Error Detection and Correction in the International Standard Book Number

Peter Waweru Kamaku

A thesis submitted in fulfillment for the Degree of Doctor of Philosophy in Pure Mathematics in the Jomo Kenyatta University of Agriculture and Technology

2013

DECLARATION

This thesis is my original work and has not been presented for a degree in any University.

Signature: .... Date: ....

Peter Waweru Kamaku

This thesis has been submitted for examination with our approval as University supervisors:

Signature: .... Date: ....

Dr. Bernard Kivunge

KU, Kenya

Signature: .... Date: ....

Dr. Jotham Raymond Akanga

JKUAT, Kenya

DEDICATION

To my lovely queen Fridah and the fruit of our love, Daniella and others to come.

ACKNOWLEDGEMENT

I acknowledge the most high GOD for designing me to be who I am.
I acknowledge my loving parents, Washington Kamaku and Martha Wanjira; their unending love I can't live without.
I acknowledge the love of my life, my wife Fridah Mumo and our daughter Daniella Wanjira for choosing to spend their lives with me.
I also acknowledge my supervisors , Dr. Bernard Kivunge (Department of Mathematics, K.U.), Dr. Jotham Raymond Akanga (Department of Pure and Applied Mathematics, JKUAT) and the late Prof. C. Mwathi for their great guidance.
I thank my colleagues C. Wamuti, S. G. Maina , V. Mwai and P. Wangai for their assistance and moral support.
Special acknowledgement to my class 1-3 primary school teacher Miss Wacheng'e and all sunday school teachers at Muringa ACK church for their love and tireless efforts to build a responsible boy in me.
Special thanks to my siblings, friends, colleagues and those who participated in building the Pure Mathematician in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF SYMBOLS, NOTATIONS AND ABBREVIATIONS

$\boldsymbol{a \mid b}$	a divides b
$\boldsymbol{a \nmid b}$	a does not divide b
$\boldsymbol{iff}$	if and only if
$\sum$	Summation
$\boldsymbol{n!}$	Factorial
$\boldsymbol{nPr}$	n elements are to be chosen r at a time, that is, n choose r
$\boldsymbol{(F_q)^n}$	set of all ordered n-tuples, $a = a_1a_2 \ldots a_n$ where each $a_i \epsilon F_q$
$\boldsymbol{C}$	Code
$\boldsymbol{u}$ or $\boldsymbol{v}$	Codeword
$(\mathbf{k}, \mathbf{m})$	gcd of k and m
$\boldsymbol{Z_n}$	Set of integers modulo n
$(\mathbb{Z}_\mathbf{n}, \times, +)$	A field of integers modulo n under operations $\times$ and $+$

ABSTRACT

The International Standard Book Number (ISBN) is a code that uniquely identifies all books published internationally. Currently ISBN-13 has been in use since the year 2007 as an improvement to ISBN-10. In this research the error detection and correction capabilities of both ISBN-10 and ISBN-13 codes against the total number of codewords that can be generated is analysed. It is shown that both codes have major weaknesses as far as these properties are concerned. The ISBN -16 code then is designed and analysed as an improvement to the existing code. Comparison to the existing code is done and a conversion tool for the existing ISBN -13 to ISBN -16 developed. It is shown that the ISBN -16 code surpases the existing code in error detection and correction capabilities and the overall dictionary.

CHAPTER ONE

INTRODUCTION

1.1 Background Information

A positive integer greater than one is said to be prime if it has no positive integer divisors other than 1 and itself; otherwise it is referred to as a composite, that is if m is a composite, then integer factors n and k can be found such that $m = nk, n > 1, k > 1$. A set is a collection of related objects. For example the collection {orange, pineapple, mango} is a set of fruits whereby orange, pineapple and mango are the elements. The set A is said to be closed under a binary operation $*$ if whenever $a, b \in A$ then $a * b \in A$.

Communication involves the sending and receiving of information from the sender to the receiver. The information may be converted into a code which is familiar to both parties. The process of converting the information to the code is referred to as encoding whereas decoding is the process of converting the received code back into information which the receiver understands.

The elements in the code are referred to as the code words whereas the length of a code is the number of digits in each codeword.

For example, if $u = 10101$,where u is a code word in C, then u is of length 5.

Let u and v be code words. The hamming distance of u and v is the number of places in which u and v differ or not agree, Irving and Chen (1999). It is denoted by $d(u, v)$. In other words, it is a count of the minimum number of bits that must be changed to convert one code word to another, Houghton *et al.* (2001).

For example, if $u = 10101$ and $v = 11101$ they only differ in the second bit string, so the distance between them is 1.

Hamming distance is a distance function since it satisfies the axioms of metric or distance functions for all $x, y, z \in (F_q)^n$

$$\begin{array}{lcccc} M_1 & : & d(x,y) \geq 0 \text{ and } d(x,y) = 0 & & \text{iff } x = y \\ M_2 & : & d(x,y) = & & d(y,x) \\ M_3 & : & d(x,y) \leq & & d(x,z) + d(z,y) \end{array}$$

Let C be a code. The minimum distance of the code denoted $d(C)$, is the smallest distance between distinct code words in the code C,Todd (2005).
In other words, $d(C) = min\{d(u,v)|u, v \in C, u{\neq}v\}$

Theorem 1.1.1. A code C can detect up to s errors in any code word if $d(C) \geq s+1$, Raymond (1986).

Corollary 1.1.1. If a code C has a minimum distance $d(C)$, then C can be used either to detect up to $d(C) - 1$ errors, or to correct upto $\frac{d(C)-1}{2}$ errors in any code word, Raymond (1986).

In any code, there is need for one to know the number of code words that it generates. In this research, the term dictionary is used to refer to the total number of code words that can be generated by a code. In a code C, symbols or digits (redundancy) may be added in order to achieve some degree of uniqeness. These symbols, referred to as the check digits help to ensure that the code satisfies a certain condition, Jacobus (1973). Suppose a sender intends to send a code word to a receiver. If u and v are the sent and received code word respectively, situations may occur such that $u \neq v$, this simply means that error(s) occurred. The ability to detect the existence of an error(s) in a code word is called error detection whereas error correction is the ability to correct the existing error(s) in a code word or to reconstruct the sent code word, Raymond (1986).
A code C is said to be linear in $(F_q)^n$ if whenever $x_0x_1...x_n \in C$, $y_0y_1...y_n \in C$ and $\lambda \in F_q$ then $[(x_0x_1...x_n) + (y_0y_1...y_n)] \in C$ and $\lambda(x_0x_1...x_n) \in C$
A linear code C is said to be a cyclic code if any cyclic shift of a code word in C is also a code word, Van (1998).

That is, if C is cyclic and $x_0x_1...x_{n-2}x_{n-1}x_n \in C$ then $x_nx_0x_1...x_{n-2}x_{n-1} \in C$. This means that cyclic code words can easily be generated by performing right or left cyclic shifts on a code word and continuing until the original code word is obtained, Raymond (1986). For example, suppose *abcd* is a code word in a cyclic code then performing cyclic shifts yields the code words $bcda, cdab, dabc$.

Let k, m, q, n be integers. k is said to be congruent to m modulo n written $k \equiv m(modn)$ if $n|(k-m)$. That is, n "divides" the difference $(k-m)$. In other words, an integer q is found such that $(k-m) = nq$, Kenneth (1993).

Modulo Arithmetic therefore involves working with the remainders generated by division. For example, if 64 is divided by 9, it yields 7 and the remainder is 1. Using modular arithmetic notation, this can be written as $64 \equiv 1(mod\, 9)$ read as "64 is congruent to 1 modulo 9". The following theorems on modulo operations will be of great importance to this research.

Theorem 1.1.2. If k, m, r and n are integers, then $k \equiv m(\text{mod } n)$ if and only if there is an integer q such that $k = m + nq$, *Kenneth (1993).*

Let n be a positive integer. Congruences modulo n is an equivalence relation. That is:

If k is an integer, then $k \equiv k(\text{mod } n)$, the reflexive property.

If k and m are integers such that $k \equiv m(\text{mod } n)$, then $m \equiv k$ (mod n), the symmetric property.

If k, m and r are integers with $k \equiv m$ (mod n) and $m \equiv r$ (mod n), then $k \equiv r$ (mod n), the transitive property, Kenneth (1993).

If k and m are integers with $k \neq 0$, then k divides m(or k is a divisor or a factor of m) if there is an integer r such that $m = kr$, denoted as $k|m$, otherwise $k \nmid m$. The greatest common divisor of two non zero integers k and m, denoted (k, m) is the largest integer that divides both k and m. Integers k and m are said to be relatively prime if $(k, m) = 1$, Kenneth (1993).

An algorithm is a finite set of precise instructions for performing a computation or for solving a problem, Kenneth (1993).

Theorem 1.1.3. ***The Division algorithm***

If k and m are integers such that $m > 0$, then there are unique integers q and r (called the quotient and remainder respectively) such that $k = mq + r$ with $0 \leq r < m$, Kenneth (1993).

Theorem 1.1.4. ***Euclidean algorithm***
Let $r_0 = k$ and $r_1 = m$ be integers such that $k \geq m$. If the division algorithm is successively applied to obtain $r_j = r_{j+1} \cdot q_{j+1} + r_{j+2}$ with $0 < r_{j+2} < r_{j+1}$ for $j = 0, 1, 2, ..., n-2$ and $r_{n+1} = 0$, then $(k, m) = r_n$ the last non zero remainder, Kenneth (1993).

Example 1.1.1. To find $(252, 198)$ using Euclidean algorithm, we find

$$252 = 1(198) + 54$$

$$198 = 3(54) + 36$$

$$54 = 1(36) + 18$$

$$36 = 2(18) + 0$$

Therefore
$$(252, 198) = 18$$

A Diophantine equation is an equation whose solutions come from a set of integers. A linear Diophantine equation in two variables is an equation of the form $kx + my = r$ where k, m and r are integers, Kenneth (2003).

Theorem 1.1.5. Let k and m be integers with $n = (k, m)$. The equation $kx + my = p$ has no integer solutions if $n \nmid p$. If $n|p$, then there are infinitely many integral solutions,Kenneth (1993).

Example 1.1.2. There are no integral solutions of the Diophantine equation $15x + 6y = 7$ since $(15, 6) = 3$ but $3 \nmid 7$, Kenneth (1993).
There are infinitely many solutions of the Diophantine equation $21x + 14y = 70$ since $(21, 14) = 7$ and $7|70$.
By Euclidean algorithm,

$$1 \times 21 + (-1) \times 14 = 7$$

so that

$$10 \times 21 + (-10) \times 14 = 70$$

Hence $x_0 = 10$, $y_0 = -10$ is a particular solution. All solutions are given by $x = 10 + 2n$, $y = -10 - 3n$ where n is an integer, Kenneth (1993).
A permutation is an arrangement of different objects in a specified order. If a number of independent choices were to be made with n possibilities for first choice, n for the second choice and so on the total number of choices is $n_1 \times n_2 \times \ldots n_k$. There are $n!$ ways to arrange n objects in a specified order. For example, eleven books can be arranged on a shelf in $11! = 39916800$ ways.
The number of ways of selecting r objects from n available objects is given as $nCr = \frac{n!}{(n-r!)r!}$, Uppal and Humphreys (2008).

1.1.1 The ISBN-10 Code

In the late 1960's, book publishers realized that they needed a uniform way to identify all the different books that were being published throughout the world. In 1966, Gordon Foster and others came up with the International Standard Book Number system (ISBN) which was later published by the international organization for standardization in 1970, ISO 2108. Every book, including new editions of older books, were to be given a special number, called an ISBN, which is not given to any other book,Viklund (2007) . According to Eric (2010a), the ISBN-10 code consist of ten digits code words made up of any of the 10-digit decimal numbers namely $0, 1, 2, \ldots 9$ and X for 10. Suppose $u = a_1a_2a_3...a_{10}$ is an ISBN-10 code word, it must satisfy the condition

$$\sum_{i=1}^{10} ia_i \equiv 0(mod 11) \tag{1.1}$$

This equation is called a parity-check equation. An ISBN is broken into groups written with hyphens within it. For example in a code word

$$a_1a_2 - a_3a_4a_5 - a_6a_7a_8a_9 - a_{10}$$

the first block of digit, "a_1a_2" , represents the language or the country of the book, the second block of digits represents the publishing company, the third block of digits is the number assigned to the book by the publishing company and the last digit is known as the check digit. For example, the book "A first course in coding theory" by Raymond Hill, has an ISBN-10: 01-985-3803-0, Raymond (1986). The ISBN parts may be of different lengths, and usually are separated with hyphens or spaces,Viklund (2007) .

In the event that the check digit is 10, the symbol X is used in the final position. The summation $\sum_{i=1}^{10} ia_i$ is called the weighted check sum of the code word $a_1a_2 \ldots .a_{10}$. If the condition is satisfied then $a_1a_2 \ldots a_{10}$ is an ISBN-10, Egghe (1985).

For example, consider the ISBN-10: 0198538030 given above. The sum

$$\begin{aligned}\sum_{i=1}^{10} ia_i &= 0 + 2 + 27 + 32 + 25 + 18 + 56 + 0 + 27 + 0 \\ &= 33 \equiv 0\,(mod\,11)\end{aligned}$$

Thus the code word is an ISBN-10 code word.

The code uses calculation modulo 11 since 11 is a prime number and since it has no factors, all the multiples of 11 would yield to 0 modulo 11.

If $u = a_1a_2 \ldots a_{10}$ and $v = b_1b_2 \ldots b_{10}$ are the sent and received code word respectively, error(s) may occur to yield $u \neq v$. The ISBN-10 code is designed to detect a single and a double-error which can be caused by the transposition of two digits. To detect an error(s) on a received vector $b_1b_2\ldots b_{10}$ the weighted check sum $B = \sum_{i=1}^{10} ib_i$ is computed. If $B \neq 0(mod 11)$, then an error(s) exists. The ISBN-10 code cannot be used to correct an error unless the digit in error is found. This is the basis of the parity check equation, Raymond (1986).

1.1.2 Limitations in the ISBN-10

1. Silent errors can go unnoticed.

2. To detect an error in ISBN-10 code word, one has to work out the parity check equation which makes the process tedious unless the error is an omission or insertion error which obviously can be detected by just counting the number of digits to confirm if they are less or more than required.

3. ISBN-10 can only correct single errors and transpose errors and for the transpose errors there has to be prior knowledge of the existence of the transpose error. The process of correcting the errors as stated in limitation 2 above may end up being tedious and only applicable to code words which meet some criteria as shown in equation (2.5) later on.

4. The conditions given for detection of at least two errors only tell if one can detect the double error and do not show how to correct it, see section 2.2.3.

5. Multiple (more than two) errors cannot be corrected.

6. ISBN-10 has a relatively small dictionary.

The great demands for a bigger dictionary lead to the design of ISBN-13 code which is now in use since year 2007. There is need to show if the improved ISBN-13 code dictionary size affected the error detection and correction capability.

1.1.3 The Cyclic ISBN-10

The Cyclic International Standard Book Number was designed by Nyaga and others to improve the conventional ISBN-10. It consists of ten digit codewords made up of any of the 10-digit decimal numbers $0, 1, 2, \dots, 9$ and X for 10 with the check digit being chosen such that the parity check equation holds just as in ISBN-10. To come up with a code word, digits are arranged in an increasing or decreasing order of flow. Zero is not allowed to be the first digit. Repetition of digits is allowed; change of order of flow of digits (from increasing to decreasing and vice versa) is done by putting a zero (0) and then starts the new flow which may be either increasing or decreasing. The check digit does not necessarily have to obey the flow of the digits, Nyaga and Cecilia (2008).

1.1.4 Limitations in the Cyclic ISBN-10

1. This code is not a cyclic code since any left or right cyclic shift does not necessarily yield another codeword. By counter example, consider the code word 7765442112. Taking a right cyclic shift yields to 7654421127 which is not a cyclic ISBN-10 code word since it does not meet the parity check equation. There is thus the need to clearly redefine the term cyclic to fit the exact intended meaning when the code was generated.

2. The dictionary size is smaller than even the ISBN-10 code. As shown later in Proposition 2.3.1, the total number of code words is less than 9×10^8. There is thus need to establish the upper limits and the lower limits for maximum and minimum total number of code words that this code can generate. This forms a major foundation for any other code that would be developed using Nyaga's approach.

3. Correction of multiple errors may end up having very many choices of possible digits to replace the ones in error thus making the process a bit tedious and inacurate.

1.1.5 The ISBN-13 Code

The ISBN-13 code is a 13 digit code whose use started in January 2007. According to Viklund (2007), the code is generated such that the calculation of its check digit begins with the first 12 digits, thus excluding the check digit itself. If $u = x_1x_2x_3...x_{12}x_{13}$ is a code word, then

$$x_{13} = 10 – (x_1 + 3x_2 + x_3 + 3x_4 + x_5 + 3x_6 + x_7 + 3x_8 + x_9 + 3x_{10} + x_{11} + 3x_{12})(mod 10) \tag{1.2}$$

$$i.e, x_{13} = 10 – (x_1 + x_3 + x_5 + x_7 + x_9 + x_{11}) - 3(x_2 + x_4 + x_6 + x_8 + x_{10} + x_{12})(mod 10) \tag{1.3}$$

To convert an ISBN-10 codeword to ISBN-13, the last character is dropped (left out) and the numbers "978" or "979" are put at the beginning of the code and a new check digit is computed to form the 13th digit as per equation 1.2.
For example, the book "*A first course in coding theory*" by Raymond Hill has an ISBN-10 of 0198538030. To convert to ISBN-13, the last digit (check digit) 0 is dropped, 978 added to the beginning to yield 978019853803$\boldsymbol{a}_{13}$, the check digit is computed as follows:

$$9 + (7 \times 3) + 8 + (0 \times 3) + 1 + (9 \times 3) + 8 + (5 \times 3) + 3 + (8 \times 3) + 0 + (3 \times 3)$$
$$= 125 \equiv 5\,(mod\,10)$$

Thus, $x_{13} = 5$ yielding to 9780198538035.

The prefix 978 or 979 is used in ISBN-13 to ensure the code is integrated into the European article number which is adopted for all products retailed internationally. This check digit computation system does not detect all errors of adjacent digit transposition. Specifically, if the difference between two adjacent digits is 5, the check digit will not catch their transposition,Viklund (2007). For instance, suppose we have a code word having bit strings 6 followed by a 1 at consecutive digit positions. If 6 is in an even digit position, the order contributes $(3 \times 6) + (1 \times 1) = 19$ to the sum; while, if the digits are transposed (1 followed by a 6), the contribution of those two digits will be $(3 \times 1) + (1 \times 6) = 9$. However, 19 and 9 are congruent modulo 10, and so produce the same, final result: both ISBNs will have a check digit of 7.
Additionally, Eric (2010a) researched on the ISBN-13 code and found out that, if one triples the sum of the $2^{nd}, 4^{th}, 6^{th}, 8^{th}, 10^{th}$ and 12^{th} digits and then add them to the remaining digits ($1^{st}, 3^{rd}, 5^{th}$, etc.), the total will always be divisible by 10. Eric noted that the ISBN-10 formula uses the prime modulus 11 which avoids this blind spot, but requires more than the digits 0-9 to express the check digit, Eric (2010a). Contrary to the ISBN-10 which gives a general equation that the code must obey 1.1, the ISBN-13 incoporates a formula for finding the check digit and does necessarily generalize the condition for the code to obey. There is thus a need to generalize this by finding an equation that the code has to obey.

Proposition 1.1.1. The ISBN-13 code must satisfy the condition;

$$\sum_{i=0}^{6} x_{2n+1} + 3\sum_{i=1}^{6} x_{2n} \equiv 0(mod10) \tag{1.4}$$

Proof. From equation 1.2 above,

$$x_{13} = 10 - (x_1 + 3x_2 + x_3 + 3x_4 + x_5 + 3x_6 + x_7 + 3x_8 + x_9 + 3x_{10} + x_{11} + 3x_{12})(mod10) \tag{1.5}$$

or simply

$$\begin{aligned} x_{13} + (x_1 + 3x_2 + x_3 + 3x_4 + x_5 + 3x_6 + x_7 + 3x_8 + x_9 & \\ +3x_{10} + x_{11} + 3x_{12})(mod 10) &= 10 \equiv 0(mod 10) \end{aligned}$$

Thus,

$$\begin{aligned} &(x_1 + 3x_2 + x_3 + 3x_4 + x_5 + 3x_6 + x_7 + 3x_8 + x_9 + 3x_{10} \\ &\ + x_{11} + 3x_{12} + x_{13})(mod 10) = 10 \\ &\ \equiv 0(mod 10) \end{aligned}$$

Thus,

$$\begin{aligned} &(x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + x_{13})(mod 10) \\ &\ + (3x_2 + 3x_4 + 3x_6 + 3x_8 + 3x_{10} + 3x_{12})(mod 10) \equiv 0(mod 10) \end{aligned}$$

or simply,

$$\sum_{i=0}^{6} x_{2i+1} + 3\sum_{i=1}^{6} x_{2i} \equiv 0(mod 10)$$

□

1.1.6 Limitations in the ISBN-13 code

1. The ISBN-13 code cannot indicate the exact bit string which is in error

2. The ISBN-13 code cannot detect some transposition errors as shown later in section 3.1.3

3. ISBN-13 cannot detect some double or multiple errors. For example, suppose that $u = 9780198538035$ is the sent code word but due to errors the receiver receives the code word $v = 8780298538035$.

   For the sent code word u,

   $x_{13} = 5 = 10–(9 + (3 \times 7) + 8 + (3 \times 0) + 1 + (3 \times 9) + 8 + (3 \times 5) + 3 + (3 \times 8) + 0 + (3 \times 3))(mod 10)$

For the received code word v,

$x_{13} = 5 = 10–(8 + (3 \times 7) + 8 + (3 \times 0) + 2 + (3 \times 9) + 8 + (3 \times 5) + 3 + (3 \times 8) + 0 + (3 \times 3))(mod 10)$

Thus v will erroneously be received as the intended code word but it is **not!** This is an example of a double error.

4. ISBN-13 cannot correct the double, multiple or transposition errors. This is proven later on in theorem 3.1.1.

1.2 Statement of the Problem

The ISBN-10, ISBN-13 and Cyclic ISBN-10 all lack a full combination of the ability to detect and correct multiple errors and at the same time generate a big dictionary.

1.3 Objectives of the Study

1.3.1 General Objective

The main objective of this research is to determine and compare the effect on the dictionary size brought about by varying conditions applied to each of the code and to design a code that improves error detection and correction yet maintaining and possibly increasing the dictionary size compared to the other three codes.

1.3.2 Specific Objectives

1. Determine the size of the dictionaries in the ISBN-10 and ISBN-13 codes against the error detection and correction capabilities.

2. Determine the efficiency of the Cyclic ISBN-10 code in error detection and correction against the size of the dictionary.

3. Develop a new code that improves the ISBN-10, ISBN-13 and Cyclic ISBN-10 codes in error detection and correction capabilities against the dictionary size.

4. Design and develop a computer program that generates the new code

5. Generate a conversion tool for the already existing codes to the new developed code.

1.4 Justification of Study

Different codes are used to identify cars, magazines, books, journals among others. In Kenya, seven (7) digits are used to code registered vehicles. The need for an extra digit arose from the once nationally recognized six (6) digit method which speedily ran out of codes. This means that the authorities involved realized the danger and need for the change and provided a viable solution which has so far worked well. The ISBN code uniquely identifies a book title.

This means that no two or more different books titles in the universe should share an ISBN. Libraries, companies and even individuals may therefore use the ISBN of a book to order for supplies from publishers or distributors. Libraries may use the ISBN in catalogueing books on their shelves for easy identification.

Without loss of generality, suppose in a day ten thousand (10,000) different books are published in the world. In a span of a millennium, ten billion (10,000,000,000) code words shall be required to cater for these books. This means that the world needs a code that can if possible generate unending number of code words to cater for this major need for millions of years ahead of us. In any environment,

noise, electromagnetic radiations and any other forms of disturbances affect communication leading to errors in the received messages or even to an extent of the message not being received at all. It is due to this reason that importance is thus attached to finding means of detecting and correcting any error that occur. Thus the need for a code that fully guarantees security in the sense that whenever two or more persons send or receive this code then data integrity and authentication are guaranteed. Occurrence of errors during communication involving a buyer to and from a seller may lead to the buyer receiving the wrong book, delaying the process or even missing his intended book completely incase the received ISBN does not exist. Faulty codes generation and consecutive printouts by publishers may lead to two or more books sharing the same ISBN which may cause great confusion and losses in the hands of buyers and the publishers.
This code will successfully improve the cyclic ISBN-10 code as far as multiple error detection and correction and increase the dictionary. It will also challenge the existing ISBN-13 code in multiple error detection and correction. The scope of this research could allow enquiry for calculation of actual number of code words (the dictionary) that this code can provide. A computer program to do this and also to aid in faster error detection and correction remains an interesting problem to be tackled in the near future.

1.5 Null Hypothesis

- The designed ISBN-16 code cannot detect or correct multiple errors and does not have sufficient dictionary.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

In 1986, Raymond researched and wrote on ISBN-10 code to show that it was designed to detect a single error and a double-error which can be caused by the transposition of two digits. Raymond gave some conditions as discussed earlier in section 2.2.1 that must be satisfied for error correction. He noted also that ISBN-10 code cannot be used to correct an error unless one knows the digit in error, this is the basis of the parity check equation.

In 1993, Kenneth researched on the ISBN-10 code and showed the check digit was a remainder upon division by 11 of a weighted sum of the first nine digits. He also showed that a single error or a transposition of two digits could be detected using the check digit and that it is possible to detect and correct errors, Kenneth (1993). In 1993, Henk described a communication channel and noted that it is made up of the sender, encoder, channel, decoder and receiver as shown in figure 2.1 below;

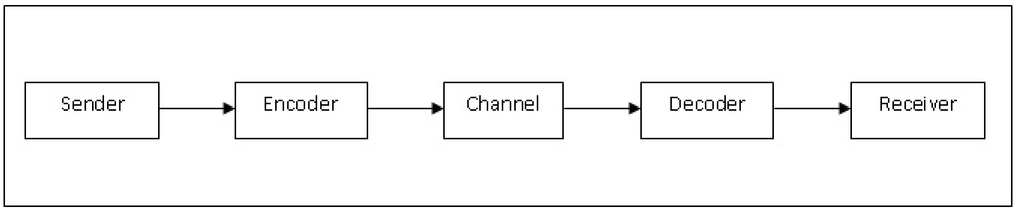


Figure 2.1: Communication Channel

In 1998, Tervo researched on the secrets of ISBN-10 and stated that "*Error control codes are often designed for known applications where certain types of errors are* *expected to occur. In this case, the most common errors expected would be those which humans would typically make when writing or typing a book order. These errors would normally be either "write a single digit incorrectly", or "switch two adjacent digits". Of course, the ability to detect errors also allows identification of invalid numbers which might be encountered (for example, if someone were to forge a credit card number). Forgery is not expected to be a real concern for book numbers, but error detection is very important*". Tervo (1998) goes ahead to note that "*A systematic approach to error detection can lead to better codes*"

.

Doumen (2003), researched on the aims of cryptography in providing secure transmission of messages in the sense that two or more persons can communicate in a way that guarantees confidentiality, data integrity and authentication.
Sebastia (2003) studied on the Block error correcting codes. He found that the minimum distance decoder maximizes the likelihood of correcting errors if all the transmission symbols have the same probability of being altered by the channel noise. He also noted that if a code has a minimum distance d, then $d(C) - 1$ is the highest integer with the property that the code detects $d(C) - 1$ errors.
Todd (2005), studied on the Error control coding. He showed that if a communication channel introduces fewer than the minimum distance errors, $d(C)$, then these can be detected and that if $d(C) - 1$ errors are introduced, then error detection is guaranteed. He also noted that the probability of error detection depends only on the error introduced by the communication channel and that the decoder will make an error if more than half of the received bit strings are in error.
Egghe (2005), studied the Coding of the ISBN. He concluded that the minimum requirement for a useful code was that all single errors as well as all permutations of two symbols must be detectable. They also discussed the strength of alternative codes, in particular with respect to the detection of double errors. They gave a

complete description of the method based on division by 11, described the power of the method with respect to the detection of three or four errors.

Egghe and Ronald (2005), studied the Detection of Double Errors in ISBN and ISSN-like codes and showed that, using division by 11, all coding methods detect the same percentage of errors. It was also shown using numerical experiments that in case larger prime numbers are used as a divisor, different methods have different detection capabilities for double errors.

Egghe (1999), also studied the Detection and Correction of Multiple Errors in General Block Codes and found the necessary condition for systems to be able to detect all $1, 2, ..., (k-1)$ errors ($k \in \mathbb{N}$) and showed that the results have applications in ISBN. He also showed that if the system detects all $1, 2, ..., k$ errors ($k \in \mathbb{N}$), then it corrects all $1, 2, ..., k$ errors.

Egghe and Ronald (2005), noted that a minimum requirement for a useful code is that all single errors as well as all permutations of two symbols must be detectable. They then left an open unsolved problem challenging the readers to Formulate a new algorithm, with one check digit (or letter), such that all single and double errors are detected (or prove that this is not possible).

In 2009, Nyaga and others studied the Cyclic ISBN-10 to improve the conventional ISBN-10. They designed a code that would detect and correct multiple errors without many conditions attached for error correction and found out that the code could correct as many errors as the code could detect. The method involves trial and error calculation and thus it needs to be improved on and simplified to speed up the process.

Eric (2010a), studied the ISBN-10 code and showed that it consist of ten digits code words made up of any of all 10-digit decimal numbers $0, 1, 2, \ldots 9$ and X for 10 such that if $u = a_1a_2a_3 \ldots \ldots a_{10}$ is an ISBN-10 code word, it must satisfy the parity check equation $\sum_{i=1}^{10} ia_i \equiv 0(mod11)$.

2.1.1 Types of errors in an ISBN code

The strength of a code may be measured by its ability to detect and correct errors. To achieve this goal, the encoding and decoding process should be as error free as possible. A code that does not guarantee this or has many weaknesses may end up being unreliable no matter how big the dictionary is. The following are errors that may exist in an ISBN code; the definitions are made in reference to ISBN-10;

1. Single error: A single error may occur as a result of incorrect typing of one digit in an ISBN code word or as a result of a smudge. The parity check equation does not hold in this case. For example writing 1 in place of 7 is a common typing single error, Raymond (1986).

2. Double error: A double error occurs when two digits in a code word are incorrect. The parity check equation may not hold in this case, Raymond (1986).

3. Silent error: A silent error occurs when the parity check equation holds despite the sent and received code words differing in some bitstrings. This means that the received code word may be an ISBN code word since it will obey the parity check equation but it will not be code word that the sender intended the receiver to receive. For example, let $u = 0198538030$ be the sent code word and let $v = 0194538230$ be the received code word. Working out for parity check equation;

   for u, $(1 \times 0) + (2 \times 1) + (3 \times 9) + (4 \times 8) + (5 \times 5) + (6 \times 3) + (7 \times 8) + (8 \times 0) + (9{\times}3) + (10{\times}0) \equiv 0(mod 11)$

   for v, $(1 \times 0) + (2 \times 1) + (3 \times 9) + (4 \times 4) + (5 \times 5) + (6 \times 3) + (7 \times 8) + (8 \times 2) + (9 \times 3) + (10 \times 0) \equiv 0(mod 11)$

Despite that v differs from u, the parity check equation is met in both cases. This means that both u and v are ISBN code words but the receiver did not receive the desired message that the sender intended. The mistyping was an error but since the receiver received a message which the decoding process could only identify as valid, v will be erroneously received as the sent code word leading to a silent error.

4. Transpose error: A transpose error may occur due to interchanging of digits in a code word, Raymond (1986). For example writing 69 in place of 96 is a transpose error. Since computation is done modulo 11 which is a prime number, the error will be noticed. If operations were done modulo n, where m is composite, then it would be possible for transpose error be silent. This would happen when the digits interchanged also yield the same result when multiplied by their respective multipliers modulo n.

5. Omission or insertion of a digit(s) error: These are errors that occur when a digit(s) are omitted or extra digit(s) are added. Clearly the code will have less (or more) than the required digits. The parity check equation does not hold in this case. For example consider $u = 01945382310$. Clearly, u has 11 digits instead of 10 digits. One may assume that the last two digits, 1 and 0 are representing the number 10 in which case the parity check equation is computed and if it holds, then X should be used in place of 10 and u would be a code word. Otherwise, u is not an ISBN-10 code word since an ISBN-10 code word has 10 digits and must satisfy the parity check equation.

2.2 ISBN-10 Code

2.2.1 Error Detection in ISBN-10 code

According to Raymond (1986), the ISBN code is designed to detect single error and also double error which comes as a result of transposition of two digits.
For a received vector $v = b_1b_2 \ldots b_{10}$ its weighted check sum $B = (\sum_{i=1}^{10} ib_i)$ is computed. If $B \neq 0(mod 11)$, then error(s) have been detected.
Suppose $u = a_1a_2...a_{10}$ is the code word and suppose u is similar to v except that digit a_j is received as $a_j + t$ with $t \neq 0$.
Then,

$$B = (\sum_{i=1}^{10} ia_i) + jt = jt \neq 0(mod 11) \tag{2.1}$$

This is because j and t are non-zero digits less than11. The ISBN code cannot be used to correct an error unless it is known that just one given digit is an error. This is the basis of the parity check.

2.2.2 Number of code words in ISBN-10 Code

To calculate the total number of code words, it is first established how many bit strings can permute. For an ISBN-10 code, the check digit is not just chosen. It is computed such that the parity check equation holds. There is need to show that there is no chance for the check digit to permute freely meaning that it is only chosen for the specific code word. This is discussed later in section 3.1.2.
Since the check digit does not permute, the code has 10 bit strings, only the first 9 bit strings can be allowed to permute (9 bit strings are chosen from 10 digits namely 0 to 9) and then the check digit computed for each code word generated. Since the digit repetition for the first nine positions is allowed, there are 10^9

permutations. Thus if no other conditions are attached to the code (for example error detection and correction), it has a dictionary of $10^9 = 1,000,000,000$ code words. If digit repetition was not allowed, there would have been ${}_{10}P_9$ permutations which yield to $3,628,800$ code words. Considering the conditions discussed above that must be met for a code to detect and correct errors the following proposition is of importance.

Proposition 2.2.1. Let u be an ISBN-10 code word. Then u does not necessarily satisfy all the conditions given in equations (2.4) and (2.5)

Proof. (By counter example): Consider the code word $u = 0198538030$

$$\sum_{i=1}^{10} ia_i \equiv 0(mod 11)$$

Therefore $u = 0198538030$ is an ISBN-10 code word.

But,

$$\sum_{i=1}^{10} a_i \equiv 4(mod 11) \neq 0(mod 11)$$

Since one of the conditions is not satisfied, then u does not necessarily satisfy all the conditions. □

This result shows that not every ISBN-10 code word will satisfy all the conditions. Thus if the conditions are imposed so that the code can correct errors, then some code words will be excluded. There is thus a need to show how each of these conditions affect the dictionary size and by how much.

2.2.3 Error Correction in ISBN-10 code

Raymond (1986) describes syndrome-decoding scheme that corrects any single error and which simultaneously detect any double error arising from the transposition of two digits of a code word. Suppose u and v are the sent and received code words respectively,

The system $(A, B) = (\sum_{i=1}^{10} y_i, \sum_{i=1}^{10} iy_i)$ is calculated $modulo 11$.

Suppose a single error has occurred, so that for some non-zero integers j and k, for the received vector v,

$$A = \sum_{i=1}^{10} y_i = \sum_{i=1}^{10} y_i + k \equiv k(mod 11) \tag{2.2}$$

$$B = \sum_{i=1}^{10} iy_i = \sum_{i=1}^{10} iy_i + jk \equiv jk(mod 11) \tag{2.3}$$

The error magnitude k is given by A and the error position j is given by the value of $B|A$ which is calculated as BA^{-1}. Hence the decoding scheme is, after calculating (A, B) from v, as follows:

1. If$(A, B) = (0, 0)$, then v is a code word and it is assumed that there are no errors.

2. If $A \neq 0$ and $B \neq 0$, then it is assumed that there is a single error which is corrected by subtracting A from the$(B/A)^{th}$ entry of v.

3. If $A = 0$ or $B = 0$ but not both, then at least two errors have been detected. Case (3) always arises if two digits of a code word have been transposed, for then $A = 0$ and (as for the ISBN code)$B \neq 0$.

For example, suppose$v = 0610271355$. Computation yields $A = 8$ and $B = 6$. Hence, $B/A = 6 \times 8^{-1} = 6 \times 7 = 42 = 9$ and so the 9^{th} bit string should have been 5–8 $= -3 = 8$. Raymond showed that, to correct a single error the sent code word, has to satisfy the following condition:

$$\sum_{i=1}^{10} a_i = \sum_{i=1}^{10} ia_i \equiv 0(mod 11) \tag{2.4}$$

Similarly, to correct a transpose error the sent code word has to satisfy the following:

$$\sum_{i=1}^{10} a_i \equiv \sum_{i=1}^{10} ia_i \equiv \sum_{i=1}^{10} i^2 a_i \equiv \sum_{i=1}^{10} i^3 a_i \equiv 0(mod 11) \tag{2.5}$$

2.3 Cyclic ISBN-10 Code

2.3.1 Calculation of the inverses in $(\mathbb{Z}_{11}^{*}, \times, +)$

Under the operation addition

According to Nyaga and Cecilia (2008), if u and v are elements in the field $\mathbb{Z}_{11}$, u is said to be the additive inverse of v (denoted by $-v$) if $v + u \equiv 0(mod 11)$. Since 0 is the additive identity, the following result hold.

Table 2.1: Inverses under addidtion for cyclic ISBN-10

v	1	2	3	4	5	6	7	8	9	10
$-v$	10	9	8	7	6	5	4	3	2	1

Under operation multiplication

According to Nyaga and Cecilia (2008), if u and v are non zero elements in the field $\mathbb{Z}$, u is said to be the multiplicative inverse of v (denoted by v^{-1}) if $u \times v \equiv 1(mod 11)$. Since 1 is the multiplicative identity, the following result hold.

Table 2.2: Inverses under multiplication for cyclic ISBN-10

v	1	2	3	4	5	6	7	8	9	10
v^{-1}	1	6	4	3	9	2	8	7	5	10

2.3.2 Calculation of the check digit in Cyclic ISBN-10 code

Let u=$a_1a_2...a_{10}$ be a code word. To calculate the check digit a_{10}, calculate $P = \sum_{i=1}^{9} ia_i$, for $i = 1, 2, \ldots, 9$

Thus $P + 10a_{10} \equiv 0(mod 11)$ or simply $10a_{10} \equiv -P(mod 11)$. Hence, $a_{10} = (10^{-1})(-P)(mod 11) \equiv 10(-P)(mod 11) \equiv -10P(mod 11)$

But since $-10 \equiv 1 mod 11$,

$$a_{10} \equiv P(mod 11) \tag{2.6}$$

Nyaga and Cecilia (2008).

Example 2.3.1. Calculate the check digit, a_{10}, for the code word $764302478a_{10}$

Solution:

$P = \sum_{i=1}^{9} ia_i \equiv 2(mod 11)$. From equation 2.6 above, $a_{10} = P$ thus $a_{10} = 2$

Other examples of Cyclic ISBN-10 code words include:

1234556782, 0112344569, 0123456789, 7765442112, 6542211098

2.3.3 Error Detection in Cyclic ISBN-10

To detect whether or not a code word has an error, the flow of the digits in the code word is checked and if any digit breaks the flow then error(s) exist. The parity check equation is then evaluated and if the result is not zero modulo eleven then there are error(s) even if the flow of the digits may be correct. To correct this error, a digit is chosen such that it satisfies the conditions of being a cyclic ISBN-10 code, Nyaga and Cecilia (2008).

2.3.4 Single Error Correction in Cyclic ISBN-10

To correct an error, the procedure starts by detecting the error position as explained earlier. If the error is at position $i = k$, for $1 \leq k \leq 10$, and suppose the correct digit is m, then according to Nyaga and Cecilia (2008), the following results follows,

$$mk + \sum_{i\ =1, i \neq k}^{10} ia_i \equiv 0(mod11) \tag{2.7}$$

Thus,

$$mk = - \sum_{i=1, i \neq k}^{10} ia_i(mod11)$$

which yields to

$$m = (k^{-1}) \times (- \sum_{i=1, i \neq k}^{10} ia_i(mod11)) \tag{2.8}$$

.

For example, suppose a code word is received as $y = 9876354209$. Nyaga and Cecilia (2008), observed that the digits are written in a decreasing order except the 5^{th} position where the sequence starts increasing and all over sudden at 6^{th} position it starts decreasing again. There is thus an error in 5th position. The correct digit can be computed as

$$m = (5^{-1}) \times (-\sum_{i=1, i\neq 5}^{10} ia_i(mod11)) \tag{2.9}$$

But

$$\sum_{i=1, i\neq 5}^{10} ia_i(mod11) \;=\; 234 = 3(mod11) \tag{2.10}$$

Since $5^{-1} = 9$, and $-3 \equiv 8(mod11)$, we have $m \;= 9 \times 8 \;= 72 \;\equiv 6(mod11)$. Replacing with 6 yields 9876654209 whose weighted sum is $264 \equiv 0(mod11)$ hence is a Cyclic ISBN-10 code word.

Alternatively using the trial and error method to correct this error, Nyaga and Cecilia (2008) observed that the only digits that would replace 3 in this position without breaking the order of flow were only 5 or 6. Replacing with 5 yields the code word 9876554209, whose weighted sum is $269 \neq 0(mod11)$ which does not satisfy the parity check equation, hence not a Cyclic ISBN-10 code word. Replacing with 6 yields 9876654209 whose weighted sum is $264 \equiv 0(mod11)$ hence is a Cyclic ISBN-10 code word, Nyaga and Cecilia (2008).

2.3.5 Double Error Correction in Cyclic ISBN-10

To correct a double error, suppose the error is at position a and b, for $1 \leq a \leq 10$ and $1 \leq b \leq 10$ and suppose the correct digit is m and n at positions a and b respectively, then the following results follows,

$$am + bn + \sum_{i\ =1, i\neq a, i\neq b}^{10} ia_i \equiv 0(mod11) \tag{2.11}$$

Through trial and error, m and n are chosen such that the correct order of flow of digits is obeyed and the parity check equation is satisfied, Nyaga and Cecilia (2008).

Example 2.3.2. If $y = 7165042112$ the digits at the 2^{nd} and 5^{th} position break the order of flow and that the weighted sum yields to $122 \neq 0(mod 11)$.
Only 7 or 6 can restore the order of flow at the 2^{nd} position whereas only 5 or 4 can restore the order of flow at the 5^{th} position. Using trial and error method, replace 7 or 6 to the 2^{th} position then replace 5 or 4 to the 5^{th} position and then the weighted sum is computed to check if it is congruent to zero modulo eleven. This gives 4 options namely 7 and 5 or 7 and 4 or 6 and 5 or 6 and 4. Working out yields 7 and 4 as the correct digits when replaced in the 2^{nd} and 5^{th} positions respectively. Thus the code word is 7765442112, whose weighted sum is $\sum_{i=1}^{10} ia_i = 264 \equiv 0(mod 11)$

The Cyclic ISBN-10 therefore can correct the double errors in the received code word.

2.3.6 Number of code words in Cyclic ISBN-10

To calculate the total number of code words, we first establish how many bit strings can permute. Just as in ISBN-10 and ISBN-13, the check digit does not permute. It is chosen to satisfy the parity check equation. According to Nyaga and Cecilia (2008), this code limits 0 from being the first digit so that this position can only be filled in 9 ways. The remaining 8 bit strings can be occupied by any of the 10 digits $0, 1, 2, \ldots, 9$ thus 10^8 permutations. Thus in total there are $9 \times 10^8 = 900,000,000$ code words, Nyaga and Cecilia (2008). This calculation however faces some oversight as shown in the proposition 2.3.1 below.

Proposition 2.3.1. The cyclic ISBN-10 generates less than 9×10^8 code words.

Proof. (By counter example): Suppose the first bit string is "2" and the second bit string is "3". To generate the third bit string, it must be chosen such that the flow of the bit strings is not broken. Hence it can neither be "1" nor "2".

Therefore this position can only be occupied by 0, or 3, or 4, or ... or 9. These are only eight choices not ten choices.
Similarly, if one has a code word with 543... as the first three bit strings, the fourth bit string cannot be $4, 5, 6, 7, 8$ or 9. Otherwise it will alter the flow of the bit string. Thus for this position, only $0, 1, 2$ or 3 can be chosen yielding to four choices not ten choices. Thus the total code words do not necessarily add up to 9×10^8 code words. □

2.4 ISBN-13 Code

2.4.1 Error Detection in ISBN-13 code

Suppose v is the received code word. To detect an error, computation is done to check if the parity check equation 1.2 holds, if not then error(s) exist in the received code word v. The error could have been as a result of the sources of errors discussed earlier. Contrary to ISBN-10, the ISBN-13 does not indicate the exact bit string which is in error. It only identifies the existence of an error in a code word but does not correct it thus one must request for retransmission if an error is detected.

2.4.2 Number of code words in ISBN-13 code

To calculate the total number of code words, it is first established how many bit strings can permute. For an ISBN-13 code, the check digit is not just chosen; it is computed such that Equation 1.2 holds. Just as in ISBN-10, there is thus need to show that there is no chance for the check digit to permute freely meaning that it is only chosen for the specific code word. This is discussed later in Proposition 3.1.4. This result emphasizes that though the code has 13 bit strings, only the first 12 bit strings permute (12 bit strings are chosen from 10 digits namely 0 to 9) with the 13th bit string, the check digit, computed for each code word generated. Since digit repetition for these permuting positions is allowed, there are 12^{10} permutations. Thus if no other conditions are attached to the code (for example error detection and correction), it has a dictionary of $12^{10} = 61,917,364,224$ code words. This clearly shows that the ISBN-13 code surpasses the dictionary size in ISBN-10 code. If digit repetition was not allowed, there would have been ${}_{12}P_{10}$ permutations which would yield to $239,500,800$ code words. The repetition of digits is thus very important and inevitable due to the great demand for a big dictionary that the repetition offers.

Remark. As discussed earlier in section 1.1.1, book publishers saw the need for a uniform way to identify all the different books that were being published throughout the world assigning each book a unique code word. If a reader is interested in purchasing a book coded u, he/she sends this code word to a seller. Suppose in the process due to errors the seller receives the code word v which is also a valid code word but different from u. The reader will end up purchasing the wrong book thus to avoid this, both parties may need to reconfirm if the sent and received code words are the same. This makes the process long and expensive. A code should thus not only guarantee the uniqueness of a book for cataloguing but also ensure it avoids this blind spot. ISBN-13 was designed as an improvement to the earlier code, ISBN-10 but as discussed earlier in section 1.1.6, this was not fully achieved. There is also a need to discuss the error detection and correction capabilities of the ISBN-10 and show how the number of codewords generated is affected. This is done in chapter 2. There is therefore great need to generate a code that improves and if possible completely resolves the weakness in the ISBN code as far as error detection, error correction and dictionary size are concerned. It is due to this great demand that this research is undertaken.

CHAPTER THREE

ISBN CODE PROPERTIES

This chapter sets the methodology of the research by discussing the properties of the ISBN-10, ISBN-13 and the Cyclic ISBN-10 codes and using the research gaps established to form the basis of the research results. It then eventually introduces the development and generation of the resulting ISBN-16 code

3.1 Properties of ISBN-10, ISBN-13 and Cyclic ISBN-10 codes

3.1.1 Number of code words in ISBN-10, ISBN-13 and Cyclic ISBN-10 codes

In this section, the total number of code words for each of the three codes is determined and when not possible, the upper bound is determined.

Proposition 3.1.1. The ISBN-10 code has a dictionary with an upper limit of 10^9 code words.

Proof. The proof was shown earlier in section 2.2.2. □

Proposition 3.1.2. The cyclic ISBN-10 code has a dictionary with an upper limit of 9×10^8 code words.

Proof. As seen earlier in section 2.3.6, the first bit string can only be filled in nine ways. The next eight bit strings can be occupied by any of the 10 digits $0, 1, 2, \ldots, 9$ thus 10^8 choices. In total the maximum number of code words that can be generated is 9×10^8 code words. This is the dictionary's upper limit for the code. □

Proposition 3.1.3. The ISBN-13 code has a dictionary of $12^{10} = 61,917,364,224$ code words.

Proof. The proof was discussed earlier in section 2.4.2 , □

Proposition 3.1.4. In an ISBN-10 code, the check bit string does not permute.

Proof. Computation for the check digit is done modulo 11. Let u and v be two ISBN-10 code words given by $u = a_1a_2 \ldots a_9a_{10}$
and $v = a_1a_2 \ldots a_9b_{10}$ (similar in the first nine bit strings and suppose the check digit can permute). Since u and v are code words then,

$$(\sum_{i=1}^{9} ia_i) + 10a_{10} \equiv \quad 0(mod 11) \quad \text{and} \quad (\sum_{i=1}^{9} ib_i) + 10b_{10} \quad \equiv \quad 0(mod 11)$$

But u and v are similar in the first 9 bit strings. Thus, $10a_{10} = 10b_{10} mod 11$. By cancellation law for fields or integral domains, $a_{10} = b_{10}$. Hence the check bit string does not permute. □

Proposition 3.1.5. In an ISBN-13 code, the check bit string does not permute.

Proof. Let u and v be two ISBN-13 code words given
by $u = a_1a_2a_{11} \ldots a_{12}a_{13}$ and $v = a_1a_2 \ldots a_{11}a_{12}b_{13}$ (similar in the first twelve bit strings and suppose the check digit can permute), then
$a_{13} = 10–(a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

and
$b_{13} = 10–(a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

But since u and v are similar in the first 12 bit strings then $a_{13} = b_{13}$. Hence the check bit string does not permute. □

3.1.2 Error detection and (or) correction capability in ISBN-10, ISBN-13 and Cyclic ISBN-10 codes

In this section, the ability to detect or (and) correct error(s) is determined and analyzed

Proposition 3.1.6. In an ISBN-13 code, any transposition of adjacent bit strings on a sent code word that yields same weighted sum modulo 10 cannot be detected nor corrected.

Proof. Let a_i and a_{i+1} be adjacent bit strings in a code word. Suppose during transmission, the bit strings are transposed and are such that they yield the same weighted sum modulo 10, that is; Suppose a_i is an even bit string and a_{i+1} is an odd bits string.
Then, $3a_i + a_{i+1} = b(mod 10)$ and $a_i + 3a_{i+1} = b(mod 10), 1 \leq i \leq 12$
Adding up the two equations yields $4a_i + 4a_{i+1} = 2b(mod 10)$. Thus there exists an integer n such that $4a_i + 4a_{i+1} - 2b = 10n, n \epsilon \mathbb{Z}$
Similarly, if a_i is an odd bit string and a_{i+1} is an even bits string, the same result will hold. Any transposition of such adjacent bit strings will go undetected. □

Proposition 3.1.7. In an ISBN-13 code, any single transposition (two bit strings transposed) of bit strings on a sent code word of the same parity (even or odd) cannot be detected nor corrected.

Proof. For this proof to be complete, it must be shown that it holds for both even and odd positions.
Without loss of generality, let a_i and a_j where $i = 2n, 1 \leq n \leq 6$ and $j = 2m, 1 \leq m \leq 6$ be the two bit strings in the sent code word a which are transposed to yield to code word b. Upon computation of the check digit, each bit string at an even position is multiplied by 3.

Clearly, $3a_i + 3a_j = 3a_j + 3a_i$ for all i and j since addition of integers is commutative. Thus upon computation of the check digit, both a and b yield the same result.

Similarly, without loss of generality suppose a_l and a_h , where $l = 2n+1, 1 \leq n \leq 6$ and $j = 2m+1, 1 \leq m \leq 6$ are the two bit strings in the sent code word a which are transposed to yield to code word b. Upon computation of the check digit, each bit string at an odd digit position is multiplied by 1.

Clearly, $a_l + a_h = a_h + a_l$ for all l and h. Thus upon computation of the check digit, both a and b yield the same result. In both cases, the error is not detected hence cannot be corrected. □

Example 3.1.1. Consider the sent code word 9780198538035 and 9087198538035 as the received code word. As shown earlier in section 1.1.5, 9780198538035 is a valid code word. Consequently, 9087198538035 will yield the same results since it is just a transposition of the second and fourth bit strings which are even bit strings. The following two propositions generalize the conditions for a double error on odd positions to be undetected.

Proposition 3.1.8. In an ISBN-13 code, any double silent errors on odd positions of a sent code word that yields same weighted sum modulo 10 with the ones on the received code word cannot be detected nor corrected.

Proof. For an error to be corrected, it must be first detected. Thus it is only needed to prove that the errors cannot be detected and consequently cannot be corrected. Suppose two silent errors occur on two odd positions of the code word $u = a_1a_2.\ldots.a_{11}a_{12}a_{13}$ namely a_i and a_j where $1 \leq i \leq 11, 1 \leq j \leq 11, a_i \neq a_j$ to yield to a code word $v = a_1a_2.\ldots.a_{11}a_{12}a_{13}$ which differ with u in only the two odd positions such that a_i is replaced by b_i and a_j is replaced by b_j.

Without loss of generality suppose $i = 1$ and $j = 3$. This choice can generalize the other odd positions since upon calculation of the check digit, the bit stings at the odd positions are each multiplied by 1 thus the digit position does not matter provided it is in an odd position.

$a_{13} = 10–(a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

and similarly

$a_{13} = 10–(b_1 + 3a_2 + b_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

Suppose $a_1 + a_3 \equiv b(mod 10)$ and $b_1 + b_3 \equiv b(mod 10), 1 \leq b \leq 9$

Then we have that,

$$(a_1 + a_3) - (b_1 + b_3) \equiv (b - b)(mod 10)$$

or

$$(a_1 + a_3) \equiv (b_1 + b_3)(mod 10)$$

This means that any such error on bit strings satisfying this equation will go undetected and hence uncorrected. □

Corollary 3.1.1. The lower limit of the number of bit strings that can be in error in an ISBN-13 codeword to yield to a silent error is 2.

Proof. Since the number of bit strings cannot be negative, then we only need to show that it cannot be zero or one. Suppose no bit string is in error in a valid code word. Thus the code word remains the same and similarly no silent error. If one bit string is in error, the check digit evaluation method would detect the error thus no silent error. Hence the lower limit on the number of bit strings that can be in error to yield to a silent error is 2.

This proof leads to the need of determining the upper limit on the maximum number of errors on a code word that can occur to yield to a silent error. This is established later in corollary 3.1.4 □

Corollary 3.1.2. In an ISBN-13 codeword, if the following values on bit strings are replaced with the ones indicated with an "or", then a silent double error occurs where $1 \leq i \leq 12, 1 \leq j \leq 12$.

$a_i = 1$, $a_j = 9$ or $a_i = 2$, $a_j = 8$ or $a_i = 3$, $a_j = 7$ or $a_i = 4$, $a_j = 6$ or $a_i = 5$, $a_j = 5$

$a_i = 2$, $a_j = 9$ or $a_i = 3$, $a_j = 8$ or $a_i = 4$, $a_j = 7$ or $a_i = 5$, $a_j = 6$

$a_i = 3$, $a_j = 9$ or $a_i = 4$, $a_j = 8$ or $a_i = 5$, $a_j = 7$ or $a_i = 6$, $a_j = 6$

$a_i = 4$, $a_j = 9$ or $a_i = 5$, $a_j = 8$ or $a_i = 6$, $a_j = 7$

$a_i = 5$, $a_j = 9$or $a_i = 6$, $a_j = 8$ or $a_i = 7$, $a_j = 7$

$a_i = 6$, $a_j = 9$ or $a_i = 7$, $a_j = 8$

$a_i = 8$, $a_j = 8$ or $a_i = 9$, $a_j = 7$

Proof. This is as a consequence of the proposition 3.1.10 above since they leave the same weighted sum modulo ten. □

Proposition 3.1.9. In an ISBN-13 code, any double silent errors on even positions on a sent code word that yields same weighted sum modulo 10 cannot be detected nor corrected.

Proof. Suppose two silent errors occur on two even positions of the code word $u = a_1a_2 \ldots a_{11}a_{12}a_{13}$ namely a_i and a_j, where $2 \leq i \leq 12, 2 \leq j \leq 12$, $a_i \neq a_j$, to yield to a code word $v = a_1a_2 \ldots a_{11}a_{12}a_{13}$ which differ with u in only the two even positions such that a_i is replaced by b_j and a_j is replaced by b_j.

Without loss of generality, suppose $i = 2$ and $j = 4$. This choice can generalize for the other even positions since upon calculation of the check digit; the bit stings at the even positions are each multiplied by 3 thus the digit position does not matter provided it is in an even position.

$a_{13} = 10–(a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

$a_{13} = 10–(a_1 + 3b_2 + a_3 + 3b_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

Suppose, $3a_2 + 3a_4 \equiv b(mod10)$ and $3b_2 + 3b_4 \equiv b(mod10), 1 \leq b \leq 9$ that then yields the same sum modulo ten.

Then $(3a_2 + 3a_4)–b = 10k_1,$ where $k_1 \in \mathbb{Z}$ and $(3b_2 + 3b_4)–b = 10k_2$ where $k_2 \in \mathbb{Z}$
Subtracting yields $3(a_2 + a_4)–3(b_2 + b_4) = 10(k_1–k_2), (k_1–k_2) \in \mathbb{Z}$ or

$$3(a_2 + a_4) \equiv 3(b_2 + b_4)(mod10)$$

Hence,

$$(a_2 + a_4) \equiv (b_2 + b_4)(mod10)$$

Any interchange of a_i with a_j satisfying this equation will be an error which would go undetected. □

Proposition 3.1.10. In an ISBN-13 code, any multiple silent errors on even positions of a sent code word that yields same weighted sum modulo 10 cannot be detected nor corrected.

Proof. Suppose there are n silent errors on n even positions of the code word $u = a_1a_2 \ldots a_{11}a_{12}a_{13}$ to yield to a code word $v = b_1b_2 \ldots b_{11}b_{12}b_{13}$ which differ with u in n bit strings such that each even position a_i is replaced by b_i for $1 \leq 2i \leq 12$. Upon calculation of the check digit, the even positions are each multiplied by 3. Suppose $3\sum_i^{12} a_{2i} \equiv c(mod10)$ for $1 \leq i \leq 6$ and $3\sum_{2i}^{12} b_i \equiv c(mod10)$ for $1 \leq i \leq 6$ (that is, they yield the same weighted sum modulo ten).
Then $(3\sum_i^{12} a_{2i}) - c = 10k_1, k_1 \in \mathbb{Z}$ and $(3\sum_i^{12} b_{2i}) - c = 10k_2, k_2 \in \mathbb{Z}$.
Subtracting yields

$$3\sum_i^{12} a_{2i} - 3\sum_i^{12} b_{2i} = 10(k_1 - k_2)$$

thus

$$3\sum_i^{12} a_{2i} \equiv 3\sum_i^{12} b_{2i}(mod10)$$

Any interchange of a_i with b_i, for $1 \leq i \leq 6$, satisfying this equation will be an error which would go undetected. □

Example 3.1.2. Suppose the code word $a = 9780198538035$ is sent and suppose three errors occur on the second, fourth and sixth bit strings to yield a code word received as $b = 9881178538035$. As discussed earlier in 1.1.5, 9780198538035 is a valid ISBN-13 code. Considering 9881178538035

$$\begin{aligned} s &= 9\times1+8\times3+8\times1+1\times3+1\times1+7\times3 \\ &\quad +8\times1+5\times3+3\times1+8\times3+0\times1+3\times3 \\ &= 9+24+8+3+1+21+8+15+3+24+0+9 \\ &= 125 \equiv 5(mod 10) \end{aligned}$$

But 10–5 $= 5 \equiv 5(mod10)$ thus our check digit is 5. Hence 9881178538035 is also a valid code word. This means that even though errors occurred during transmission, the system will not detect the error made!

Corollary 3.1.3. In an ISBN-13 code, any multiple silent errors on odd positions on a sent code word that yields same weighted sum modulo 10 cannot be detected nor corrected.

Proof. Suppose there are n silent errors on n odd positions of the code word $C_1 = a_1a_2 \ldots a_{11}a_{12}a_{13}$ to yield to a code word $C_2 = a_1a_2 \ldots a_{11}a_{12}a_{13}$which differ with C_1 in n bit strings such that each odd positions, a_i, in error is replaced by b_i for $1 \leq 2i+1 \leq 11$

Upon calculation of the check digit, the odd positions are each multiplied by 1. Suppose $\sum_i^{11} a_{2i+1} \equiv c(mod10)$ and $\sum_i^{11} b_{2i+1} \equiv c(mod10)$ where $0 \leq i \leq 5$, that is,

they yield the same sum modulo ten.

Then $(\sum_i^{11} a_{2i+1}) - c = 10k_1, k_1 \in \mathbb{Z}$ and $(\sum_i^{11} b_{2i+1}) - c = 10k_2, k_2 \in \mathbb{Z}$.

Subtracting yields

$$\sum_{i}^{11} a_{2i+1} - \sum_{i}^{11} b_{2i+1} = 10(k_1 - k_2)$$

Thus

$$\sum_{i}^{11} a_{2i+1} \equiv \sum_{i}^{11} b_{2i+1} (mod 10)$$

Any interchange of a_i with b_i, for $0 \leq i \leq 5$ satisfying this equation will be an error which would go undetected. $\square$

Example 3.1.3. Consider the sent code word 9780198538035, received code word 9740598538035.

As discussed earlier in section 1.1.5, 9780198538035 is a valid ISBN-13 code. Considering 9740598538035

$$\begin{aligned} s &= 9 \times 1 + 7 \times 3 + 4 \times 1 + 0 \times 3 + 5 \times 1 + 9 \times 3 \\ &\quad +8\times1 + 5\times3 + 3\times1 + 8\times3 + 0\times1 + 3\times3 \\ &\equiv 5(mod 10) \end{aligned}$$

But 10–5 $= 5 \equiv 5(mod 10)$ Thus our check digit is 5. Hence 9740598538035 is also a valid code word. This means that even though errors occurred during transmission, the system will not detect the errors!

Corollary 3.1.4. The upper limit on the maximum number of errors on a code word that can occur to yield to a silent error in ISBN-13 code is twelve.

Proof. Since the check digit is computed from the other digits and any multiple silent errors on a sent code word that yields the same weighted sum modulo 10 cannot be detected as in corollary 3.1.3, the upper limit is thus twelve. $\square$

Proposition 3.1.11. The ISBN-13 detects any single error on any even digit position.

Proof. By contradiction, suppose it cannot. This means that there exists a codeword u and a received codeword v such that they differ on one even digit position say i where a_i and b_i are the i^{th} digit position respectively which shows a single error.
Since i is even, then $i = 2n, n \in \mathbb{Z}, 1 \leq n \leq 6$. The check digit is not an even position thus it is not in error and hence the check digits for the codewords are similar.

$$a_{13} \;=\; 10\text{–}(a_1 + 3a_2 + ... + 3a_i + a_{i+1} + ... + 3a_{12})(mod 10), \quad 2 \leq i \leq 12$$

and

$$a_{13} \;=\; 10\text{–}(a_1 + 3a_2 + ... + 3b_i + a_{i+1} + ... + 3a_{12})(mod 10), \quad 2 \leq i \leq 12$$

Suppose $3a_i \equiv k(mod 10)$ and $3b_i \equiv w(mod 10)$
Then $a_{13} \;=\; 10\text{–}(a_1 + 3a_2 + ... + k + a_{i+1} + ... + 3a_{12})(mod 10), \quad 2 \leq i \leq 12$
and $a_{13} \;=\; 10\text{–}(a_1 + 3a_2 + ... + w + a_{i+1} + ... + 3a_{12})(mod 10), \quad 2 \leq i \leq 12$
In $\mathbb{Z}_{10}, k = w$, since the code word is only made of digits between 0-9, there cannot be two different numbers between 0 and 9 which yield the same remainder modulo ten. Thus, any single error on an even digit position will be detected. □

Proposition 3.1.12. The ISBN-13 code detects any single error on any odd digit position.

Proof. By contradiction, suppose it cannot.
This means that there exists two code words x and y which differ on one odd digit position say i. This position cannot be the check digit since the check digit is unique to a codeword.
Since i is odd, then $i = 2n + 1, n \in \mathbb{Z}, 0 \leq n \leq 6$

Let a_i and b_i be bit strings on u and v respectively which differ.

$$a_{13} \quad = \quad 10–(a_1 + 3a_2 + ... + a_i + 3a_{i+1} + ... + 3a_{12})(mod 10), \quad 1 \leq i \leq 12$$

and

$$a_{13} \quad = \quad 10–(a_1 + 3a_2 + ... + b_i + 3a_{i+1} + ... + 3a_{12})(mod 10), \quad 1 \leq i \leq 12$$

In $\mathbb{Z}_{10}, a_i = b_i$, since the code word is only made of digits between 0 and 9, there cannot be two different numbers between 0 and 9 which yield the same remainder modulo ten. Thus, any single error on an odd digit position will be detected. □

Corollary 3.1.5. The ISBN-13 code can correct any single error on any digit position (even or odd).

Proof. Suppose an error has been detected on any digit position, a_i. To correct it, a digit is chosen such that the parity check equation holds. Since computation is done modulo 10 and the code words are only made of bit strings between 0 and 9, there cannot be two digits between 0 and 9 satisfying the above equation. Thus the error detected is corrected. □

Theorem 3.1.1. The ISBN-13 code does not detect transposition errors on even bit strings.

Proof. In ISBN-13 as computation of the check digit is done such that the parity check equation holds. That is,
$a_{13} = 10–(a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12})(mod 10)$

Since even bit strings are each multiplied by 3 then summed up modulo ten, if a single transposition error occurs such that an even bit string is transposed with another even bit string in the same code word, the summation will not change and so the computation of the check digit will not be affected. □

Corollary 3.1.6. The ISBN-13 code does not detect any transposition error on odd bit strings.

Proof. The proof is as a consequence of Theorem 3.1.1 above. □

Theorem 3.1.2. The ISBN-13 code does not detect all double transposition errors.

Proof. Suppose a double transposition error occurs on a code word u to yield a code word v such that for any two bit strings at distinct digit positions a_x and a_y, the bit string at digit position a_x is transposed with the one at a_w whereas the bit string at digit position a_y is transposed with the one at a_p, in the same code word, where $x \neq w$ and $y \neq p$ and that $1 \leq x, y, w, p \leq 12$. If $x = w$, then there is no transposition! If $y = p$, then there is no transposition! Since the transposition occurs on the same code word, the received code word differs with the sent code word in four bit strings. Therefore the minimum distance $d(u, v) = 4$. Upon computation of the check digit for u and v, the working is based on the parity of each of the bit string transposed so the check digit for v will differ with that of u if the transposed digit positions differ in their parity respectively. Otherwise if the digits transposed are of the same parity, the error will go undetected as shown in theorem 3.1.1 and Corollary 3.1.6 above.
The choices can be found by drawing a tree diagram involving the two parities to occupy the four positions. This yields to 2^4 options.

Table 3.1: Double transposition error in ISBN-13

	$a_x$transposed to a_w		$a_y$transposed to a_p	
cases	a_x	a_w	a_y	a_p
1	Odd	Even	Even	Even
2	Odd	Even	Even	Odd
3	Odd	Even	Odd	Even
4	Odd	Even	Odd	Odd
5	Odd	Odd	Even	Even
6	Odd	Odd	Even	Odd
7	Odd	Odd	Odd	Even
8	Odd	Odd	Odd	Odd
9	Even	Even	Odd	Even
10	Even	Even	Even	Odd
11	Even	Even	Odd	Odd
12	Even	Even	Even	Even
13	Even	Odd	Even	Even
14	Even	Odd	Even	Odd
15	Even	Odd	Odd	Even
16	Even	Odd	Odd	Odd

As far as parity of the transposed bit strings is concerned, Cases 5, 8, 11 and 12 represent multiple transpositions of bit strings with same parity and as discussed earlier Theorem 3.1.1 and Corollary 3.1.6, these errors cannot be detected.

In Cases 1, 4, 6, 7, 9, 10, 13 and 16, only one parity position is transposed. This represents a single transposition of bit strings with different parity. The error will not be detected if at these positions, the two weighted sums in the two codes are the same modulo 10 as discussed earlier in corollary 3.1.2. If not, then the error will be detected as follows:

Considering Case 1: a_x (odd) transposed to a_w (even) whereas a_y (even) transposed to a_p (even)

For the sent code word: $a_x + 3a_w + 3a_y + 3a_p \equiv k(mod 10), k \in \mathbb{Z}$

Received code word: $3a_x + a_w + 3a_y + 3a_p \equiv n(mod 10), n \in \mathbb{Z}$

Since a_y and a_p have the same parity, they are both multiplied by 3 thus the overall sum is the same irrespective of the transposition.

The difference in the two sums (sent and received code word) is therefore between $a_x + 3a_w$ and $3a_x + a_w$

Suppose $a_x + 3a_w \equiv h(mod 10)$ and $3a_x + a_w \equiv y(mod 10)$

This is the same as $(a_x + 3a_w) - h = 10c$ and $(3a_x + a_w) - y = 10d$ for some $c, d \in \mathbb{Z}$

Subtracting yields

$$(2a_x – 2a_w) - (y - h) = 10(d - c)$$

and

$$(2a_x – 2a_w) \equiv (y - h)(mod 10)$$

Suppose $y = h$, then $y – h = 0$ thus, $2a_x – 2a_w \equiv 0(mod 10)$, since computation is done in $\mathbb{Z}_{10}$ this can only happen when $a_i = a_w$ contradicting the fact that a transposition took place, hence $y \neq h$. Thus, $a_x + 3a_w \neq 3a_x + a_w$ and therefore $k \neq n$ so the errors are detected.

This shows that since the parity of a_x differs with that of a_w, if u and v have the same check digit, then an error(s) must have occurred.

Cases 4, 6, 7, 9, 10, 13 and 16 follow a similar argument since in each, one of the transpositions occurs on digit positions with the same parity.

Cases 2, 3, 14, and 15 represent a double transposition of bit strings with different parity of the digit position.

The error will not be detected if at these positions, the two weighted sums in the two codes are the same modulo ten as discussed earlier in corollary 3.1.2.

Otherwise, the error will be detected as follows: Consider case 14. a_x (even) is transposed to a_w(odd) whereas a_y(even) is transposed to a_p(odd).

Sent code word: $3a_x + a_w + 3a_y + a_p \equiv k(mod 10), k \in \mathbb{Z}$

Received code word: $a_x + 3a_w + a_y + 3a_p \equiv n(mod 10), n \in \mathbb{Z}$

If $k = n$, then it means the bit strings from the sent and received code words at these digit positions yield the same weighted sum modulo ten. This error cannot be detected as in corollary 3.1.2 above. If $k \neq n$, the errors are obviously detected. Cases 2, 3 and 15 follow a similar argument since they are double transposition of bit strings with different parity of the digit position. □

3.1.3 Number of Code words versus Error Detection and Correction

In this section, the effect of the capability to detect or correct error(s) on the dictionary size is discussed and analyzed.

Proposition 3.1.13. The ability to detect an error(s) in the ISBN-10 code does not affect (increase or reduce) its dictionary size.

Proof. As discussed earlier in section 2.2.1, the ISBN-10 code can detect single errors and double errors which come as a result of transposition of two digits. To detect the error(s), the parity check equation is computed. This condition is the general condition for a code word to be an ISBN-10 code word. Thus to detect an error(s), there is no extra condition imposed. With this initial condition, the code will still generate the expected code words which satisfy the condition. Any arbitrary code word which does not satisfy this condition will not be an ISBN-10 code word thus does not increase or reduce the code's intended dictionary. If a silent error occurs, it yields a different code word which is also a member of the code and similarly does not increase or reduce the dictionary. □

Proposition 3.1.14. The conditions for error correction given by equations 2.4 and 2.5 in section 2.2.3 above reduces the size of the dictionary of the ISBN-10.

Proof. By counter example, Consider the code word $u = 0198538030$

$$(\sum_{i=1}^{10} ia_i) = 187 \equiv 0 (mod 11)$$

Therefore u is an ISBN-10 code word.

But

$$(\sum_{i=1}^{10} a_i) = 37 \equiv 4(mod 11) \neq 0(mod 11)$$

Then u does not necessarily satisfy the equation 2.4 above. Thus if the code is to correct errors, v will not be contained in it since it does not satisfy the required conditions. This in turn reduces the number of code words in the code. □

Proposition 3.1.15. The ability to detect an error(s) in the ISBN-13 code does not affect (increase or reduce) its dictionary size.

Proof. As earlier discussed in section 1.1.5 , to detect the existence of error(s) in an ISBN-13 code word, the equation below is evaluated

$x_{13} = 10–(x_1+3x_2+x_3+3x_4+x_5+3x_6+x_7+3x_8+x_9+3x_{10}+x_{11}+3x_{12})(mod 10)$

This equation is the general condition for a code word to be an ISBN-13 code word. The code generates code words which satisfy the condition. An arbitrary code word $\boldsymbol{x}$ which does not satisfy this condition will not be an ISBN-13 code word thus its existence does not increase or reduce the dictionary. If a transposition or silent error occurs, it yields a different code word which is also a member of the code and similarly does not increase or reduce the dictionary. □

Proposition 3.1.16. The ability to correct a single error in the ISBN-10 code does not affect (increase or reduce) its dictionary size.

Proof: As earlier discussed in section 2.2.3, to correct a single error in the sent code word, a code word $a_1a_2 \ldots a_{10}$ has to satisfy the following:

$\sum_{i=1}^{10} a_i \equiv \sum_{i=1}^{10} ia_i \equiv 0(mod 11)$.

That is,

$$\sum_{i=1}^{10} a_i = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10} \equiv 0(mod 11)$$

and

$$\sum_{i=1}^{10} ia_i = a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 + 6a_6 + 7a_7 + 8a_8 + 9a_9 + 10a_{10} \equiv 0 (mod 11)$$

Thus

$$\begin{aligned} a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 & \\ +a_9 + a_{10} &= 11k, k \in \mathbb{Z} \end{aligned} \tag{3.1}$$

and

$$\begin{aligned} a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 + 6a_6 + 7a_7 + 8a_8 & \\ +9a_9 + 10a_{10} &= 11r, r \in \mathbb{Z} \end{aligned} \tag{3.2}$$

Equation 3.2 −Equation 3.1 yields

$$\begin{aligned} a_2 + 2a_3 + 3a_4 + 4a_5 + 5a_6 + 6a_7 + 7a_8 & \\ +8a_9 + 9a_{10} &= 11(r–k) \text{ where } (r - k) \in \mathbb{Z} \end{aligned} \tag{3.3}$$

This means that

$$a_2 + 2a_3 + 3a_4 + 4a_5 + 5a_6 + 6a_7 + 7a_8 + 8a_9 + 9a_{10} \equiv 0 (mod 11) \tag{3.4}$$

Equation 3.4 does not involve a_1 and hence a_1 can be chosen freely from digits 0 to 9. As discussed earlier, the check digit a_{10} does not permute; it is computed to satisfy the parity check equation. The other bit strings $a_2, a_3, \ldots, a_8$ can be chosen from $0 - 9$ such that equation 3.4 holds.

Solving equation 3.3 requires that the GCD of $2, 3, 4, 5, 6, 7$ and 8 divide $11(r-k)$. The GCD is 1 which will always divide $11(r-k)$ and thus the equation has integer solution. The bit strings $a_2, a_3, \ldots, a_8$ can be chosen from 0 to 9 yielding to 10^7 choices. In total we have 10^8 choices which show that the dictionary is not affected.

Theorem 3.1.3. The ability to correct transpose errors in the ISBN-10 code reduces its dictionary size.

Proof: As earlier discussed in section 2.2.3, Raymond showed that, to correct double errors the sent code word $a_1, a_2, \ldots, a_{10}$, has to satisfy the following:

$$\textstyle\sum_{i=1}^{10} a_i \equiv \sum_{i=1}^{10} ia_i \equiv \sum_{i=1}^{10} i^2a_i \equiv \sum_{i=1}^{10} i^3a_i \equiv 0(mod 11)$$

But

$$\sum_{i=1}^{10} a_i = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10} \equiv 0(mod 11)$$

whereas

$$\sum_{i=1}^{10} ia_i = a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 + 6a_6 + 7a_7 + 8a_8 + 9a_9 + 10a_{10} \equiv 0(mod 11)$$

and

$$\begin{aligned}\sum_{i=1}^{10} i^2a_i = a_1 &+ 2^2a_2 + 3^2a_3 + 4^2a_4 + 5^2a_5 + 6^2a_6 \\ &+ 7^2a_7 + 8^2a_8 + 9^2a_9 + 10^2a_{10} \equiv 0(mod 11)\end{aligned}$$

and

$$\begin{aligned}\sum_{i=1}^{10} i^3a_i = a_1 &+ 2^3a_2 + 3^3a_3 + 4^3a_4 + 5^3a_5 + 6^3a_6 \\ &+ 7^3a_7 + 8^3a_8 + 9^3a_9 + 10^3a_{10} \equiv 0(mod 11)\end{aligned}$$

Thus

$$\begin{aligned}a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 \\ +a_9 + a_{10} &= 11r, r \in \mathbb{Z}\end{aligned} \tag{3.5}$$

$$\begin{aligned}a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 + 6a_6 + 7a_7 + 8a_8 \\ +9a_9 + 10a_{10} &= 11p, p \in \mathbb{Z}\end{aligned} \tag{3.6}$$

$$\begin{aligned}a_1 + 4a_2 + 9a_3 + 16a_4 + 25a_5 + 36a_6 + 49a_7 + 64a_8 \\ +81a_9 + 100a_{10} &= 11q, q \in \mathbb{Z}\end{aligned} \tag{3.7}$$

$$\begin{aligned}a_1 + 8a_2 + 27a_3 + 64a_4 + 125a_5 + 216a_6 + 343a_7 \\ +572a_8 + 729a_9 + 1000a_{10} &= 11k, k \in \mathbb{Z}\end{aligned} \tag{3.8}$$

Equation 3.6 - equation 3.5 yields

$$\begin{aligned}a_2 + 2a_3 + 3a_4 + 4a_5 + 5a_6 + 6a_7 + 7a_8 + 8a_9 \\ +9a_{10} &= 11(p–r) \text{ where } (p-r)\epsilon \in Z\end{aligned} \tag{3.9}$$

Equation 3.8 - equation 3.7 yields

$$\begin{aligned} 4a_2 + 18a_3 + 48a_4 + 100a_5 + 180a_6 + 294a_7 \quad + 508a_8 \quad + 648a_9 & \\ +900a_{10} \quad = \quad & 11(k – q) \text{ where } (k – q) \in Z \end{aligned} \tag{3.10}$$

Then, equation 3.10 – [4 × (Equation 3.9)] yields

$$\begin{aligned} 10a_3 + 36a_4 + 84a_5 + 160a_6 + 270a_7 \quad + 480a_8 \quad + 616a_9 & \\ +864a_{10} \quad = \quad & 11(k - 4p - q + 4r) \\ \text{where} \quad & (k - 4p - q + 4r) \in Z \end{aligned} \tag{3.11}$$

This condition does not involve a_1 and a_2 hence a_1 and a_2 can be freely chosen from 0 to 9.

These are 10^2 choices. The check digit a_{10} does not permute as shown earlier in proposition 3.1.5.

The other bit strings $a_3, a_4, \ldots, a_8$ can be chosen from $0 - 9$ such that equation 3.11 holds. The GCD of $10, 36, 84, 160, 270, 480, 616$ is 2.

The equation therefore has integer solution if

$$2 \mid 11(k - 4p - q + 4r)$$

That is, if $11(k - 4p - q + 4r)$ is even.

If $(k - 4p - q + 4r)$ is not even, it means that the equation cannot be solved so a double error on any code word that does not satisfy this condition will not be corrected.

But $11 = 2(5) + 1$ which yield to:

$$\begin{aligned} 11(k - p - q + r) &= (2(5) + 1)(k - 4p - q + 4r) \\ &= (2(5)(k - 4p - q + 4r)) + (k - 4p - q + 4r) \end{aligned} \tag{3.12}$$

which is even if $(k-4p-q+4r)$ is even since $2(5)(k-4p-q+4r)$ is always even. Thus bit strings $a_3, a_4, \ldots, a_8$ can be chosen from 0–9 such that $(k-4p-q+4r)$ is even to ensure equation 3.11 above has integer solutions.

But $k-4p-q+4r = k-q-4(p-r)$ and $4(p-r)$ is always even since $4(p-r) = 2(2(p-r))$, implying that $((k-q)-4(p-r))$ will be even if $(k-q)$ is even since the difference of any two even numbers is always even.

$(k-q)$ cannot be odd otherwise $k-q-4(p-r)$ would end up being odd since the difference of an odd integer and an even one is odd.

But generally, $(k-q)$ is not always even. It may either be odd (when either k or q is odd and the other is even) or even (when both are even or both odd). If it is even, then the equation 3.11 will have integral solutions and hence the error can be corrected thus the dictionary is not affected.

If $(k-q)$ is odd, the equation 3.11 would not have an integral solution and so the double error won't be corrected.

This affects the choice of $a_3, a_4, \ldots, a_8$ to freely permute up to 10^6 choices since as seen, not all the choices will satisfy the equation. This in turn reduces the dictionary size as far as the double error correction is concerned.

There is thus need to generate a code that combines the strength of the ISBN-10, ISBN-13 and cyclic ISBN-10 codes and at the same time attempt to resolve their limitations. The code should have a big dictionary and at the same time improve on error detection and error correction capabilities.

CHAPTER FOUR

THE ISBN-16 CODE

This chapter discusses the results extensively by discussing the development and generation of ISBN-16 and analysing its properties. The strength and weaknesses of the code are also analyzed with respect to the dictionary, error detection and correction capabilities.

4.1 Development and Generation of ISBN-16 Code

In this section the development and generation of the designed ISBN-16 code; an improvement to the conventional ISBN and the Cyclic ISBN-10 codes are discussed. The ISBN-16 code is aimed at using the strengths in the ISBN-10, ISBN-13 and the Cyclic ISBN-10 codes to create an improved code as far as the following properties are concerned:

- Error detection
- Error correction
- Dictionary size

The ISBN-16 code is made up of all code words; length sixteen (16) bit strings, consisting of any of the numbers $0, 1, 2, \ldots, 9, A, B, C, D, E, F, G$ where A, B, C, D, E, F, G represent the values indicated in the table 4.1.

Table 4.1: ISBN-16 letter representation

Letter	A	B	C	D	E	F	G
Value	10	11	12	13	14	15	16

The choice of the letters is done to avoid confusion that may occur when the two digit numbers namely $10, 11, 12, 13, 14, 15$ and 16 are used. For example, if...121... represent some digits in a code word, it is not easy to know if the digits are $\ldots 1$ followed by 2 then $1\ldots$ or it is $\ldots 12$ followed by a $1\ldots$ The use of the letters therefore avoids this confusion.

Suppose $u = a_1a_2a_3 \ldots a_{15}a_{16}$ is an ISBN-16 code word; it must satisfy the condition

$$\sum_{i=1}^{16} ia_i \equiv 0(mod17) \text{ for } i = 1, 2, ..., 16 \tag{4.1}$$

Equation 4.1 is called a parity-check equation. All the code words in the code are of the form:

$$a_1a_2 - a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12}a_{13}a_{14}a_{15} - a_{16}$$

4.1.1 Generation of an ISBN-16 Code word

1. The first block of bit strings (made up of two bit strings) is chosen randomly and represent the country where the book is published. It is estimated that there are about 196 countries in the world, Atlas-World (2012). Since two bit strings permute (out of 16 choices), there are $16^2 = 256$ possible permutations. This means that all the estimated countries will be represented. For example, 01 to represent Kenya.

2. The second block of digits (made up of thirteen bit strings) written in an increasing or decreasing order or constant flow; it represents the number assigned to the book by the publishing company. The flow depends on the order of flow of the first block of bit strings. For example 01-3456789ABCDE-

3. The check bit strings is chosen such that Equation 4.1 above is satisfied. For example 01-3456789ABCDE-G

4. Repetition of numbers is allowed.

5. The digits do not have to follow one another consecutively. That is, if you start with a 1 for an increasing order then 2 is not necessarily the next digit; it can be any of the other numbers including 1 itself since repetition is allowed.

6. Changing the order of flow of bit strings from increasing to decreasing and vice versa is done by inserting a zero (0). The new flow may start with zero or any other digit. Zero (0) does not have to necessarily change the order of flow it may be used as a part of the code word itself. That is, a zero may be used and the order of flow does not change. Here the zero acts as a neutral element. This means that the order of flow of digits does not necessarily change once a zero is put. It is at the discretion of the code word developer to decide whether to change the order or not.

7. The check digit does not necessarily have to obey the order of flow of the bit strings. As seen earlier in equation 4.1, the check digit obeys the given condition. It is therefore computed, not chosen randomly.

8. $(\mathbb{Z}_{17}^{*}, \times, +)$ is a field.

4.1.2 Calculation of inverses in ISBN-16 Code

Under operation $+$

If k and m are elements in the field $\mathbb{Z}_{17}$, k is said to be the additive inverse of m (denoted by $-m$) if $m + k \equiv 0(mod17)$. Since 0 is the additive identity, the following result hold.

Table 4.2: ISBN-16 Inverse under addition

k	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G
$-k$	G	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1

Under operation $\times$

If k and m are non zero elements in the field $\mathbb{Z}_{17}$, k is said to be the multiplicative inverse of m (denoted by m^{-1}) if $k \times m \equiv 1(mod 17)$. Since 1 is the multiplicative identity the following result hold.

Table 4.3: ISBN-16 Inverse under multiplication

k	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G
k^{-1}	1	9	6	D	7	3	5	F	2	C	E	A	4	B	8	G

4.1.3 Calculation of the check digit in ISBN-16 Code

Let $u = a_1 a_2 \ldots . a_{16}$ be a code word. To calculate the check digit, a_{16};
Evaluate $P = \sum_{i=1}^{15} i a_i \equiv 0(mod 17)$ for $i = 1, 2, \ldots, 15$
This yield to $P + 16a_{16} \equiv 0(mod 17)$ or $16a_{16} \equiv (-P)(mod 17)$
Therefore

$$a_{16} \equiv 16^{-1}(-P)(mod 17) \tag{4.2}$$

Thus, $17|a_{16} - (16^{-1})(-P)$ but since 16^{-1} is 16 in $\mathbb{Z}_{17}$, then $17|a_{16} + 16P$. Considering $P \in \mathbb{Z}_{17}$, $a_{16} = P$ to yield $17|17P$.

Example 4.1.1. Find the check digit denoted by $*$ in the code word $111111111111111*$

$P = \sum_{i=1}^{15} ia_i = 120 \equiv 1(mod17)$

But $a_{16} = 16^{-1}(-P)(mod17)$

This yield to

$$a_{16} = 16^{-1}(-1)(mod17) \quad = 16{\times}16(mod17) \quad \equiv 1(mod17)$$

The check digit is therefore 1 and the code word is 1111111111111111.

Other examples of ISBN-16 code words

$0123456789ABCDEG$	$AAAAAABBBBBCCCCA$	$810567889990000G$
$6890D7422220111D$	$10D0707610508003$	5555566666777775
$987654321012345D$	$15689A0B5321023B$	$102030405060708F$

4.2 Error Detection in ISBN-16 Code

Let $u = a_1a_2 \ldots .a_{16}$ be a code word and suppose an error(s) exist in the code word.

To detect the error(s) the following steps are followed.

1. Count the number of bit strings in the code word; they must be 16. If not, an error(s) exists

2. Check if the first block of bit strings indicates the correct country where the book was published. If not then we have error(s). Since the countries are many, it is very tedious to keep checking to the respective code for each country hence unless one is sure of the code this step may be avoided. In this research, the assignment of codes to each country has not been done.

3. Check the consistency flow of the bit strings in the code word. The flow may be either increasing or decreasing or constant. If any bit string(s) breaks the flow then we have error(s).

4. Work out for Equation 4.1. Even if steps $1, 2$ and 3 are correct, step 4 must be computed. It is the basis of the parity check.

5. If the steps $1, 2, 3$ and 4 are correct, then the code word is a valid ISBN-16 code word. If any of the steps is not correct, then error(s) exist.

Consider the code word $810567889290000G$. In this code word the flow of bit strings is inconsistent at tenth bit string. By observation, only 9 or 0 can occupy this bit string without breaking the order of flow. Replacing with 9 in this position we find $810567889990000G$. Working out the parity check equation yields to

$$\sum_{i=1}^{16} ia_i \equiv 0(mod 17)$$

It can be shown easily that 0 cannot occupy this position since the parity check equation will not hold.
Hence $810567889990000G$ is the correct code word.

4.3 Error Correction in ISBN-16 Code

To correct an error(s), it must first be detected. This means that the bit string(s) in the error have been identified and (or) Equation 4.1 does not hold.

4.3.1 Errors Correction on first block of bit strings

If an error exists on the first block, that is, the first block of bit strings differs from the country of publication (upon assigning each country with a specific code), then to correct the error correction is done as follows:

1. Identification of the correct country code where the book was published is done.

2. Replacement of the faulty digit with the correct one is done.

3. Identification of any other error(s) is done by using the other error(s) detection steps listed in section 4.2.

4.3.2 Check digit Error Correction

Suppose an error occurs on the check digit. To correct it, computation is done as in section 4.1.3 thus

$$16a_{16} + \sum_{i=1}^{15} ia_i = 0(mod 17)$$

Example 4.3.1. Consider the code word 5555566666777778.

The order of flow is correct. But,

$$\sum_{i=1}^{16} ia_i \equiv 14(mod 17)$$

Hence the code word is in error.

Working out the correct check digit, $a_{16} \equiv 16^{-1}(-P)(mod 17)$

But

$P = \sum_{i=1}^{15} ia_i \equiv 5(mod 17)$

So

$$a_{16} \equiv 16^{-1}(-5) \quad = 16(12) \quad \equiv 5(mod 17)$$

Or simply $a_{16} = P = 5$

The correct code word is thus 5555566666777775

4.3.3 Single Error correction

A single error occurs mainly due to typing mistakes or due to smudge. Suppose the order of flow is incorrect at one bit string, then a single error is said to exist if upon correcting the faulty bit string, the parity check equation holds, otherwise multiple errors exist.

Once the error has been detected and error position noted, error correction is done as follows.

1. The order of flow of bit strings (increasing, decreasing or constant) just before and after the faulty bit string is checked.
2. A number that obeys the order flow is chosen (they may be many).
3. Equation 4.1 is computed for each of the possible codewords. Once a correct one is found, replacement is done with the faulty one.

Example 4.3.2. Consider the code word $235AA053710GDBBD$.

We have an increasing flow of digits up to the 5th bit string, then at position six we have a zero followed by a decreasing flow but at the 9th position, order of flow is broken (error exists). The only digits that can obey the order of flow are $3, 2, 1$ and 0.

Replacing 3 with the faulty digit yields $235AA053310GDBBD$; working out the parity check yields

$$\sum_{i=1}^{16} ia_i \equiv 10 \neq 0(mod 17)$$

Hence $235AA053310GDBBD$ is not the correct bit string.

Replacing 2 with the faulty digit yields $235AA053210GDBBD$; working out the parity check yields

$$\sum_{i=1}^{16} ia_i \equiv 0(mod 17)$$

Hence $235AA053210GDBBD$ is the correct code word. Once the correct bit string is found, there is no need to work out for other "possible" bit strings. Since inverses in $(\mathbb{Z}_{17}^{*}, \times, +)$ are unique. Thus no two bit strings can be replaced for one faulty position to satisfy Equation 4.1.

4.3.4 Double error correction

Double errors occur when two bit strings are in error.

4.3.4.1 Any two bit strings excluding the check digit

This error is noticed easily since it simply occurs when any two bit strings (excluding the check digits) break the order of flow of digits. To correct these errors, the order of flow on the preceding and consequent digits is established (either increasing, decreasing or constant). A pair is chosen to replace the faulty digits such that the flow of digits is correct and Equation 4.1 holds.

Example 4.3.3. Consider the received code word $810567184990000G$

At position seven, the bit string 1 is between a bit string 7 and 8, breaking the order of flow leading to an error. Similarly at position nine, the bit string 4 is between a bit string 8 and 9, breaking the order of flow leading to an error.
At position seven only digits $0, 7$ or 8 can occupy this position without breaking the order of flow whereas at position nine, only $0, 8$ or 9 can occupy this position without breaking the flow.
Any pair that satisfy the parity check equation yields the correct code word. Replacing the seventh and the ninth bit strings with 7 and 8 respectively yielding $810567889990000G$ which obeys the parity check equation hence the sent correct code word. This choice is done through trial and error.
Due to uniqueness of the check digit which is as a result of unique inverses in $\mathbb{Z}_{17}$, no other pair can satisfy the equation.

4.3.4.2 The check digit and any other single bit string error

This error occurs when one bit string and the check bit string are in error. The bit string breaks the order of flow and even when corrected, the equation 4.1 does not hold. To correct these errors, the order of flow for the preceding and consequent digit is established (either increasing, decreasing or constant). Replacement for the faulty bit string is done and the check digit is computed such that equation 4.1 holds. Since upon replacement, different digits may obey the order of flow, the error correction may yield many different but valid code words. This simply means that the original intended code word must be among the possible choices and to identify it, one may need to know the intended check digit on the sent code word, otherwise one may not get the intended codeword.

Example 4.3.4. Consider the received code word 8105678895900005

At position ten, the bit string 5 is in between two 9's hence breaking the order of flow leading to an error. Only 0 or 9 can occupy this position without breaking the flow of digits. But upon replacing them in the position, none of the resultant code words, 8105678890900005 or 8105678899900005, satisfy the equation 4.1 hence the check digit is also in error.
Considering 8105678890900005 and using the check digit computation method yields B as the correct check digit hence 810567889090000B is the corresponding correct code word.
Similarly considering 8105678899900005 and using the check digit computation method yields G as the correct check digit hence 810567889990000G is the corresponding correct code word. The code word sent could therefore be any of the two code words. No other possible code word could have been the sent one since no other code word can satisfy the flow of digits.

4.3.5 Triple Error Correction

Triple errors occur when three bit strings are in error. They may occur either on any three bit strings (that do not obey the order of flow) or any two bit string and the check bit string. The error correction is similar as in section 4.3.4. If the error is on any three bit strings, a triple is chosen such that it obeys the order of flow and satisfy's the parity check equation. If the error is on any two bit string and the check bit string, a pair is chosen and check digit worked out such that the order of flow is obeyed and the parity check equation obeyed. Just as in section 4.3.4.2, for errors involving the check digit, different digits may obey the order of flow. The error correction may yield many different but valid code words. This simply means that the original intended code word must be among the possible choices and to identify it, one may need to know intended check digit on the sent code word.

Example 4.3.5. Consider the received code word 5515566966772775

Bit string 1 at position three breaks the order of flow since from position one to seven we have an increasing flow. Similarly bit string 9 and 2 at positions eight and thirteen respectively break the order of flow. The possible bit strings that can fill these faulty (third, eighth and thirteenth) positions without breaking the order of flow are:

Table 4.4: Triple error correction

	Position Three	Position Eight	Position Thirteen
case 1	0	0	0
case 2	0	6	0
case 3	0	0	7
case 4	0	6	7
case 5	5	6	7
case 6	5	0	0
case 7	5	6	0
case 8	5	0	7

Considering the non faulty positions in the code word 55 _5566_6677_775, and taking the sum

$$\sum_{i=1}^{16} ia_i(mod17), \quad i \neq 3, i \neq 8 \text{ and } i \neq 13$$

yields

$$(5\times1)+(5\times2)+(5\times4)+(5\times5)+(6\times6)+(6\times7)+(6\times9)+(6\times10) \\ +(7\times11)+(7\times12)+(7\times14)+(7\times15)+(5\times16) \equiv 16(mod17)$$

To get the correct code word, the sum as in equation 4.1, the faulty digits should yield $1(mod17)$ so that the sum $16+1 \equiv 0(mod17)$. That is, $3a_3 + 8a_3 + 13a_{13} \equiv 1(mod17)$

Working out case by case yields the following

Case 1: $(0\times3)+(0\times8)+(0\times13) \equiv 0(mod17)$ thus incorrect

Case 2: $(0\times3)+(6\times8)+(0\times13) \equiv 14(mod17)$ thus incorrect

Case 3: $(0\times3)+(0\times8)+(7\times13) \equiv 6(mod17)$ thus incorrect

Case 4:$(0\times3)+(6\times8)+(7\times13) \equiv 3(mod17)$ thus incorrect

Case 5: $(5\times3)+(6\times8)+(7\times13) \equiv 1(mod17)$ thus correct

Case 6: $(5\times3)+(0\times8)+(0\times13) \equiv 15(mod17)$ thus incorrect

Case 7: $(5\times3)+(6\times8)+(0\times13) \equiv 12(mod17)$ thus incorrect

Case 8: $(5\times3)+(0\times8)+(7\times13) \equiv 4(mod17)$ thus incorrect

The correct code word is 5555566666777775. Each of the above cases would yield to valid codewords if one would replace the digits and work out their new check digits separately.

4.3.6 Other Multiple Errors Correction

Just as in double and tripple errors, errors occurring on more than 3 bit strings can also be corrected. After detecting the faulty digits positions, replacement is done on these positions such that the order of flow and the parity equation are obeyed.

Example 4.3.6. Let 117118116117111 be the received code word.

Checking the order of flow of bit strings, the third, sixth, ninth and twelveths bit strings are in error since each of them is in between two 1's breaking the order of flow.

Bit strings should be choosen to replace these faulty positions and satisfy the parity check equation. The table below gives all possible options that can replace the faulty position without breaking order of flow.

Table 4.5: Other mutiple error corrections

	Position Three	Position Six	Position Nine	Position Twelve
case 1	0	0	0	0
case 2	0	0	0	1
case 3	0	0	1	0
case 4	0	1	0	0
case 5	1	0	0	0
case 6	0	0	1	1
case 7	0	1	0	1
case 8	0	1	1	0
case 9	1	0	0	1
case 10	1	0	1	0
case 11	1	1	0	0
case 12	0	1	1	1
case 13	1	0	1	1
case 14	1	1	0	1
case 15	1	1	1	0
case 16	1	1	1	1

These sixteen cases will yield to a valid code word if one would work out their check digits separately.

For this example, the check digit is 1;

working out yields ,$\sum_{i=1}^{16} ia_i, i \neq 3, i \neq 6, i \neq 9$ and $i \neq 12$ yield $4(mod 17)$.

Therefore to get the correct bit strings, $3a_3 + 6a_6 + 9a_9 + 12a_{12} \equiv 13(mod 17)$ so that $13 + 4 \equiv 0(mod 17)$

Working out case by case yields case 16 as the correct one since

$$(1 \times 3) + (1 \times 6) + (1 \times 9) + (1 \times 12) \equiv 13(mod 17)$$

The correct code word is thus 1111111111111111.

4.4 Number of code words in the ISBN-16 Code

Proposition 4.4.1. *In an ISBN-16 code, the check bit string does not permute.*

Proof. Each ISBN-16 code word has a check digit which is computed to satisfy the equation 4.2. Since the inverses in $(\mathbb{Z}_{17}^{*}, \times, +)$ are unique, there cannot be any two or more code words with the first fifteen bit strings similar but have a different check digit hence the check digit is unique to each code and cannot permute freely. $\square$

Proposition 4.4.2. *The ISBN-16 code has a dictionary with an upper limit of* $15^{16} = 6,568,408,355,712,890,625$ *code words.*

Proof. Since each code word is 16 bit strings long and the check digit does not permute (as shown in proposition 4.4.1 above), only 15 bit strings can permute. The first bit string can be chosen among the 16 choices. Assuming no condition is imposed on order of flow of digits, the second digit also can be chosen among the 16 choices. Going on this way and making the same assumption leads to 15 bit strings being chosen freely among 16 choices with repetition allowed. This yields to 15^{16} permutations hence $15^{16} = 6,568,408,355,712,890,625$ code words. But since order of flow of bit strings must be obeyed, this total number of code words is thus an upper bound. This upper bound may not necessarily be achieved since consider a code whose first three bit string are $432\ldots$, the fourth digit can only be a 2 or 1 or 0. This means that it does not permute freely hence the maximum number of permutations will not be achieved. The upper limit is thus 15^{16} code words. $\square$

4.4.1 Error Detection and Correction in the ISBN-16 Code

Proposition 4.4.3. *The ISBN-16 code detects and corrects any single error.*

Proof. Let $u = a_1a_2a_3 \dots a_{15}a_{16}$ be an ISBN-16 code word.

Any error on any single bit string (either it breaks the flow of bit strings or not) in any valid code word will not obey the parity check equation due to the uniqueness of the inverses hence error is detected.

This means that in the ISBN-16 code, no single silent error can go undetected. That is, no single smudge, typing error, omission or insertion error can go undetected. If the single error is on the check digit, the correction is done as in section 4.3.2. If the single error is on the other bit strings and it breaks the order of flow of bit strings, the error correction is done as in section 4.3.3.

If the single error does not break the order of flow of bit strings, the parity check equation detects and corrects it as if it is always a check digit error. If the error was not on the check digit, the intended sent code word may not be achieved. To avoid this and to correct the intended existing error the following steps are followed:

Suppose the error is on bit string k, where $k = 1, 2, \dots, 15$, clearly $k \neq 16$ otherwise it would be a check digit error and is discussed in section 4.3.2). For a received vector $u = b_1b_2 \dots b_{16}$ and since an error exists, its weighted check sum

$$W = \sum_{i=1}^{16} ib_i(mod 17) \neq 0(mod 17)$$

The sum

$$W_{i\neq k} = \sum_{i=1, i\neq k}^{16} ib_i(mod 17)$$

is then computed.

Clearly, the summation $W_{i\neq k} \neq 0(mod 17)$ unless $b_k = 0$ in the sent code word before error occurred.

The bit string b_k and the error position k are chosen such that

$$W_{i\,\neq k} + kb_k \equiv 0(mod 17)$$

If k is known (by observing the order of flow), then the bit string

$$b_k \equiv (-(W_{i\neq k}))(k^{-1})(mod 17)$$

and error is easily corrected. If k is not known (the error does not break order of flow), then k and b_k are chosen from $1, \ldots, 16$ such that

$b_k = (-W_{i\neq k})(k^{-1})(mod 17)$.

Comparison is then made with the bit string in the received code word. If they differ, then the received b_k is the faulty bit string.

This is a trial and error method which may end up being very cumbersome but is accurate and achievable since 17 is a prime number and inverses are unique. $\square$

Example 4.4.1. Let the code word $0123456789ABCDEF$ be the sent code word and suppose a single error occur to yield the received code word $0123356789ABCDEF$

Clearly, the error does not break the order of flow in the received code word $W = \sum_{i=1}^{16} ib_i \ = 12(mod 17)$ which means that an errors exist.

Since there is no prior knowledge of the exact bit string that is in error, this error could just be interpreted as a check digit error which can be corrected as in section 4.3.2.

Assumming that the check digit is not in error then the k and b_k are chosen form $1, \ldots, 16$ such that

$$b_k = (-W_{i\neq k})(k^{-1})(mod 17)$$

and comparison is made with the one in the received code word. If they differ, then the received b_{k} is the faulty bit string.

When $k = 1, b_1 = (-12)(1^{-1}) \equiv 12(mod 17)$ just as in the received code word

when $k = 2, b_2 = (-15)(2^{-1}) = 2 \times 9 \equiv 1(mod 17)$ just as in the received code word when $k = 3, b_3 = (-11)(3^{-1}) = 6 \times 6 \equiv 2(mod 17)$ just as in the received code word

when $k = 4, b_4 = (-5)(4^{-1}) = 12 \times 13 \equiv 3(mod 17)$ just as in the received code word

when $k = 5, b_5 = (-14)(5^{-1}) = 3 \times 7 \equiv 4(mod 17)$ which is different from the 5^{th} bit string in the received code word.

Replacing 4 in the 5^{th} bit string and working out the parity check equation yields $0(mod 17)$ hence$0123456789ABCDEF$ is the correct code word.

Theorem 4.4.1. *The ISBN-16 code detects any single transposition errors.*

Proof. Suppose $u = (a_1 a_2 \ldots a_{16})$ is the sent code word and v the received code word and suppose a single transposition error occurred during transmission on some bit strings a_i and a_{i+k} for some $i = 1, 2, \ldots, 16$ and $k = 1, 2, \ldots, 15$ such that $v = (a_1, \ldots a_i, a_{i+1}, \ldots a_{i+k} \ldots a_{16})$.

If the transposition leads to the order of flow of digits being broken on one or both bit string positions, the error is detected by the observation method. If the order of flow of bit strings is not broken, the following cases arise: $\square$

Table 4.6: Single transposition error detection in ISBN-16

	Possibility	Effect on parity check equation after transposition	Transposition detection
Case 1	$a_i = a_{i+k} = 0$	$ia_i + (i+k)a_{i+k} = ia_{i+k} + (i+k)a_i = 0$	No Transposition
Case 2	$a_i = a_{i+k} \neq 0$	$ia_i + (i+k)a_{i+k} = ia_{i+k} + (i+k)a_i$	No Transposition
Case 3	$a_i < a_{i+k}$ (clearly if $a_{i+k} = 0$ then a_i does not exist for this case)	$ia_i + (i+k)a_{i+k} > (i+k)a_i + ia_{i+k}$ The products $[ia_i], [(i+k)a_{i+k}], [ia_{i+k}]$ and $[(i+k)a_i] \neq 0(mod 17)$ since 17 is prime hence no existence of zero divisors under multiplication	Transposition detected due to difference in summation of the parity check equation but the faulty bit strings are not identified
Case 4	$a_i > a_{i+k}$ (clearly if $a_i = 0$ then a_{i+k} does not exist, for this case)	$ia_i + (i+k)a_{i+k} < ia_{i+k} + (i+k)a_i$ The products $[ia_i], [(i+k)a_{i+k}]$, $[ia_{i+k}]$and $[(i+k)a_i] \neq 0(mod 17)$ since 17 is prime hence no existence of zero divisors under multiplication	Transposition detected due to difference in summation of the parity check equation but the faulty bit strings are not identified

In case 1 and 2, $a_i = a_{i+k}$ which means there is no transposition.

In case 3 and 4, upon computation of the parity check equation, the sent and received code words yield a different summation hence the transposition error is detected. Cases $1, 2, 3$ and 4 represent all possible single transpositions that can occur and as seen, the existence of error is detected but they do not show how to identify the faulty bit strings. Suppose the bit string a_k and a_w where $k \neq w$ (otherwise there is no transposition) are the faulty bit strings.

Then b_k and b_w are then chosen such that $W_{i \neq k,w} + kb_k + wb_w \equiv 0(mod 17)$

That is

$$kb_k + wb_w \equiv (-(W_{i \neq k,w}))(mod 17)$$

Example 4.4.2. Let the code word $810567889990000G$ be the sent code word and suppose that a single transposition error occur to yield $180567889990000G$ as the received code word.

The order of flow is not broken by the error but the parity check equation yields $7(mod17)$ hence an error(s) exists. This error may be interpreted as a check digit error which can be corrected as in section 4.3.2 above but will yield the code word $180567889990000F$ which is not the sent code word hence the conclusion that other error(s) exists. The error may also be interpreted as a single error on any bit string and can be corrected as in section 4.3.3 above but this will not yield the sent code word. Prior knowledge of the type of error caused is therefore needed for identification of the faulty bit strings otherwise it is very cumbersome. Suppose it is known that a single transposition error occurred during transmission on digit position one and two.
Thus $k = 1$ and $w = 2$, then $W_{i \neq 1,2} = 7(mod17)$
and $1b_1 + 2b_2 \equiv -7 \equiv 10(mod17)$. Thus $b_1 + 2b_2 \equiv 10(mod17)$
Choosing b_1 and b_2 from $0, 1, \ldots, 16$ to satisfy this equation yield $b_1 = 8$ and $b_2 = 1$. Other pairs (for example $b_1 = 2$ and $b_2 = 4$ or $b_1 = 6$ and $b_2 = 2$) may exist but comparing with the sent code word, $b_1 = 8$ and $b_2 = 1$ is the correct pair yielding to $810567889990000G$ as in the sent code word.

Theorem 4.4.2. *The ISBN-16 code detects any double transposition error.*

Proof. If the transposition leads to the order of flow of digits being broken on all (four) transposed bit string positions, the double error is detected. If the transposition leads to the order of flow of digits not being broken or it is broken only on some (one, two or three), but not all bit string positions, the error may or may not be detected by the observation method but will still be detected by the parity check equation.

Let $u = (a_1 a_2 \ldots a_{16})$ be the sent code word and suppose a double transposition error occurred during transmission on some bit strings a_i and a_h (for some $i, h = 1, 2, \ldots, 16$) to yield a received code word b such that a_i and a_h are transposed to a_{i+k} and a_{h+p} where $k, p = 1, 2, \ldots, 16(mod 17)$ respectively.

Table 4.8: Double transposition error detection in ISBN-16

	Possibility	**Effect on parity check equation after transposition**	**Transposition detection**
Case 1	$a_i = a_{i+k} = a_h = a_{h+p} = 0$	$ia_i + (i+k)a_{i+k} + ha_h + (h+p)a_{h+p} = (i+k)a_i + ia_{i+k} + (h+p)a_h + ha_{h+p}$	No Transposition
Case 2	$a_i = a_{i+k} = a_h = a_{h+p} \neq 0$	$ia_i + (i+k)a_{i+k} + ha_h + (h+p)a_{h+p} = (i+k)a_i + ia_{i+k} + (h+p)a_h + ha_{h+p}$	No Transposition
Case 3	$a_i \neq a_{i+k}$ but $a_h = a_{h+p}$ or vice versa	Single transposition	Detected as discussed in theorem 4.4.1 above

Case 4	$a_i < a_{i+k}$ and $a_h < a_{h+p}$	$ia_i + (i+k)a_{i+k} + ha_h + (h+p)a_{h+p} > (i+k)a_i + ia_{i+k} + (h+p)a_h + ha_{h+p}$ all these products $\neq 0 (mod 17)$since 17 is prime hence no zero divisors under multiplication	Transposition detected due to difference in summation of the parity check equation
Case 5	$a_i > a_{i+k}$ and $a_h > a_{h+p}$	$ia_i + (i+k)a_{i+k} + ha_h + (h+p)a_{h+p} < (i+k)a_i + ia_{i+k} + (h+p)a_h + ha_{h+p}$ all these products $\neq 0 (mod 17)$ since 17 is prime hence no zero divisors under multiplication	Transposition detected due to difference in summation of the parity check equation

Case 6	$a_i > a_{i+k}$ and $a_h < a_{h+p}$ or $a_i < a_{i+k}$ and $a_h > a_{h+p}$	$ia_i + (i+k)a_{i+k} + ha_h + (h+p)a_{h+p} \neq (i+k)a_i + ia_{i+k} + (h+p)a_h + ha_{h+p}$ due to uniqueness of products when working with $mod 17$, which is prime. All these products $\neq 0 (mod 17)$ since 17 is prime hence no zero divisors under multiplication.	Transposition detected due to difference in summation of the parity check equation

In case 1 and 2, $a_i = a_{i+k}$ which means there is no transposition. Case 3 is a single transposition and is discussed in theorem 4.4.1 above. In cases 4, 5 and 6, upon computation of the parity check equation, the sent and received code words yield different summations as shown in each case hence the transposition error is detected. All these six cases represent all possible double transpositions that can occur and as seen, they are detected. □

Corollary 4.4.1. *The ISBN-16 code corrects any single and double transposition errors that break the order of flow of bit strings on both bit string positions.*

Proof. Let $u = (a_1 a_2 \ldots a_{16})$ be the sent code word and let $v = (a_1 \ldots a_i \ldots a_i + k \ldots a_{16})$ and suppose during transmission, any single or double transposition error that breaks the order of flow of bit strings occurred. As seen in Theorems 4.4.1 and 4.4.2 above, the errors will be detected. To correct the single transposition

error, the faulty bit strings are re-transposed and parity check equation worked out to confirm validity, if it holds then the corrected code word is the valid sent code word. Otherwise other errors exist (not single transposition errors).
To correct the double transposition error, the faulty bit strings are re-transposed in the four positions and parity check equation worked out to confirm validity. If it holds then the corrected code word is valid otherwise other errors exist (not double transposition errors). The re-transposition may be cumbersome since four bit string positions are being chosen from four choices yielding to $4^4 = 256$ choices which is the upper limit on the choices that can be made but since the order of flow of digits must be obeyed, these cases reduce. □

Corollary 4.4.2. *The ISBN-16 code detects any multiple errors that break the order of flow of bit strings.*

Proof. Any multiple errors that break the order of flow of bit strings can be detected by the observation method hence proven. □

It is shown later in Proposition 4.6.2 that the ISBN-16 code cannot detect or correct some multiple errors that do not break the order of flow of bit strings.

4.5 Conversion Tool from ISBN-13 to ISBN-16

4.5.1 Guidelines towards designing a conversion tool

As discussed in section 1.1.5, the ISBN-13 code was designed as an improvement of the ISBN-10 code. There was therefore a need to have a conversion tool to convert the existing ISBN-10 code words to the ISBN-13.
The following guidelines are vital towards designing a conversion tool for the existing code words to the new ISBN-16 code. The guidelines act as a blue print to ensure that the new conversion tool is efficient and effective. Let $f : I_{13} \to I_{17}$

be a function from the existing ISBN-13 code to the ISBN-16 defined as,

$$f(a_1a_2...a_{13}) = b_1b_2...b_{13}00b_{16} \text{ where}$$

$$b_{16} = (-(\sum_{i=1}^{15} ib_i) \times 16) mod 17$$

Then, the following propositions must be obeyed and act as a guideline for defining the function.

Proposition 4.5.1. *The function f is injective (one to one).*

Proof. By contradiction, suppose the function $\boldsymbol{f}$ is not one to one and let u and v be two different code words in the ISBN-13 code and let $f(u) = f(v)$.
Since u and v are different, they differ in at least 1 bit string position. Since there is uniqueness of inverses $modulo17$ in the ISBN-16 code, and since $f(u)$ and $f(v)$ are ISBN-16 code words then $u = v$ otherwise the uniqueness of inverses won't be obeyed which would mean that 17 is not prime. □

Proposition 4.5.2. *The function* $\boldsymbol{f}$ *is not necessarily surjective (onto).*

Proof. By contradiction, suppose the function $\boldsymbol{f}$ is onto. Since the function is one to one as shown in proposition above, the ISBN-13 code must have the same dictionary as the ISBN-16 code which is a contradiction. Hence the function is not necessarily onto. □

4.5.2 The Conversion tool

Since all ISBN-10 code words have an ISBN-13 representation, to convert an ISBN-13 code word to an ISBN-16 code word, the following steps are followed:

1. The order of flow of the thirteen consecutive bit strings in the ISBN-13 code word is considered. Since the ISBN-13 codewords start with the prefix 978 or 979, convertion is done to yield the prefixes 970 and 971 respectively.

2. If the order of flow (incresing or decreasing) of the remaining ten bitstrings (including the check digit) is obeyed, step four and five are followed. Otherwise, step three is followed. The thirteenth bitstring (the check digit) in the ISBN-13 code word is considered as part of the new codeword.
3. If the order of flow of the other bitstrings is not obeyed, any bitstring that breaks the order of flow is replaced with a zero (0); this ensures that the order of flow is obeyed.
4. The digits "00" are added to represent the fourteenth and fifteenth bit strings.
5. The new check digit b_{16} is computed as in section 4.1.

For example, to convert the ISBN-13 code word 9780198538035 to an ISBN-16 code word, it is noticed that the seventh and tenth bitstrings break the order of flow. They are replaced by zeros to yield $9700190530035a_{14}a_{15}a_{16}$. Putting "00" for the fourteenth and fifteenth bit strings yields to $970019053003500a_{16}$. Finally the check digit a_{16} is computed as in section 4.1 to yield $970019053003500C$ as the ISBN-16 code word.
This conversion tool is efficient and reliable since it can convert all the ISBN-13 and ISBN-10 code words to ISBN-16. It also maintains some of the bit strings in the ISBN-13 code words that do not break the order of flow.
Other converted code words from ISBN-13 code to ISBN-16 code include:

Table 4.10: Converted codeword ISBN-13 to ISBN-16 code

Book title, year and author	Assigned ISBN-13	Conversion to new ISBN-16
Van Lint J.H., (1998) *Introduction to Coding Theory*	9783540641339	970350064103900
Todd K. Moon., (2005), *Error correction coding: mathematical methods and algorithms*	9780471648000	9700471648000003
Sebastia X. D., (2003), *Block error correcting codes.*	9783540003953	9703540003953006
Kenneth H. Rosen, (2003), *Discrete mathematics and its applications.*	9780072930337	970007293033700F
Houghton A., (2001), *Error coding for engineers*	9780792375227	9700792375227009

4.5.3 The ISBN-16 generator

ISBN-16 generator is an application developed using an algorithm derived from the guidelines in section 4.5.1 and written in .Net framework's $C\#$ programming language, to generate codewords. This section discusses the ISBN-16 generator with the following screen shots illustrating how the application works.

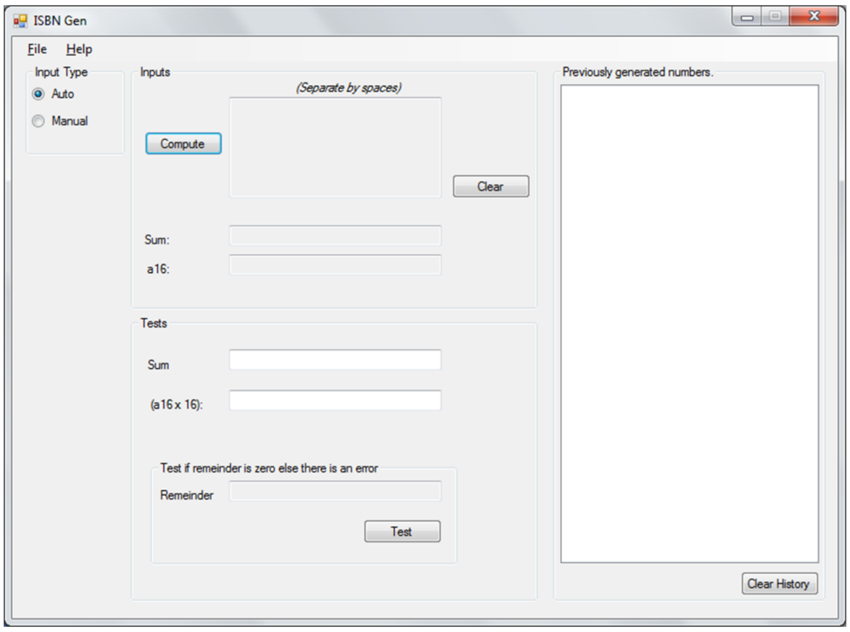
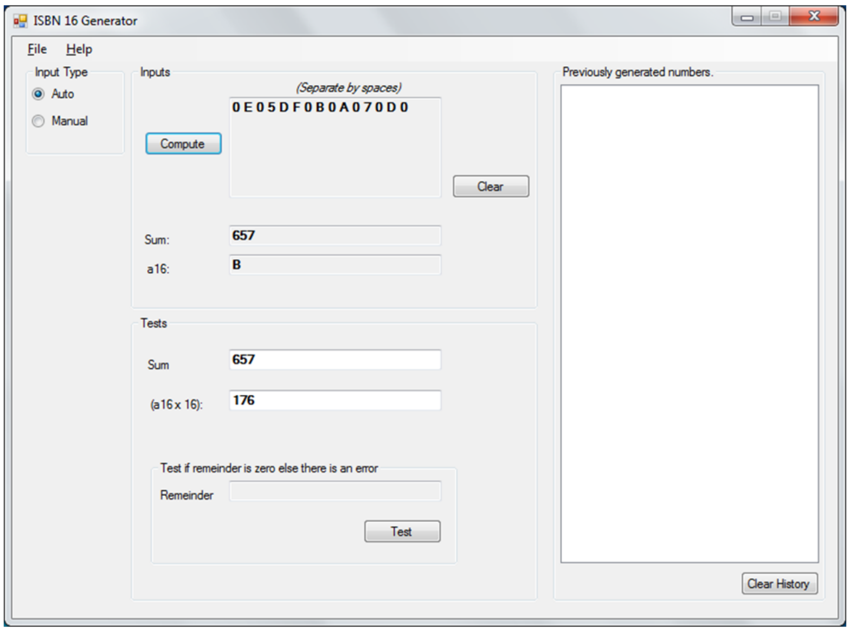


Figure 4.1: Main Window

There are two options in code generation using ISBN 16 application as depicted in figure 4.1 in the input type section . The numbers can be generated automatically or manually.

Automatic Generation

This is done by selecting the auto option under the input type section. Numbers are randomly generated from the set $\mathbb{Z}_{17}$. It is from this set that the 16^{th} position is calculated by hitting the compute button as shown in the figure 4.2 :


ISBN 16 Generator
File
Help
Compute
Clear
Test
Clear History
Previously generated numbers.
Test if remeinder is zero else there is an error

Input Type: [✓ Auto] Manual

Inputs (Separate by spaces): 0 E 0 5 D F 0 B 0 A 0 7 0 D 0

Sum: 657

a16: B

Tests

Sum: 657

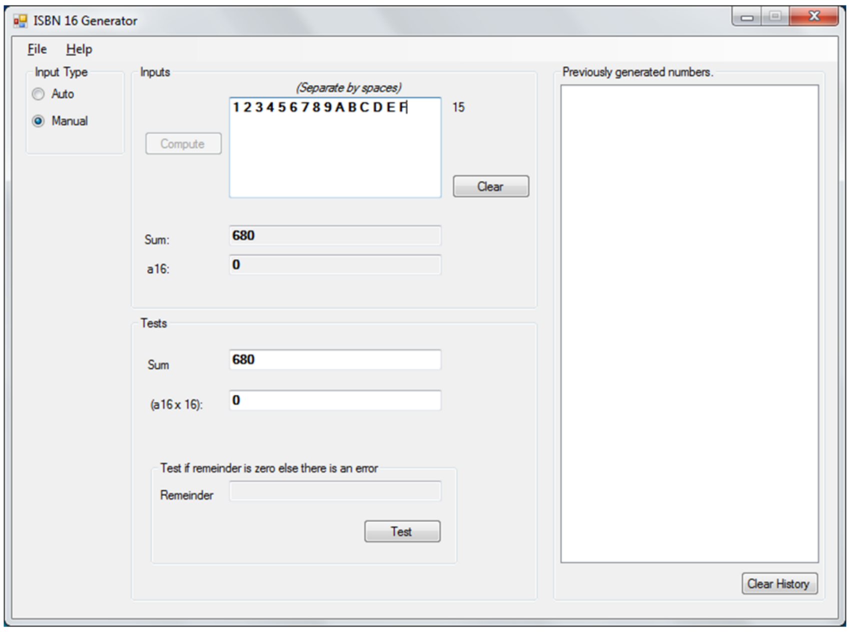
(a16 x 16): 176

Remeinder:

Figure 4.2: Automatic Generation

Manual Generation

With this option, the numbers are input by the user of the application. These numbers may be input from the set $\mathbb{Z}_{17}$ as shown in figure 4.3 :


ISBN 16 Generator
File
Help
Compute
Clear
Test
Clear History
Previously generated numbers.
Test if remeinder is zero else there is an error

Input Type: Auto [✓ Manual]

Inputs (Separate by spaces): 1 2 3 4 5 6 7 8 9 A B C D E F 15

Sum: 680

a16: 0

Tests

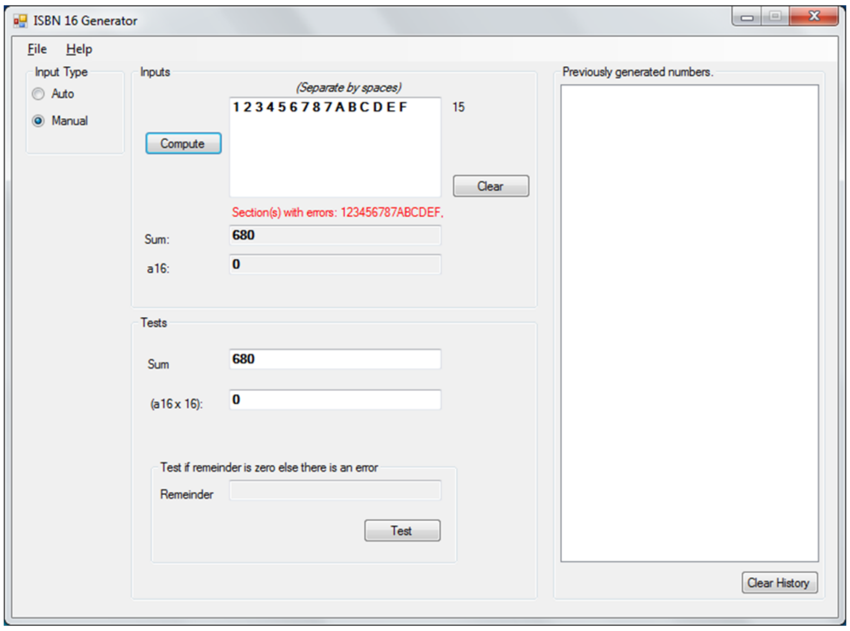
Sum: 680

(a16 x 16): 0

Remeinder:

Figure 4.3: Manual Generation

This could be error prone thus, ISBN-16 application is capable of error detection as shown in figure 4.4 ;



Input Type: [✓ Manual]

1 2 3 4 5 6 7 8 7 A B C D E F   15

Section(s) with errors: 123456787ABCDEF.

Sum: 680

a16: 0

Sum 680

(a16 x 16): 0

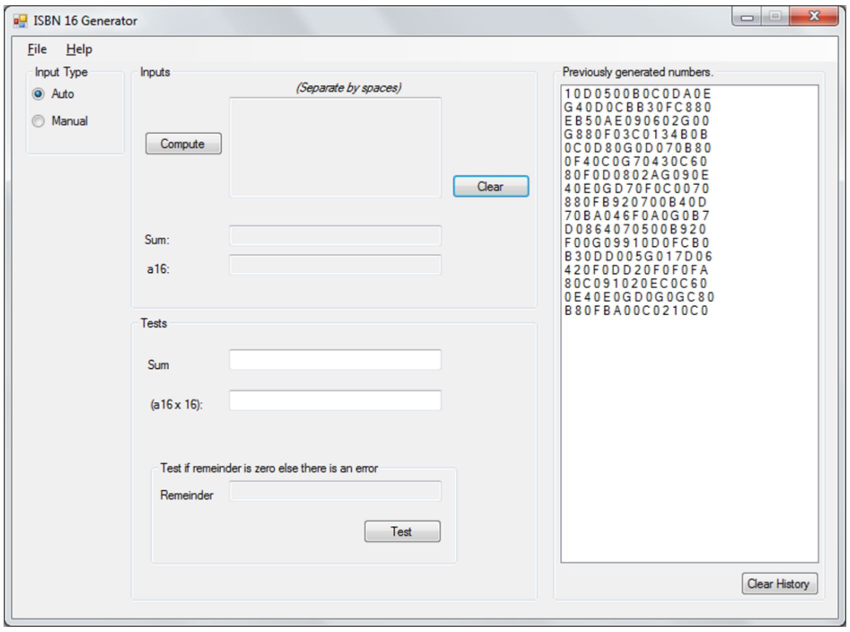
Test if remeinder is zero else there is an error

Remeinder

Figure 4.4: Error Handling

History

Upon computations, the numbers are cleared by using the "clear" button, they are then listed in the previously generated numbers display on the right hand side and can be used for reference. This history can be cleared with hitting the "clear history" button below the historical numbers. The figure 4.5 shows seventeen previously generated numbers.



Previously generated numbers.

```
10D0500B0C0DA0E
G40D0CBB30FC880
EB50AE090602G00
G880F03C0134B0B
0C0D80G0D070B80
0F40C0G70430C60
80F0D0802AG090E
40E0GD70F0C0070
880FB920700B40D
70BA046F0A0G0B7
D0864070500B920
F00G09910D0FCB0
B30DD005G017D06
420F0DD20F0F0FA
80C091020EC0C60
0E40E0GD0G0GC80
B80FBA00C0210C0
```

Figure 4.5: History

4.6 Weaknesses in the ISBN-16 Code

The error correction may be very cumbersome for multiple errors. Also as explained in section 4.4.1, for some multiple errors in codewords, error correction may yield multiple valid codewords and identifying the intended codeword may be impossible if the check digit is not known. In some cases, if multiple error occur on consecutive bitstrings, one may not know the correct order of flow. This may make error correction impossible.

Proposition 4.6.1. The dictionary of the ISBN-16 code may never reach the upper limit of 15^{16} code words.

Proof. By counter example, consider a code word that has the first four bit strings being $5687\ldots$ the fifth bit string can only be a $7, 8, \ldots, 16$ or 0. The bit strings $1, 2, 3, 4, 5$ or 6 cannot occupy this position since they would break the order of flow. This means that the number of all possible permutations on the fifth bit string is ten not the expected sixteen. This in turn means that the upper limit may never be achieved. □

Proposition 4.6.2. *The ISBN-16 code cannot detect or correct some multiple silent errors.*

Silent errors in ISBN-16 code are defined as errors that do not break the order of flow of bit strings and yet the check digit is the same.

Proof. Consider the sent code word $u = 1111111111111111$ versus the received code word $v = 1111110110111111$.
Working out the parity check equations, both code words yield 1 as the check digit. These are silent errors which go undetected by both the observation method and the parity check method. Since the errors are not detected, they cannot be corrected. □

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMENDATION

5.1 Summary

This research determined the size of the dictionaries in the ISBN-10 and ISBN-13 codes and gave an upper limit that both codes can ever achieve. The research also showed that the dictionaries reduce against error detection and correction capalities.

The efficiency of the Cyclic ISBN-10 code in error detection and correction against the size of the dictionary was analyzed and is shown that the code is inefficient.

The ISBN-16 code was designed to improve the ISBN-10, ISBN-13 and Cyclic ISBN-10 codes in errors detection and correction capabilities against the dictionary size. It is shown that the code is better as far as error detection and correction and dictionary size are concerned.

A computer program that generates the ISBN-16 was developed and implemented as shown in the demonstration screen shots in subsection 4.5.3. The program can either generate codes automatically or manually. The user can therefore input values, the program check for errors and computes the check digit.

An algorithm for the conversion from already existing ISBN-13 code to ISBN-16 code was designed. Any ISBN-10 code word has an ISBN-13 equivalent and thus conversion to ISBN-16 by use of the equivalents.

5.2 Conclusion

In ISBN-10, Silent errors can go unnoticed. To detect an error in ISBN-10 code word, one has to work out the parity check equation. It can only correct single errors and transpose errors and for the transpose errors there has to be prior knowledge of the existence of the transpose error. The conditions given for detection of at least two errors only tell if one can detect the double error and do not show how to correct it and multiple (more than two) errors cannot be corrected. ISBN-10 has a relatively small dictionary.

The cyclic ISBN-10 is not a cyclic code since any left or right cyclic shift does not necessarily yield another codeword since it does not meet the parity check equation. The dictionary size is smaller than even the ISBN-10 code, the total number of code words is less than 9×10^8 which forms a major foundation for any other code that would be developed using this approach. The process of correction of multiple errors is bit tedious and inacurate since it may end up having very many choices of possible digits to replace the ones in error.

The ISBN-13 code cannot indicate the exact bit string which is in error and cannot detect some transposition errors and some double or multiple errors. It also cannot correct the double, multiple or transposition errors.

The ISBN-16 code detects and corrects any single error, single transposition error and double transposition error that break the order of flow of bit strings on both bit string positions and detects any multiple errors that break the order of flow of bit strings. Moreover, it has been established that the ISBN-16 code has a big dictionary.

5.3 Recommendations for future research

1. Computing the exact dictionary of the ISBN-16. The research determined the upper limit of the ISBN-16 code; the actual dictionary of the code with respect to varying coditions on error detections and corrections need to be determined.

2. Detection of multiple silent errors. Existence of multiple silent errors in

a code word without any ability by the code to detect them needs to be determined.

3. Design an automated conversion tool for the existing ISBN-13 code to ISBN-16. The conversion tool given in 4.5 only gives the algorithm that can be used to convert an ISBN-13 TO ISBN-16. An automated conversion tool that can be developed.

REFERENCES

Atlas-World (2012). How many countries? Retrieved from: http://www.worldatlas.com/nations.htm, Date accessed: April 2011.

Doumen, J. M. (2003). *Some Applications of Coding Theory in Cryptography.* Eindhoven University Press, Germany, Germany.

Egghe, L. (1985). A note concerning two isbn checking algorithms. *Journal of information science*, 11:41–42.

Egghe, L. (1999). Detection and correction of multiple errors in general block codes. *Mathematical and computer modelling*, 30(7):113–121.

Egghe, L. (2005). The coding of the isbn. Retrieved from: http://en.scientificcommons.org /leo_ e gghe, Date accessed: 6th May 2010.

Egghe, L. and Ronald, R. (2005). On the detection of double errors in isbn and issn-like codes. Retrieved from: http://en.scientificcommons. org/leo_egghe, Date accessed: 6th May 2010.

Eric, W. (2010a). Isbn code. Retrieved from: http://mathworld.wolfram.com/ISBN.html, Date accessed: 5th May 2010.

Eric, W. (2010b). Isbn code. Retrieved from: http://mathworld.wolfram.com/Transposition.html, Date accessed: 5th May 2010.

Henk, V. T. (1993). Coding theory; a first course. Retrieved from: Date accessed: 6th May 2010, http://hyperelliptic.org/tanja/teaching/ CCI11/CODING.pdf.

Houghton, A., One, A., Two, A., and Three, A. (2001). *Error coding for engineers.* Kluwer Academic Publisher.

Irving, S. R. and Chen, X. (1999). *Error-Control Coding for Data Networks*. Kluwer Academic Publishers.

Jacobus, H. (1973). *Coding Theory*. Springer Verlag.

Kenneth, H. R. (1993). *Elementary number theory and its applications*. Addison-Wesley Publishing company.

Kenneth, H. R. (2003). *Discrete mathematics and its applications*. McGraw-Hill Publishing company.

Nyaga, L. and Cecilia, M. (2008). Increasing error detection and correction efficiency in the isbn. *Discovery and Innovation*, 20:3–4.

Raymond, H. (1986). *A first Course in Coding Theory*. CLARENDON Press, U.S.A.

Sebastia, X. D. (2003). *Block error correcting codes*. Springer Verlag. ISBN 3-540-00395-9.

Tervo, R. (1998). Secrets of the isbn - an error detection method. Retrieved from: http://www.ee.unb.ca/tervo/ee4253/isbn.shtml, Date accessed: April 2011.

Todd, K. M. (2005). *Error Correction Coding: Mathematical Methods and Algorithms*. John Wiley & Sons Inc.

Uppal, S. M. and Humphreys, H. M. (2008). *Mathematics for Science*. New Age International, New Delhi, India, 2nd edition. ISBN 978-8122409949.

Van, L. J. H. (1998). *Introduction to Coding Theory*. Springer Verlag, 3rd edition. ISBN 3-540-64133-5.

Viklund, A. (2007). Isbn information home. Retrieved from : http://isbn-information.com/ind ex.html, Date accessed: 5th May 2010.