code book is given by:

$$s \leq d_{\text{min}} - 1.$$  (6.10)

For the number theoretic codes, it is also noticed that

$$\Delta w_{\text{min}} + d_{\text{min}} \geq 2s + 1,$$

where $\Delta w_{\text{min}}$ denotes the minimum weight difference between any two code words in the code book under consideration. Then an upper bound on the $s$ - insertion/deletion correcting capability of a number theoretic code book is given by:

$$s \leq \left\lfloor \frac{\Delta w_{\text{min}} + d_{\text{min}} - 1}{2} \right\rfloor.$$  (6.11)

6.5.2 Lower bound
From proposition 6.10 a lower bound on the $s$ - insertion/deletion correcting capability of a code is derived by noticing that the common subword of two words that differ in weight by $\Delta w$, must have a weight less than or equal to the weight of the maximum weight word minus the weight difference. The deletions that occur must at least cancel the weight difference between the words. The lower bound is then given by:

$$s \geq \Delta w_{\text{min}} - 1,$$  (6.12)

where $\Delta w_{\text{min}}$ denotes the minimum weight difference between any two words in the code book under consideration.
6.6 A general number theoretic $s$ - insertion/deletion correcting construction

In this section a number theoretic construction is given that divides the $n$ - space into partitions that are all able to correct up to $s$ synchronization errors. The number theoretic construction has the form of (6.7) and the following rules on the value of the vector $v$ are determined from the propositions in section 6.4.1.

**Rule 6.3**

The elements of $v$ must be unique, i.e. an element may occur only once in $v$.

**Proof**

If the same element occurs at positions $i$ and $j$, then there exists two words of weight $w = 1$, where the binary ones are at positions $i$ and $j$, of which the weighted sum are the same. These words are then in the same code book. This contravenes proposition 6.9, thus an element may occur only once. Note that this case extends to where an element mod $(u)$ is the same as another element.

**Rule 6.4**

The sum of up to $s$ elements may not be the same as the sum of up to $s$ other elements mod$(u)$, or be the same as a single element mod$(u)$ or be equal to zero mod$(u)$.

**Proof**

If

$$\sum_{i=1}^{m} v_{i} \mod (u) = \sum_{j=1}^{s} v_{j} \mod (u), \forall i \neq j, l, m \leq s,$$

then there exists two words of weight $w \leq s$ with binary ones at all positions $i$ and $j$ respectively, of which the weighted sum is the same. These words are then in the same code book, which contravenes proposition 6.9.

It should be noted that the choice of vector $v$ and the value of $u$ is related. The following general construction for number theoretic $s$-correcting codes is proposed using the information contained in propositions 6.9 to 6.11 and rules 6.3 and 6.4:
Theorem 6.2
Every partition of the $n$ space words according to the partition (6.14), is capable of correcting up to $s$ synchronization errors consisting of deletions and insertions of symbols.

$$\sum_{i=1}^{n} v_{i} x_{i} = a \mod (u)$$

(6.14)

where

$$v_i = 1 + \sum_{j=1}^{s} v_{i-j}$$

$$v_i = 0 \text{ for } i \leq 0,$$

$$s - 1$$

and

$$u = 1 + \sum_{j=0}^{s} v_{n-j}$$

The proof consists of considering all possible combinations of deletions and proving that the resulting subword can only result from one code word after $s$ deletions. The case for the insertions is proven by recalling that Levenshtein [22] proved that if a code can correct $s$ deletions, it can also correct $s$ deletions and insertions.

Proof
Consider any code word that complies to (6.14). Any $s$ deletions will diminish the code word $s-1$ moment in (6.14) by at most $\sum_{j=0}^{s} v_{n-j}$. This is seen as follows:

Consider the code word consisting of binary ones. If a deletion occurs at position $i$, the resulting subword will be the subword of length $n-1$ consisting of binary ones. The difference between the code word moment of this subword and the original code word will then be the value of $v_n$. Any other sequence has a lower weight than the all ones code word and thus the code word moment cannot differ by more than $v_n$ in the case of a single deletion.

This case is extended such that if $s$ deletions occur at any position in the all ones code word, the code word is shortened by $s$ symbols, and the code word moment is diminished by the value of the $s$ highest valued elements of $v$. The difference in the code word moment of the original code word and the subword is less than the value of $u$. This means that, due to the
constraints upon \( v \), (rule 6.3), up to \( s \) deletions do not cause the code word moment of the subword to be the same as the code word moment of any other subword. This proves that up to \( s \) deletions in any two code words in one partition, do not result in the same subword.

The code is able to decode any sequence after \( s \) deletions, and thus also any combination of \( s \) synchronization errors consisting of deletions and/or insertions.

These general \( s \) - insertion/deletion correcting codes are also able to detect or correct a certain number of additive errors. From proposition 6.10 it is found that the minimum Hamming distance of these codes is \( d_{min} = s + 1 \). A code can correct up to \( t \) additive errors if \( d_{min} \geq 2t + 1 \). Thus, if a code can correct \( s \) synchronization errors, it can correct \( t \) additive errors if \( s \geq 2t \).

It should be noted that for the \( s = 1 \) case, the general construction in (6.14) results in the same construction as described by Levenshtein for the correction of single synchronization errors. In figures 6.5 to 6.7, the rates of \( s = 3, 4, \) and \( 5 \) insertion/deletion correcting codes respectively are compared to the bounds for such codes, as described in [30].

![Figure 6.5 A comparison between cardinality and bounds for a \( s = 3 \) code.](image-url)
Figure 6.6 A comparison between cardinality and bounds for a $s = 4$ code.

Figure 6.7 A comparison between cardinality and bounds for a $s = 5$ code.
6.7 Corroborating results

To inspire confidence in the theorems presented in section 6.6, it is preferable to corroborate the results using another method, if possible. Another way of testing the construction, is to generate the code books and to subject these code books to an exhaustive test. This test entails obtaining all possible subwords of every code word after \( s \) deletions and comparing the subwords of every code word to the subwords of every other code word. A valid code is a code where no subwords of a code word is also the subword of another code word. The following example illustrates this test on a \( n = 8 \) code capable of correcting \( s = 3 \) deletions/insertions from the class of codes described in theorem 6.2.

6.7.1 Example

Consider the following code book with 2 code words, along with the respective subwords after all possible combinations of \( s = 1, 2 \) and \( 3 \) deletions:

After \( s = 1 \) deletion:

<table>
<thead>
<tr>
<th>Code Word</th>
<th>Subwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111001</td>
<td>011110 011110 011101 111101 111101</td>
</tr>
<tr>
<td>10000101</td>
<td>100001 100001 100001 100010 000010</td>
</tr>
</tbody>
</table>

After \( s = 2 \) deletions:

<table>
<thead>
<tr>
<th>Code Word</th>
<th>Subwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000101</td>
<td>100101 000101 000001 100011 000011 100001 100000 100010 000010</td>
</tr>
<tr>
<td>01111001</td>
<td>011001 111001 011111 011101 111101 011110 011100 111100</td>
</tr>
</tbody>
</table>

After \( s = 3 \) deletions:

<table>
<thead>
<tr>
<th>Code Word</th>
<th>Subwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111001</td>
<td>11110 01110 01111 11111 11110 11101 01100 01101 01001 11001</td>
</tr>
<tr>
<td>10000101</td>
<td>10000 00000 00010 00011 00001 10010 10011 10001 10101 00101</td>
</tr>
</tbody>
</table>

After \( s = 4 \) deletions:

<table>
<thead>
<tr>
<th>Code Word</th>
<th>Subwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111001</td>
<td>1101 0101 1011 1001 1010 0011 0001 0010 1000 0000</td>
</tr>
<tr>
<td>10000101</td>
<td>0001 1001 0101 0100 1101 1100 0111 0110 1111 1110</td>
</tr>
</tbody>
</table>

From the above example it is seen that the subwords of each code word do not occur as a subword of any other code word for up to \( s = 3 \) deletions. However, when \( s = 4 \) deletions occur, it is seen that several subwords occur at both the code words. Thus, the code is not
able to correct $s = 4$ deletions/insertions. The same test was applied to all the code books of word lengths $n = 4$ to $n = 16$ for $s = 2$, word lengths $n = 4$ to $n = 15$ for $s = 3$ and word lengths $n = 4$ to $n = 14$ for $s = 4$ and $s = 5$. Overall, 91217 code books were tested with a hundred percent success rate for the rated insertion/deletion correcting capability of the codes as given in theorem 6.2.

In the next chapter, the results of the work presented in this thesis is discussed. Some open problems for future research are also presented, along with a few general remarks concerning the applicability of the new classes of codes that have been presented.
CHAPTER

SEVEN

CONCLUSIONS AND OPEN QUESTIONS

7.1 Overview of new contribution

The main purpose of this study was to investigate coding techniques as a method of controlling and correcting synchronization errors. After an investigation into the main thoughts around this topic, it was decided to pursue the number theoretic construction techniques as pioneered by Levenshtein as possible solutions to the problem of synchronization loss due to the spurious insertion and/or deletion of bits in the bitstream. The reason for this decision is based on the knowledge that most work on the correction of synchronization actually only corrects the synchronization and not the errors that occurred while synchronization was lost. Of the few coding techniques that do recover the lost information, the Levenshtein codes appear to offer the most elegant solution to the problem of bit insertions or deletions. Accordingly, it was decided to try and extend the insertion/deletion correcting capabilities of the Levenshtein codes to codes that are capable of correcting any number of random insertions and/or deletions.

First the Levenshtein codes were investigated to establish some known and several new properties. The most important of these new properties are the Hamming distance and weight relationships between the code words of a Levenshtein code. The next step in the research was to investigate the subcodes of the Levenshtein codes for possible applications where it would be preferable to combine the insertion/deletion correcting capability of the code with another constraint such as runlength or charge constraints. From this investigation it was
found that the runlength constrained subcodes have very low rates when compared to conventional runlength constrained codes. The most interesting result of this investigation, however, was that the first class of Levenshtein codes are particularly suited to spectral shaping techniques. Several classes of subcodes were investigated which have spectral zeros at 0 Hz or at the Nyquist frequency, higher order spectral zeros at these frequencies or a combination of several of these spectral properties. Amongst the results it was found that the first order dc free codes of Immink [34] are subcodes of the balanced Levenshtein codes and can therefore also correct a single insertion or deletion. Another result is a rate $R = 9/16$, $d_{min} = 4$, dc free code that has more properties than the known codes of the same rate. An in-depth study of the Levenshtein subcodes that have a first order spectral zero at 0 Hz and a spectral null at the Nyquist frequency was made. These codes have an abundance of properties that make them suitable for application on magnetic and optic recording channels. A decoding strategy for these codes is presented along with a construction technique that yields semi systematic codes, that is codes in which the code words contain most of the information bits at specific positions and the remaining information bits can be determined using simple boolean algebra. These codes approach the theoretic maximum rate of systematic balanced codes, namely $R \approx 1/2$ for large code word lengths.

In an effort to increase the knowledge regarding the insertion/deletion correcting capability of a code, number theoretic double adjacent insertion/deletion correcting code constructions were attempted. Three new constructions are presented which improve on or equal the two known codes in the literature. The best known code of this nature was also developed by Levenshtein, who courteously sent us a translated manuscript. The best new code differs significantly from the code presented by Levenshtein, but offers the same approximate rate using a much easier construction technique.

Using the knowledge gained by investigating the number theoretic single insertion/deletion correcting codes and the number theoretic double adjacent insertion/deletion correcting codes, an approach was developed for constructing multiple insertion and/or deletion correcting codes. This approach entails determining the relationships between the weight and Hamming distance of code words that are capable of correcting a certain number of deletions and then imposing a number of rules on a weight vector for use in determining the code word moment. Using this approach, number theoretic codes that are capable of correcting one or two random insertions/deletions were constructed. By generalizing the relationship between the weight and Hamming distance of code words several new bounds on the cardinality of multiple
insertion/deletion correcting codes were determined as well as bounds on the insertion/deletion correcting capability of a given code book. Using this Hamming distance/weight structure, we presented a general number theoretic multiple insertion and/or deletion correcting code construction. It is interesting to note that the single insertion/deletion correcting Levenshtein codes are contained in the new general class of number theoretic insertion/deletion correcting codes. The rates of these codes were investigated and presented. As can be expected, the correction of insertions/deletions requires a large amount of redundancy. This is the first time, however, that general insertion and/or deletion correcting codes are presented. The properties of the new class of codes also include significant Hamming distance properties, which means that the codes are also able to correct a certain number of additive errors. This is of importance as it is highly probable that the mechanism that causes the insertion/deletion of bits also could introduce additive errors.

7.2 Future research and open questions

When considering the work presented in this thesis, several questions crop up. One such question is if the rates of the new multiple insertion/deletion correcting codes can be improved. It must be noticed that the new construction technique partitions the $n$ space of binary sequences into code books which all must be able to correct the specified number of insertions/deletions. In practice only one such code book is actually needed. This constitutes one of the main drawbacks of the number theoretic construction technique presented here. Intuitively, it is felt that it should be possible to construct a single code book with the desired error correcting properties which has a larger rate than the equivalent number theoretic construction. Possible disadvantages of this scheme could be that the resulting code might not have the structure or ease of construction that the number theoretic construction has, and that such a construction might be difficult to generalize.

There is scope for improving the proofs of the new number theoretic constructions of which the proofs are involved and not very elegant. Such an investigation could lead to more insight in the structure of these codes which could be used to develop a simple decoding scheme. At present, decoding is done through the use of a lookup table which contains all possible subwords and the corresponding code words.

Another approach to the insertion/deletion correcting problem can be found through the investigation of the linear cyclic codes. During an investigation of the $(7, 4)$ Hamming code,
it was found that this code can be partitioned into two single insertion/deletion correcting codes. The following example demonstrates the partition.

Example
Consider the (7, 4) Hamming code in table 7.1. This code has Hamming distance of \( d_{\min} = 3 \) and thus the upper limit on the \( s \)-correcting capability of the code is \( s \leq 2 \). The Hamming codes form a subset of the cyclic codes.

TABLE 7.1
A (7, 4) HAMMING CODE

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>1111111</td>
</tr>
<tr>
<td>1101000</td>
<td>0010111</td>
</tr>
<tr>
<td>0110100</td>
<td>1001011</td>
</tr>
<tr>
<td>0011010</td>
<td>1100101</td>
</tr>
<tr>
<td>0001101</td>
<td>1110010</td>
</tr>
<tr>
<td>1000110</td>
<td>0111001</td>
</tr>
<tr>
<td>0100011</td>
<td>1011100</td>
</tr>
<tr>
<td>1010001</td>
<td>0101110</td>
</tr>
</tbody>
</table>

It is now possible to find a \( s = 1 \) correcting code by considering the different sets of words within the code that have the same weight. By taking every second cyclic shift of a word of a certain weight the code in table 7.2 is found.

TABLE 7.2
A (7, 3) \( s = 1 \) INSERTION/DELETION CORRECTING CODE

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>1111111</td>
</tr>
<tr>
<td>1101000</td>
<td>0010111</td>
</tr>
<tr>
<td>0011010</td>
<td>1100101</td>
</tr>
<tr>
<td>1000110</td>
<td>0111001</td>
</tr>
</tbody>
</table>

It should be noted that the Hamming distance properties of the original code are retained and that this subcode is also able to correct a single additive error.
In the above code, the choice of words is made in such a way that no word of a certain weight has a neighbor closer than \( s + 1 = 2 \) shifts in either direction. In a similar way it is now possible to construct a \( s = 2 \) correcting code by choosing words of a certain weight that have no neighbors closer than \( s + 1 = 3 \) shifts in either direction. The resulting \((7, 2)\) \( s = 2 \) code is shown in table 7.3.

**TABLE 7.3**

A \((7, 2)\) \( s = 2 \) INSERTION/DELETION CORRECTING CODE

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>1111111</td>
</tr>
<tr>
<td>1101000</td>
<td>0001101</td>
</tr>
</tbody>
</table>

Note that in this case the words of weight \( w = 4 \) are not included in the code book as it is possible that two deletions may result in the same subword.

It seem that it is possible that multiples of (cyclic) shifts of a code word described by a primitive polynomial result in a \( s \) - correcting code. In certain cases, as determined from inspection, these code words can be combined with the complementary code words to form a \( s \) - correcting code with a larger cardinality. It is suspected that this can be done if the complementary code words differ in weight from the code words described by the primitive polynomial by at least \( \Delta w = s \). It should also be noted that the \( s \) - correcting capability of a code is limited by the minimum Hamming distance of the code, as well as the minimum weight of the code. (In the case of cyclic codes these two parameters are the same.)

The codes described in the example are special cases of a polynomial approach to the description of insertions and deletions. It is clear from the results that a full investigation of these codes is warranted. It might be possible to generate the insertion/deletion correcting codes through a primitive polynomial and use conventional decoding techniques.

One of the most difficult problems to address is that of the calculation of the Levenshtein distance. This important parameter describes the insertion/deletion correcting capability of a code and if it can be calculated with ease, it would be possible to construct codes in much the same way that additive error correcting codes are presently constructed. This problem, however, seems to be the most difficult to solve.
7.3 Conclusion

A main result of this study has been the generalization of the known number theoretic insertion/deletion correcting code construction techniques. Additional results included simplified construction techniques for double adjacent insertions or deletions as well as several constrained codes that are able to correct a single insertion or deletion. A specific class of single insertion/deletion correcting codes that incorporate spectral shaping constraints was also presented along with an effective decoding strategy. In addition, many new properties of insertion/deletion correcting codes were found through this study, which were used to derive bounds on both the insertion/deletion correcting capability of a code and the cardinality of a code with a specific insertion/deletion correcting capability.

The investigation of possible methods to construct insertion/deletion correcting codes has lead to several promising fields of study, of which the polynomial approach appears the most important. The original aims of this study have been attained. A positive contribution to the field of synchronization error correcting coding has been made and new areas of interest have been exposed.
REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


APPENDIX A

SEMI SYSTEMATIC SPECTRAL SHAPING CODES

A.1 A class of semi systematic spectral shaping Levenshtein subcodes

In this appendix we present a construction method for a class of spectral shaping subcodes of the Levenshtein codes that have semi-systematic third stage decoders similar to the rate $R = 5/12$ code demonstrated in section 4.4.4. Following a similar approach, the properties of the 4-bit balanced segments of the spectral shaping Levenshtein subcodes of section 4.3.4 are investigated first. These properties are then used to concatenate 4-bit balanced segments in such a way that the code word moment equals $(n + 1)n/4$. To achieve this, the moments of all possible 4-bit balanced segments are determined. The total required code word moment is then expressed in terms of the moments of these six possible balanced 4-bit segments.

A code word from the class of spectral shaping codes in section 4.3.4, consists of $n/4$ segments of weight $w = 2$ and length $l = 4$. Because the positions of the segments influence their contribution to the total code word moment, it is necessary to determine what the total code word moment must be when the contribution due to the positions of the segments is disregarded. In this way it will be possible to concatenate the 4-bit segments without considering the position of the segment. It will then only be necessary to ensure that the total code word moment is correct.
**Property A.1**

The code word moment of a spectral shaping code word that only consists of 4-bit balanced segments, add up to $5n/4$.

**Proof**

The code word moment of a spectral shaping code word is:

$$\sum_{i=1}^{n} ix_i = (n+1)n/4.$$  

The contribution of every segment, except the first segment, to the code word moment is reduced. The total value by which the code word moment is reduced is:

$$\frac{n}{4} \times 2 \times 4 \times \sum_{i=1}^{n/4-1} i = (n/4 - 1)(n/4 - 1 + 1) 4$$

$$= n (n/4 - 1),$$

where the first term reflects the number of ones in a segment and the second and third terms reflect the contribution of a binary one in the $i$th 4-bit segment to the code word moment. The total code word moment then becomes:

$$\sum_{i=1}^{n/4} \sigma_i = (n + 1)n/4 - n(n/4 - 1)$$

$$= (n + 1)n/4 - (n - 4)n/4$$

$$= (n + 1 - n + 4)n/4$$

$$= 5n/4,$$  \hspace{1cm} (A.1)

where $\sigma_i$ denotes the moment of a 4-bit segment.

It should be noted that property A.1 does not take into account the effect of RAS on the weighted sum. A code must comply to the following criteria to have a RAS $= 0$:
Property A.2
In the case where a 4-bit segment does not have \( RAS = 0 \), the complement of that 4-bit segment must be included in the code word to ensure that the minimum bandwidth property is maintained.

**Proof**
The concatenation of a balanced word and the complement thereof always has \( RAS = 0 \). This is seen as follows:

Consider a code word \( y \) that has a \( RAS = c \). Thus:

\[
\sum_{i=1}^{n} (-1)^i y_i = c
\]

The complement of \( y \) is \(-y\). Thus:

\[
\sum_{i=1}^{n} (-1)^i (-y_i) = - \sum_{i=1}^{n} (-1)^i y_i = -c
\]

The RAS of the concatenated words then becomes:

\[
\sum_{i=1}^{n} (-1)^i y_i + \sum_{i=1}^{n} (-1)^i (-y_i) = c - c = 0.
\]

A method for mapping data bits onto the spectral shaping Levenshtein codes is now presented.
A.2 Mapping construction

Consider the six balanced 4-bit segments, $\sigma_i$, along with their respective code word moments:

<table>
<thead>
<tr>
<th>segment</th>
<th>moment</th>
<th>segment</th>
<th>moment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>50</td>
<td>1001</td>
<td>51</td>
</tr>
<tr>
<td>0011</td>
<td>7</td>
<td>1100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>6</td>
<td>1010</td>
<td>4</td>
</tr>
</tbody>
</table>

where the subscripts 0 and 1 are chosen arbitrarily to distinguish between those 4-bit segments that have the same moment. Of the 4-bit segments presented above, only 0110 and 1001 have $RAS = 0$.

A valid code word consists of these segments such that:

$$\sum_{i=1}^{n/4} \sigma_i = 5n/4.$$  

i) If $\sigma_i \neq 5$, then the complement of $\sigma_i$ is also included in the word to ensure that the minimum bandwidth property is maintained.

ii) The last segment, $\sigma_{n/4}$, is determined by the total code word moment of the $n/4-1$ previous segments, $\sum_{i=1}^{n/4-1} \sigma_i$.

The following mapping of data bits is done in such a way that the data bits appear at specific indices of the 4-bit segments.

<table>
<thead>
<tr>
<th>Index</th>
<th>Data</th>
<th>1, 4</th>
<th>2, 3</th>
<th>50</th>
<th>51</th>
<th>7, 6</th>
<th>7, 4</th>
<th>3, 4</th>
<th>6, 3</th>
<th>51</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7, 6</td>
<td>7, 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3, 4</td>
<td>6, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>51</td>
<td>50</td>
</tr>
</tbody>
</table>
The code words are then constructed by partitioning the $m$ data bits into 2-bit segments and mapping the first $\left[\frac{m-1}{2}\right]$ segments onto the appropriate 4-bit code segments in such a way that there is at most one 4-bit segment that does not have a complement in the code word. This is easily achieved by alternating the systematic positions of the data between indices (1, 4) and (2, 3). The final data segment is then mapped onto a 4-bit segment such that the code complies to properties (i) to (iii).

This construction results in a code with $n/2 - 2$ systematic positions. The code rate is given by the following proposition:

**Proposition A.1**

The mapping construction discussed above yields the class of spectral shaping subcodes of the Levenshtein codes (c.f. section 4.3.4) with rates $R = 1/2 - 1/n$.

**Proof**

The code construction yields code words of length

$$n = 4 \left(\left[\frac{m-1}{2}\right] + 1\right)$$

$$n/4 = \left[\frac{m-1}{2}\right] + 1$$

$$n/4 - 1 = \left[\frac{m-1}{2}\right]$$

$$n/2 - 2 = m - 1$$

$$n/2 = m + 1$$

$$1/2 = m/n + 1/n$$

$$\therefore \quad R = m/n = 1/2 - 1/n$$ (A.2)

The above result corresponds with statement 4.4 in which it was shown that the code rate of a systematic balanced code $R \leq 1/2$.

**A.3 Example**

From equation (A.2) we find that 7 data bits may be mapped onto 16 code bits to give a rate $R = 7/16$.

Consider the following data sequence: $01 \ 00 \ 10 \ 1$
The first three 2-bit data segments are now mapped onto 4-bit code segments such that the systematic positions alternate:

$$010010 \rightarrow \begin{array}{cccc} (1,4) & (2,3) & (1,4) & (2,3) \\ 7, 6 & 5_0 & 6, 3 & \\ \end{array}$$

The last data segment is then mapped onto a 4 bit segment such that $\frac{5n}{4} = 20$.

The possibilities in this case are either $5_0$ or $5_1$. $5_1$ is arbitrarily chosen and the final mapping is the as follows:

$$0100101 \rightarrow \begin{array}{cccc} (1,4) & (2,3) & (1,4) & (2,3) \\ 7 & 5_0 & 3 & 5_1 & \end{array}$$
APPENDIX B

PROOFS OF DOUBLE ADJACENT INSERTION/DELETION CORRECTING CODE CLASSES

B.1 Proof of theorem 5.2

In this section the proof of theorem 5.2 is presented. For clarity, the theorem is repeated here.

Theorem 5.2

The following equation describes a class of codes which are able to correct two adjacent deletions:

\[ \sum_{i=1}^{n} ix_i + \sum_{i=1}^{n-1} (n-i)x_i x_{i+1} = a \mod (3n), \]  

(B.1)

where \( x_i \in \{0, 1\} \) and \( n \) is the length of the code word \( x \), and \( a \) is an integer corresponding to the partition of the space of integers.

The proof of the error correcting capabilities of the code consists of showing that if a binary word \( x \) from a partition is subjected to any two adjacent deletions at positions \( i \) and \( i + 1 \) and then to any two insertions at positions \( j + 1 \) and \( j + 2 \), then the resulting word \( y \) is not in the same partition as \( x \) for all values of \( i \) and \( j \). Thus, the codes have a Levenshtein distance \( \varrho_a(x, y) > 2 \), where \( \varrho_a(x, y) \) denotes the Levenshtein distance for adjacent errors.
Proof
Consider the binary words $x$ and $y$ along with the indices corresponding to the first and second parts of the weighted sum respectively:

$$
\begin{array}{cccccccccccc}
n-1 & n-2 & n-i+1 & n-i & n-i-1 & n-i-2 & n-j+1 & n-j & n-j-1 & n-j-2 & n-j-3 & 0 \\
1 & 2 & i-1 & i & i+1 & i+2 & j-1 & j & j+1 & j+2 & j+3 & n \\
\end{array}
$$

$$
\begin{array}{cccccccccccc}
x_1 & x_2 & \cdots & x_{i-1} & x_i & x_{i+1} & x_{i+2} & \cdots & x_{j-1} & x_j & x_{j+1} & x_{j+2} & x_{j+3} & \cdots & x_n \\
x_1 & x_2 & \cdots & x_{i-1} & x_{i+2} & x_{i+3} & x_{i+4} & \cdots & x_{j+1} & x_{j+2} & y_a & y_b & x_{j+3} & \cdots & x_n \\
\end{array}
$$

Notice that in the second case, two adjacent bits, $x_i$ and $x_{i+1}$, were deleted and two other bits, $y_a$ and $y_b$, were inserted at positions $j+1$ and $j+2$. This ensures that there is at least one unchanged bit between the deletion and insertion of the adjacent bits.

Let 
$$
\sigma_x = \sum_{i=1}^{n} i x_i + \sum_{i=1}^{n-1} (n-i)x_i+1
$$

and 
$$
\sigma_y = \sum_{i=1}^{n} i y_i + \sum_{i=1}^{n-1} (n-i)y_{i+1}
$$

Then 
$$
\sigma_x - \sigma_y = \left[ \sum_{h=1}^{i-1} h(x_h - x_h) + \sum_{h=i}^{j} h(x_h - x_{h+2}) + \sum_{h=1}^{i-2} (j+1)(x_{j+1} - y_a) + \right.
$$

$$
\left. (j+2)(x_{j+2} - y_b) \right] + \sum_{h=1}^{j-1} \left( n-h \right)(x_{h+1} - x_{h+1}) + \sum_{h=i}^{j-1} \left( n-h \right)(x_{h+1} - x_{h+3}) + \sum_{h=i}^{n-1} (n-j)(x_{j+1} - y_b) + \sum_{h=1}^{n-1} \left( n-h \right)(x_{h+1} - x_{h+3}) + \sum_{h=1}^{n-1} \left( n-h \right)(x_{h+1} - x_{h+1})
$$
This can also be written as the decrease due to the deleted and shifted bits and the increase due to the inserted bits. Thus,

$$
\sigma_x - \sigma_y = [ix_i + (i + 1)x_{i+1} + (n - i + 1)x_{i-1}x_i + (n - i)x_{i+1} + (n - i - 1)x_{i+1}x_{i+2} + (n - j - 1)x_{j+1}x_{j+2} + (n - j - 2)x_{j+2}x_{j+3}] + \\
2\left[ \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right] - \left[ (n - j)x_{j+2} y_a + (n - j - 1)y_a y_b + (n - j - 2)y_b x_{j+3} + (n - i + 1)x_{i-1}x_{i+2} + (n - j + 1) y_a + (n - j + 2)y_b \right]
$$

It must now be shown that $\sigma_x - \sigma_y \neq 0 \mod (3n)$. An upper and lower bound on the value of $\sigma_x - \sigma_y$ is determined first.

**Upper Bound:**

Let $y_a = y_b = x_{i+2} = 0; x_i = x_{i-1} = x_{i+2} = x_{j+3} = 1$.

Then from (B.3):

$$
\sigma_x - \sigma_y \leq [i + i + 1 + (n - i + 1) + (n - i) + (n - j - 2)] + 2\left[ \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right] \\
= [3n - j] + 2\left[ \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right]
$$
The second term is maximized if there are no adjacent binary ones between positions \( i + 2 \) and \( j + 2 \). Because \( x_{j+2} = 1 \) and \( x_{i+1} = 1 \), this term will have a maximum value if binary "zeros" and "ones" alternate for every bit after position \( i + 1 \), starting with a binary "zero" and ending with a binary "zero" at position \( j + 1 \). The maximum number of binary "ones" that can occur is \( \left\lceil \frac{(j + 2) - (i + 2)}{2} \right\rceil \), where \( \lceil x \rceil \) is the ceiling function.

It now follows that:

\[
2 \left( \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right) \leq \left[ \frac{(j + 2) - (i + 2)}{2} \right] \leq j - i + 1.
\]

Furthermore, \( i + 2 \leq j \leq n - 3 \) and \( i \geq 2 \). Thus,

\[
\sigma_x - \sigma_y \leq [3n - j + j - i + 1] = 3n - 2 + 1 = 3n - 1
\]

Lower bound:
Let \( y_a = y_b = x_{i+2} = x_{i-1} = x_{j+2} = 1; x_i = x_{i+1} = x_{j+3} = 0 \).

Then from (B.3):

\[
\sigma_x - \sigma_y \geq - [n - i + 1 + j + 1 + j + 2 + n - j + n - j - 1] + 2 \left( \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right)
\]

\[= - [3n - i + 3] + 2 \left( \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right)\]

For the lower bound, the lowest possible value of the second term must be found under the present conditions. This value is found by noting that \( x_{j+2} = x_{i+2} = 1 \), and if all the other bits between these two respective positions are zero, the resulting summation equals 2 and the second term equals \(-1\). It is however possible that these two positions are adjacent, which
causes the summation of adjacent bits to equal 1. The result is that the minimum value of the second term is 2. Furthermore, \( i \geq 2 \) and

\[
\sigma_x - \sigma_y \geq -[3n - i + 3] + 2 \\
\geq -[3n - 2 + 3] + 2 \\
= [3n + 1 - 2] \\
= [3n - 1]
\]

It has now been proven that \([3n - 1] \geq \sigma_x - \sigma_y \geq -[3n - 1]\) for the case of the insertion taking place after the deletion. The case of the insertion of two adjacent bits before the deletion is proven as above and the result is \([3n - 1] \geq \sigma_x - \sigma_y \geq -[3n - 1]\).

The only case of interest is when \( \sigma_x - \sigma_y = 0 \). This occurs when \( x = y \). Thus any of the partitions described in (B.3) can correct two deletions, and because \( \rho_{ns}(x, y) > 2 \), two adjacent insertions can also be corrected.

The case of two adjacent deletions and then two adjacent insertions in the same position can be considered as equivalent to at most two adjacent additive errors.

**Theorem 5.3**

Every partition described by (B.3) can detect two additive errors.

By considering all possible cases of two adjacent additive errors which changes a word \( x \) into a word \( y \), it is proven in appendix A that the resulting word \( y \) is not in the same partition as the original word \( x \). It is however possible that the same resulting word is obtained from two different original words. This implies that the error can be detected but not corrected. It should be noted that the case of a single insertion and a single deletion in adjacent positions can result at the worst in a single additive error. Because the code described in (B.3) has a minimum distance \( d_{\text{min}} = 2 \), this error can be detected but not corrected. Thus, the codes described by (B.3) can only correct two adjacent insertions or two adjacent deletions, but not a single insertion and a single deletion at adjacent positions.
By careful observation of the values which \( \sigma_x - \sigma_y \) can assume in the proof given in appendix A, it is found that the following equation also describes partitions of the \( 2^n \) words that are able to correct either two adjacent deletions or two adjacent insertions, but not a combination of adjacent insertions and deletions:

**B.2 Proof of theorem 5.3**

In this section, the proof of theorem 5.3 is given. The theorem is restated below:

**Theorem 5.3**

The code words in the partitions described by (B.4) and (B.5) are able to detect two adjacent additive errors in a code word.

\[
\sum_{i=1}^{n} i x_i + \sum_{i=1}^{n-1} (n - i) x_i x_{i+1} \equiv a \mod (3n),
\]

where \( x_i \in \{0, 1\} \) and \( n \) is the length of the code word \( x \), and \( a \) is an integer corresponding to the code partition.

\[
\sum_{i=1}^{n} i x_i + \sum_{i=1}^{n-1} (n - i) x_i x_{i+1} \equiv a \mod (2n + 1),
\]

where the variables are the same as for (A.1).

**Proof**

Let the double adjacent additive error take place at positions \( i \) and \( i + 1 \). Consider the bits surrounding the double adjacent additive error.

\[
\begin{array}{cccccccc}
  n-1 & n-2 & n-i+1 & n-i & n-i-1 & n-i-2 & 0 \\
  1 & 2 & i-1 & i & i+1 & i+2 & n \\
  x_1 & x_2 & \ldots & x_{i-1} & x_i & x_{i+1} & x_{i+2} & \ldots & x_n \\
  x_1' & x_2' & \ldots & x_{i-1}' & x_i' & x_{i+1}' & x_{i+2}' & \ldots & x_n \\
\end{array}
\]
All possible values of the difference between the weighted sum of the original code word and the resulting word are investigated. These values are listed below:

<table>
<thead>
<tr>
<th>Position</th>
<th>$\Delta = \sigma_x - \sigma_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>$2n - 1 \geq \Delta \geq n + 3$</td>
</tr>
<tr>
<td>0110</td>
<td>$n + i + 1$</td>
</tr>
<tr>
<td>0011</td>
<td>$2n + 2$</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>$3n - i + 1$</td>
</tr>
<tr>
<td>0011</td>
<td>$3n - 1 \geq \Delta \geq 2n + 3$</td>
</tr>
<tr>
<td>0101</td>
<td>$2n$</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>$n - i$</td>
</tr>
</tbody>
</table>

From the above it is noted that $\Delta = \sigma_x - \sigma_y \neq 0 \ mod(3n)$ or $\ mod(2n + 1)$. Therefore the resulting word after two adjacent additive errors is not in the same partition as the original word, and the error can thus be detected. This does not imply that the error can be corrected.
In this section the proof of theorem 5.4 is given. The theorem is restated below:

**Theorem 5.4**

The code words in the partitions described by (B.6) are able to correct two adjacent deletions in a code word.

\[
\sum_{i=1}^{n} ix_i + \sum_{i=1}^{n-1} (n-i)x_i x_{i+1} \equiv a \mod (2n+1),
\]  

(B.6)

where \(x_i \in \{0, 1\}\) and \(n\) is the length of the code word \(x\), and \(a\) is an integer corresponding to the code partition.

**Proof**

Let the double adjacent deletions take place at positions \(i\) and \(i+1\), and let two adjacent insertions take place at positions \(j+1\) and \(j+2\) respectively. Let \(y\) be the resulting word. Then the differences in the weighted sum for all possible combinations of insertions and deletions can be calculated. Consider the two words \(x\) and \(y\):

\[
\begin{array}{cccccccccccc}
\text{n-1} & \text{n-2} & \text{n-i+1} & \text{n-i} & \text{n-i-1} & \text{n-i-2} & \text{n-j+1} & \text{n-j} & \text{n-j-1} & \text{n-j-2} & \text{n-j-3} & 0 \\
1 & 2 & i-1 & i & i+1 & i+2 & j-1 & j & j+1 & j+2 & j+3 & n \\
x_1 & x_2 & \ldots & x_{i-1} & x_i & x_{i+1} & x_{i+2} & \ldots & x_{j-1} & x_j & x_{j+1} & x_{j+2} & x_{j+3} & \ldots & x_n \\
x_1 & x_2 & \ldots & x_{i-1} & x_{i+2} & x_{i+3} & x_{i+4} & \ldots & x_{j+1} & x_{j+2} & y_a & y_b & x_{j+3} & \ldots & x_n \\
\end{array}
\]

Notice that in this case two adjacent bits, \(x_i\) and \(x_{i+1}\), were deleted and two other bits, \(y_a\) and \(y_b\), were inserted at positions \(j+1\) and \(j+2\). This ensures that there is at least one unchanged bit between the deletion and insertion of the adjacent bits.

Let \(\sigma_x = \sum_{i=1}^{n} ix_i + \sum_{i=1}^{n-1} (n-i)x_i x_{i+1}\)
and \[ \sigma_y = \sum_{i=1}^{n} iy_i + \sum_{i=1}^{n-1} (n-i)y_{i+1}. \]

Then \[ \sigma_x - \sigma_y = \left[ \sum_{h=1}^{i-1} h(x_h - x_h) + \sum_{i=1}^{j} h(x_h - x_{h+2}) + (j+1)(x_{j+1} - y_a) + (j+2)(x_{j+2} - y_b) \right] + \left[ \sum_{h=1}^{j-2} \sum_{h=h+1}^{n-i} (n-h)(x_h - x_{h+1}) + \sum_{h=h+1}^{n-i} (n-h)(x_h - x_{h+1}) - x_{h+2}x_{h+3} \right] + \left[ \sum_{h=h+1}^{j-1} (n-h)(x_h - x_{h+1}) - x_{h+2}x_{h+3} \right] \]

\[ \left[ \sum_{h=h+1}^{j-1} (n-h)(x_h - x_{h+1}) - x_{h+2}x_{h+3} \right] \]

This can also be written as the decrease due to the deleted and shifted bits and the increase due to the inserted bits. Thus,

Decrease due to deleted bits:

\[ \Delta_d = \sigma_x - \sigma_y = \left[ \hat{x}_i + (i+1)x_{i+1} \right] + \left[ (n-i+1)x_{i+1}x_i - (n-i+1)x_{i+1}x_{i+2} \right] + \left[ (n-i)x_{i+1}x_i + (n-i-1)x_{i+1}x_{i+2} \right] \]
By considering all possible values of the bits $x_{i-1}, x_i, x_{i+1}, x_{i+2}$, the following values for $\Delta_d$ are calculated:

<table>
<thead>
<tr>
<th>$x_{i-1}, x_i, x_{i+1}, x_{i+2}$</th>
<th>$\Delta_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>$i + 1$</td>
</tr>
<tr>
<td>0011</td>
<td>$n$</td>
</tr>
<tr>
<td>0100</td>
<td>$i$</td>
</tr>
<tr>
<td>0101</td>
<td>$i$</td>
</tr>
<tr>
<td>0110</td>
<td>$n + i + 1$</td>
</tr>
<tr>
<td>0111</td>
<td>$2n$</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>$-(n - i + 1)$</td>
</tr>
</tbody>
</table>

Increase due to inserted bits:

$$\Delta_i = \sigma_x - \sigma_y = [(j + 1)y_a + (j + 2)y_b] + [(n - j)x_{j+2}y_a + (n - j - 1)y_a y_b + (n - j - 2)y_b x_{j+3} - (n - j - 2)x_{j+2}x_{j+3}]$$  \hspace{1cm} (B.9)
By considering all possible values of the bits $x_{j+2}y_a y_b x_{j+3}$, the following values for $\Delta$ are calculated:

<table>
<thead>
<tr>
<th>$x_{j+2}y_a y_b x_{j+3}$</th>
<th>$\Delta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>$j + 2$</td>
</tr>
<tr>
<td>0011</td>
<td>$n$</td>
</tr>
<tr>
<td>0100</td>
<td>$j + 1$</td>
</tr>
<tr>
<td>0101</td>
<td>$j + 1$</td>
</tr>
<tr>
<td>0110</td>
<td>$n + j + 2$</td>
</tr>
<tr>
<td>0111</td>
<td>$2n$</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>$(n - j - 2)$</td>
</tr>
<tr>
<td>1010</td>
<td>$j + 2$</td>
</tr>
<tr>
<td>1011</td>
<td>$j + 2$</td>
</tr>
<tr>
<td>1100</td>
<td>$n + 1$</td>
</tr>
<tr>
<td>1101</td>
<td>$j + 3$</td>
</tr>
<tr>
<td>1110</td>
<td>$2n + 2$</td>
</tr>
<tr>
<td>1111</td>
<td>$2n + 2$</td>
</tr>
</tbody>
</table>

Decrease due to shifted bits:

$$\Delta_s = \sigma_x - \sigma_y = 2 \left[ \sum_{h=i+2}^{j+2} x_h - \sum_{h=i+2}^{j+1} x_h x_{h+1} \right]$$ \hspace{1cm} (B.10)

This term is maximized if there are no adjacent binary ones between positions $i+2$ and $j+2$ and a maximum value is achieved if binary "zeros" and "ones" alternate for every bit after position $i+1$, starting with a binary "zero" and ending with a binary "zero" at position $j+1$. The maximum number of binary "ones" that can occur is

$$\Delta_s = \sigma_x - \sigma_y \leq \left[ \frac{(j+2) - (i+2)}{2} \right]$$ \hspace{1cm} (B.11)

where $[x]$ is the ceiling function. The following maximum values of (B.11) are found:
\[ \Delta_s = \sigma_x - \sigma_y \leq j - i + 1, \quad \text{if } j - 1 \text{ is even; and} \]
\[ \Delta_s = \sigma_x - \sigma_y \leq j - i, \quad \text{if } j - 1 \text{ is odd.} \]

The minimum value of \( \Delta_s = \sigma_x - \sigma_y \) is zero. This occurs when all the bits between \((j + 2)\) and \((i + 2)\) are binary zeros. It should be noted that the boundary conditions for the differences due to the deleted and inserted bits, play an important role in the determination of the minimum value of the difference due to the shifted bits. The differences possible due to the deleted and inserted bits are summarized as follows:

\begin{align*}
\Delta_d & \quad \Delta_i \\
0 & \quad 0 \\
i - 1 & \quad j + 1 \\
i & \quad j + 2 \\
i + 1 & \quad j + 3 \\
n & \quad n \\
n + 1 & \quad n + 1 \\
n + i + 1 & \quad n + j + 2 \\
2n & \quad 2n \\
- (n - i + 1) & \quad - (n - j - 2)
\end{align*}

Furthermore, \( i + 2 \leq j \leq n - 3 \) and \( i \geq 2 \). It is now possible to evaluate all the different values that \( \sigma_x - \sigma_y \) can assume.

Consider the different maxima:

Let \( \Delta_d = 2n + 2 \) and \( \Delta_i = - (n - j - 2) \). Thus,

\[
\begin{align*}
\sigma_x - \sigma_y &= 2n + 2 + n - j - 2 + \Delta_s \\
&= 3n - j + \Delta_s \\
&\leq 3n - j + j - i + 1 \\
&= 3n - 2 + 1 \\
&= 3n - 1
\end{align*}
\]
and \[ \sigma_x - \sigma_y \geq 3n - j + \Delta_s \]
\[ = 3n - (n - 3) + 2 \]
\[ = 2n + 5 \]
(note the boundary conditions)

Thus \( \sigma_x - \sigma_y \neq 0 \mod (2n + 1) \) \( \forall i + 2 \leq j \leq n - 3 \) and \( n - 5 \geq i \geq 2, n \geq 8 \).

Let \( \Delta_d = 2n + 2 \) and \( \Delta_i = 0 \). Thus,

\[ \sigma_x - \sigma_y = 2n + 2 + \Delta_s \]
\[ \geq 2n + 2 \]
\[ < 3n - 1 \]

Thus \( \sigma_x - \sigma_y \neq 0 \mod (2n + 1) \) \( \forall i + 2 \leq j \leq n - 3 \) and \( n - 5 \geq i \geq 2, n \geq 8 \).

Let \( \Delta_d = 2n + 2 \) and \( \Delta_i = j + 1 \). Thus,

\[ \sigma_x - \sigma_y = 2n + 2 - j - 1 + \Delta_s \]
\[ = 2n + 1 - j + \Delta_s \]
\[ \leq 2n + 1 - j + j - i + 1 \]
\[ \sigma_x - \sigma_y \leq 2n + 2 - i \]
\[ \leq 2n \]

and

\[ \sigma_x - \sigma_y \geq 2n + 1 - j + \Delta_s \]
\[ = 2n + 1 - n + 3 + 0 \]
\[ = n + 4 \]

(note the boundary conditions)

Thus \( \sigma_x - \sigma_y \neq 0 \mod (2n + 1) \) \( \forall i + 2 \leq j \leq n - 3 \) and \( n - 5 \geq i \geq 2, n \geq 8 \).

Because the other values of \( \Delta_i > 0 \), the difference \( \sigma_x - \sigma_y < 2n + 1 \neq 0 \mod (2n + 1) \) for \( \Delta_d = 2n + 2 \).
Let $\Delta_d = 2n$ and $\Delta_i = -(n - j - 2)$. Thus,

$$\sigma_x - \sigma_y = 2n + n - j - 2 + \Delta_s$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for $i + 2 \leq j \leq n - 3$ and $n - 5 \leq i \leq n - 8$.

Let $\Delta_d = 2n$ and $\Delta_i = 0$. Thus,

$$\sigma_x - \sigma_y = 2n + \Delta_s$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for $i + 2 \leq j \leq n - 3$ and $n - 5 \leq i \leq n - 8$.

Let $\Delta_d = 2n$ and $\Delta_i = j + 1$. Thus,

$$\sigma_x - \sigma_y = 2n - j + \Delta_s$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for $i + 2 \leq j \leq n - 3$ and $n - 5 \leq i \leq n - 8$.

Because the other values of $\Delta_i > 0$, the difference $\sigma_x - \sigma_y < 2n + 1 \neq 0 \mod (2n + 1)$ for $\Delta_d = 2n$. 


Let $\Delta_d = n + i + 1$ and $\Delta_i = -(n - j - 2)$. Thus,

$$\sigma_x - \sigma_y = n + i + 1 + n - j - 2 + \Delta_s$$

$$= 2n - 1 + i + j + \Delta_s$$

$$\leq 2n + i + j - 1 + j - i + 1$$

$$= 2n$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for all values of $i$ for all values of $\Delta_i$, the resulting difference $\sigma_x - \sigma_y$ is.

Because the other values of $\Delta_d < n + i + 1$ for all values of $\Delta_i$, the resulting difference $\sigma_x - \sigma_y$ is.

Consider the minimum values that $\sigma_x - \sigma_y$ can assume:

Let $\Delta_d = -(n - i + 1)$ and $\Delta_i = 2n + 2$. Thus,

$$\sigma_x - \sigma_y = -(n - i + 1) - (2n + 2) + \Delta_s$$

$$= -n + i - 1 - 2n - 2 + \Delta_s$$

$$= -3n - 3 + i + \Delta_s$$

$$\leq -3n + i - 3 + j - i + 1$$

$$= -3n - 2 + n - 3$$

$$= -(2n + 5)$$

$$\sigma_x - \sigma_y = -(n - i + 1) - (2n + 2) + \Delta_s$$

$$= -n + i - 1 - 2n - 2 + \Delta_s$$

$$\geq -3n - 3 + 2 + 2$$

$$= -(3n - 1)$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for all values of $i$ for all values of $\Delta_i$, the resulting difference $\sigma_x - \sigma_y$ is.
Let $\Delta_d = -(n - i + 1)$ and $\Delta_i = 2n$. Thus,
\[
\sigma_x - \sigma_y = -(n - i + 1) - (2n) + \Delta_s
\]
\[
= -n + i - 1 - 2n + \Delta_s
\]
\[
= -3n - 1 + i + \Delta_s
\]
\[
\leq -3n + i - 1 + j - i + 1
\]
\[
= -3n + n - 3
\]
\[
= -(2n + 3)
\]
\[
\sigma_x - \sigma_y = -(n - i + 1) - (2n) + \Delta_s
\]
\[
= -n + i - 1 - 2n + \Delta_s
\]
\[
= -3n - 1 + i + \Delta_s
\]
\[
\geq -3n - 3 + 2 + 2
\]
\[
= -(3n - 1)
\]
Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1) \forall i + 2 \leq j \leq n - 3$ and $n - 5 \geq i \geq 2, n \geq 8$.

Let $\Delta_d = -(n - i + 1)$ and $\Delta_i = n + j + 2$. Thus,
\[
\sigma_x - \sigma_y = -(n - i + 1) - (n + j + 2) + \Delta_s
\]
\[
= -2n + i - j - 3 + \Delta_s
\]
\[
\leq -2n + i - j - 3 + j - i + 1
\]
\[
= -2n - 2
\]
\[
= -(2n + 2)
\]
\[
\sigma_x - \sigma_y = -2n + i - j - 3 + \Delta_s
\]
\[
\geq -2n - 1 - j + i
\]
\[
= -2n - 1 - (n - 3) + 2
\]
\[
= -(3n - 4)
\]
Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1) \forall i + 2 \leq j \leq n - 3$ and $n - 5 \geq i \geq 2, n \geq 8$. 
Let $\Delta_d = -(n - i + 1)$ and $\Delta_i = n + 1$. Thus,

$$
\sigma_x - \sigma_y = -(n - i + 1) - (n + 1) + \Delta_s
= -2n + i + \Delta_s
\leq -2n + i + j - i + 1
= -2n + 1 + j
= -2n + 1 + n - 3
= -(n - 2)
$$

$$
\sigma_x - \sigma_y = -2n + i + \Delta_s
\geq -2n - 2 + 2
= -2n
$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for all values of $\Delta_d$.

Because the other values of $\Delta_i < n + 1$, the difference $\sigma_x - \sigma_y < 2n + 1 \neq 0 \mod(2n + 1)$ for $\Delta_d$.

Let $\Delta_d = 0$ and $\Delta_i = 2n + 2$. Thus,

$$
\sigma_x - \sigma_y = 0 - 2n - 2 + \Delta_s
= -2n - 2 + \Delta_s
\leq -2n - 1 + j - i
= -2n - 1 + n - 5
= -n - 4
= -(n + 4)
$$

$$
\sigma_x - \sigma_y = -2n - 2 + \Delta_s
\geq -2n - 2 + 2
= -2n
$$

Thus $\sigma_x - \sigma_y \neq 0 \mod (2n + 1)$ for all values of $\Delta_i$.

Because the other values of $\Delta_d > 0$ for all values of $\Delta_i$, the resulting difference $\sigma_x - \sigma_y < 2n \neq 0 \mod (2n + 1)$ for $\Delta_d$.
From the above it is seen that the only possible case for $\sigma_x - \sigma_y = 0$ is when $x = y$.

It has now been proven that the difference in the weighted sum of a sequence $x$ and the same sequence after two adjacent deletions and two adjacent insertions, are not the same mod($2n + 1$). These sequences are then able to correct two adjacent deletions and insertions. Note that the case of the deletions and insertions taking place in the same positions causes at most two adjacent additive errors. The codes are not able to correct such errors, but are able to detect them. This has been proven in Appendix B, section B.2.
APPENDIX C

TEST PROGRAM FOR $a = 2$ INSERTION/DELETION CORRECTING CODES

In this appendix, the program for testing a code book for the ability to correct two adjacent deletions, is given. The program, which is written in Borland's Turbo Pascal version 6.0, consists of several subroutines and a main program section.

One of the key points in the approach to the problem of a general test for deletion correcting ability, is the fact that the number of code words and the number of subwords vary significantly. Due to this variation, it is not possible to use a single data structure of a fixed size that could be able to contain all the possible code words and subwords of different lengths. The solution to this problem is to use a variable size data structure that "grows" and "shrinks" according to the specific code book being tested. This variable size data structure was realized through the use of pointers which form a variable sized two-dimensional array containing the code words and the corresponding subwords. Efficient memory management was applied through the re-use of old data locations. The program also makes use of command line parameters. This enables the program to be used in a batch file without supervision, which saves a lot of time.

The program is used as follows: 

```
NS_TOETS /I[infile.txt] /T[outfile.txt],
```

where the infile contains the code book to be tested and outfile.txt will be used to save the result of the test. Note that the outfile.txt is appended and not rewritten, and that it must exist prior to the test.
{R+}

Program s_toets; {Hierdie program lees 'n kode boek vanaf le're en bepaal die maksimum s-korreksie vermoe daarvan.}

Uses CRT, COMMS, iomanage;

type shrt = string[16];
    datapointer = ^datarekord;
    listpointer = ^listrekord;

    datarekord = RECORD
        data : longint;
        nextdata : datapointer
    end;

    listrekord = RECORD
        xdata : datapointer;
        nextlist : listpointer;
    end;

    Var
    code : shrt;
    i,j,lbnd,wordlength,s : Integer;
    pnt,xlist,xobject,xavail,oldxlist : datapointer;
    ylist,yobject,yavail,oldylist : listpointer;
    vlag,endvlag : boolean;
    b : char;
    filevar : text;
    ofile : string;
procedure Initialise;

begin

xlist:=nil;
ylist:=nil;
oldxlist:=nil;
oldylist:=nil;
xavail:=nil;
yavail:=nil;
vlag:=false;
endvlag:=false;
crSCR;
s:=0;
ofile:=paramget('/t');
end;

Procedure xpush(VAR xobject, list :datapointer);

begin

xobject^.nextdata:=list;
list:=xobject;
end;

Procedure xpop(var xobject,list :Datapointer);

begin
If list=nil then New(xobject) else
begin
xobject:=list;
list:=list^.nextdata;
end;
end;
TEST PROGRAM FOR $a = 2$ INSERTION/DELETION CORRECTING CODES

Procedure ypush(VAR yobject, list :listpointer);

begin
  yobject^.nextlist:=list;
  list:=yobject;
end;

Procedure ypop(var yobject, list :listpointer);

begin
  If list=nil then New(yobject) else
  begin
    yobject:=list;
    list:=list^.nextlist;
  end;
end;

Procedure addxobject(data:longint);

begin
  xpop(xobject, xavail);
  xobject^.data:=data;
  xpush(xobject, xlist);
end;

Procedure addyobject(var xlist:datapointer);

begin
  ypop(yobject, yavail);
  yobject^.xdata:=xlist;
  ypush(yobject, ylist);
  xlist:=nil;
end;
Procedure deletexobject(var list:datapointer);

begin
xpop(xobject.list);
xpush(xobject,xavail);
end;

Procedure showxlist(p:datapointer);

var wrd :shrt;

begin
while p <> nil do
begin
wrd := des2bin(p^.data.wordlength);
write(wrd,' ');
{write(filevar,wrd,' ');
 p:=p^.nextdata;
 end;
{writeln(filevar);
 writeln;
end;

Procedure deleteryobject(var list:listpointer);

begin
ypop(yobject.list);
ypush(yobject,yavail);
end;
Procedure showxylst(pp:listpointer);

var wrd :shrt;

begin
{assign(filevar,ofile);
append(filevar);}
while pp <> nil do
begin
showxlist(pp^.xdata);
pp:=pp^.nextlist;
end;
{close(filevar);} 
end;

procedure s_bound;

var i,j,dtmp,dh :Integer;
tmp1,tmp2 :shrt;
a :char;
data :Longint;

begin
setinfilenlength;
tmp1:=sread;
wordlength:=codelength(tmp1);
dtmp:=wordlength + 1;
for i:=1 to (infilenlength) do
begin
resetinfile;
setinfilen(i);
tmp1:=sread;
data:=Bin2des(tmp1);
addxobject(data);
addyobject(xlist);
for j:=(i+1) to (infilelength) do
begin
setinfile(j);
tmp2:=sread;
dh:=dmin(tmp1,tmp2);
if dh < dtmp then dtmp:=dh;
end;
end;
writeln('Die kode in ',infilename,' het dmin = ',dtmp);
{repeat until keypressed;
a:=readkey;}
subword:=tmpword;
delete(subword,1,2);
subdata:=bin2des(subword);
pnt:=xlist;
flag:=0;
while pnt<>nil do
  begin
    if subdata=pnt^.data then flag:=1;
    pnt:=pnt^.nextdata;
  end;
if flag=0 then addxobject(subdata);
ed;
deletexobject(oldxlist);
ed;
addyobject(xlist);
deleteyobject(oldylist);
ed;
wordlength:=wordlength-2;
showxylisr(ylist);
{repeat until keypressed;
a:=readkey;}
ed;

Procedure comparesubwords;
var
  ypnt1,ypnt2 :listpointer;
  xpnt1,xpnt2 :datapointer;
Begin
  vflag:=false;
  ypnt1:=ylist;
  while ypnt1<>nil do
    begin
      xpnt1:=ypnt1^.xdata;
      

while xpnt1<>nil do
begin
  ypnt2:=ypnt1^.nextlist;
while ypnt2<>nil do
  begin
    xpnt2:=ypnt2^.xdata;
    while xpnt2<>nil do
      begin
        if xpntl^.data=xpnt2^.data then
          begin
            vlag:=true;
            xpntl:=nil;
            xpnt2:=nil;
            ypntl:=nil;
            ypnt2:=nil;
            end
        ELSE
          xpnt2:=xpnt2^.nextdata;
        end;
      if ypnt2<>nil then ypnt2:=ypnt2^.nextlist;
    end;
  end;
while xpnt1<>nil do
begin
  if infilename=" then endvlag:=true;
  if infilelength<=1 then endvlag:=true;
  while endvlag=false do
    begin

{writeln('There are ',memavail,' bytes available.');}
repeat until keypressed;
b:=readkey;
s_bound;
writeln;
{ showxylist(ylist);
writeln('There are ',memavail,' bytes available.');}
getsubwords;
comparesubwords;
if vlag=false then begin
writeln('Die kode het ns = 2');
{ assign(filevar,ofile);
append(filevar);
writeln(filevar,'Die kode in ',infilename,' het ns = 2');}
end
else
begin
{ writeln(filevar,'Die kode is NIE ns=2 NIE!!');}
writeln('DIE KODE IS NIE NS=2 NIE');
end;
endvlag:=true;
end;
{close(filevar);} 
closeup;
end.
APPENDIX D

TEST PROGRAM FOR GENERAL $s$ - INSERTION/DELETION CORRECTING CODES

In this appendix, the program for determining the maximum $s$ - insertion/deletion correcting capability of a code book, is given. The program, which is written in Borland's Turbo Pascal version 6.0, consists of several subroutines and a main program section.

As in appendix C, a variable size data structure that "grows" and "shrinks" according to the specific code book being tested, is used. This variable size data structure was realized through the use of pointers which form a variable sized two - dimensional array containing the code words and the corresponding subwords. Efficient memory management was applied through the re - use of old data locations. The program also makes use of command line parameters. This enables the program to be used in a batch file without supervision, which saves a lot of time.

The program is used as follows: $S_{TOETS} /I[\text{in file}.txt] /T[\text{out file}.txt]$, where the infile contains the code book to be tested and outfile.txt will be used to save the result of the test. Note that the outfile.txt is appended and not rewritten, and that it must exist prior to the test.

Because the program is basically the same as the program in Appendix C, only the subroutine/procedure for the generation of the subwords is given here. This subroutine replaces the same subroutine in the program in Appendix C.
Procedure getsubwords; {Kry die subwords van elke woord/subwoord na 1 deletion}

Var i,j,flag :Integer;
subword,tmpword :string;
subdata,tmp :Longint;
a :char;

Begin
oldylist:=ylist;
ylist:=nil;
xlist:=nil;
while oldylist<>nil do
begin
oldxlist:=oldylist^.xdata;
while oldxlist<>nil do
begin
for i:=1 to wordlength do
begin
  tmp:=oldxlist^.data;
tmpword:=des2bin(tmp,wordlength);
subword:=delbit(tmpword,i);
subdata:=bin2des(subword);
pnt:=xlist;
flag:=0;
while pnt<>nil do
begin
  if subdata=pnt^.data then flag:=1;
pnt:=pnt^.nextdata;
end;
if flag=0 then addxobject(subdata);
end;
deletexobject(oldxlist);
end;
addyobject(xlist);
deleteyobject(oldylist);
end;
wordlength:=wordlength-1;
showxylisylist(ylist);
writeln('Press any key to continue ...');
repeat until keypressed;
a:=readkey;
end;