Tendências em Matemática Aplicada e Computacional, **19**, N. 2 (2018), 369-389

www.scielo.br/tema
doi: 10.5540/tema.2018.019.02.0369

Sequences of Primitive and Non-primitive BCH Codes

A.S. ANSARI[1], T. SHAH[1], ZIA-UR-RAHMAN[1] and A.A. ANDRADE[2]

Received on March 27, 2017 / Accepted on March 21, 2018

ABSTRACT. In this work, we introduce a method by which it is established that how a sequence of non-primitive BCH codes can be obtained by a given primitive BCH code. For this, we rush to the out of routine assembling technique of BCH codes and use the structure of monoid rings instead of polynomial rings. Accordingly, it is gotten that there is a sequence $\{C_{b^j n}\}_{1\leq j\leq m}$, where $b^j n$ is the length of $C_{b^j n}$, of non-primitive binary BCH codes against a given binary BCH code C_n of length n. Matlab based simulated algorithms for encoding and decoding for these type of codes are introduced. Matlab provides in routines for construction of a primitive BCH code, but impose several constraints, like degree s of primitive irreducible polynomial should be less than 16. This work focuses on non-primitive irreducible polynomials having degree bs, which go far more than 16.

Keywords: Monoid ring, BCH codes, primitive polynomial, non-primitive polynomial.

1 INTRODUCTION

Introducing more general algebraic structures lead to various gains in coding applications and the generality of the algebraic structures helps to find more efficient encoding and decoding algorithms for known codes. This has motivated special attention of many researchers in considering ideals of certain rings [1], [5], [6], [9] and [10]. The extension of a BCH code embedded in a semigroup ring was considered by Cazaran in [4]. More information regarding ring constructions and its corresponding polynomial codes were given by Kelarev [5]. In [1], the authors elaborated cyclic, BCH, Alternant, Goppa and Srivastava codes over finite rings, which are constructed through a polynomial ring in one indeterminate. Several classes of cyclic codes constructed using the monoid rings are discussed in [2], [14], [13], [12], [11] and [15]. These constructions address the error correction and the code rate in a good way. In [16], Shah et al., showed the existence of a binary cyclic code of length $(n+1)n$ such that a binary BCH code of length n is embedded in it. Though they were not succeeded to show the existence of binary BCH code of length

*Corresponding author: Antonio A. Andrade – E-mail: antonio.andrade@unesp.br
[1]Department of Mathematics, Quaid-i-Azam University, Islamabad, Pakistan. E-mail: asia_ansari@hotmail.com; stariqshah@gmail.com; ziaagikian@gmail.com
[2]Departamento de Matematica, Universidade Estadual Paulista – São José do Rio Preto, SP, Brazil. E-mail: andrade@ibilce.unesp.br

$(n+1)n$ corresponding to a given binary BCH code of length n. In [17], a construction method is given by which cyclic codes are ideals in $\mathbb{F}_2[x;a\mathbb{Z}_0]_n$, $\mathbb{F}_2[x]_{an}$, $\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]_{bn}$ and $\mathbb{F}_2[x;\frac{1}{b}\mathbb{Z}_0]_{abn}$, where $\mathbb{Z}_0$ is the set of non-negative integers, $a,b\in\mathbb{Z}$ and $a,b>1$. These codes are capable of correcting random as well as burst errors. Moreover, a link between all these codes are also been developed. Furthermore, in [3], the work of [16] is improved and an association between primitive and non-primitive binary BCH codes is obtained by using the monoid ring $\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$, where $a,b>1$. It is noticed that the monoid ring $\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$ does not contain the polynomial ring $\mathbb{F}_2[x]$ for $a>1$. To handle this situation, a BCH code in the monoid ring $\mathbb{F}_2[x;a\mathbb{Z}_0]$ is constructed and then the existence of a non-primitive BCH code in the monoid ring $\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$ is showed. The non-primitive BCH code C_{bn} in the monoid ring $\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$ is of length bn and generated by a generalized polynomial $g(x^{\frac{a}{b}})\in\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$ of degree br. In this line, corresponding to a binary BCH code C_n of length n generated by a generalized polynomial $g(x^a)\in\mathbb{F}_2[x;a\mathbb{Z}_0]$ of degree r it is constructed a code C_{bn} such that C_n is embedded in C_{bn}.

This work extends the work of [3], where the monoid ring $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$, with $b=a+i$, $1\le i,j\le m$, where m is a positive integer, is used. Corresponding to the sequence $\{\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]\}_{j\ge1}$ of monoid rings, we obtain a sequence of non-primitive binary BCH codes $\{C^j_{b^jn}\}_{j\ge1}$, based on a primitive BCH code C_n of length n. The non-primitive BCH code $C^j_{b^jn}$ in the monoid ring $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ is of length b^jn and generated by a generalized polynomial $g(x^{\frac{a}{b^j}})\in\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ of degree b^jr. Similarly, corresponding to a given binary BCH code C_n of length n generated by a polynomial $g(x^a)\in\mathbb{F}_2[x;a\mathbb{Z}_0]$ of degree r it is constructed a code $C^j_{b^jn}$ such that C_n is embedded in $C^j_{b^jn}$, where the length of the binary BCH code $C^j_{b^jn}$ is well controlled with better error correction capability. Along with the construction of a sequence $\{C^j_{b^jn}\}_{j\ge1}$ of non-primitive binary BCH codes our focus is on its simulation as well, where the simulation is carried out using *Matlab*. It provides built in routines only for primitive BCH codes with degree of primitive polynomial less than 16. Whereas in constructing non-primitive BCH codes, the degree of non-primitive polynomial goes beyond 16, where to overcome this situation *Generic Algorithm* is developed in Matlab. The whole method of the algorithm is carried out in two major steps: I) Generating non-primitive polynomial and II) Error correction in received polynomial. In Table 5, some examples are listed. The non-primitive BCH codes with same code rate and error corrections are found to be interleaved codes. By interleaving a t random error correcting (n,k) code to degree β, we obtain a $(\beta n,\beta k)$ code which is capable of correcting any combination of t bursts of length β or less [7, Section 9.4], where the non-primitive BCH codes $C^j_{b^jn}$ have burst error correction capability along with the random error correction.

2 BCH CODES IN $\mathbb{F}_2[X;\frac{A}{B^J}\mathbb{Z}_0]_{B^JN}$, WHERE $1\le J\le M$.

The set of all finitely nonzero functions f from a commutative monoid $(S,*)$ into the binary field $\mathbb{F}_2$ is denoted by $\mathbb{F}_2(S)$. This set $\mathbb{F}_2(S)$ is a ring with respect to binary operations addition and multiplication defined as: $(f+g)(s)=f(s)+g(s)$ and $(fg)(s)=\sum_{t*u=s}f(t)g(u)$, where the symbol $\sum_{t*u=s}$ indicates that the sum is taken over all pairs (t,u) of elements of S such that

$t*u=s$ and it is understood that in the situation where s is not expressible in the form $t*u$ for any $t,u\in S$, then $(fg)(s)=0$. The ring $\mathbb{F}_2(S)$ is known as the *monoid ring* of S over $\mathbb{F}_2$ and it is represented by $\mathbb{F}_2[x;S]$ whenever S is an additive monoid. A nonzero element f of $\mathbb{F}_2[x;S]$ is uniquely represented in the canonical form $\sum_{i=1}^{n} f(s_i)x^{s_i}=\sum_{i=1}^{n} f_i x^{s_i}$, where $f_i\neq 0$ and $s_i\neq s_j$ for $i\neq j$. The monoid ring $\mathbb{F}_2[x;S]$ is a polynomial ring in one indeterminate if $S=\mathbb{Z}_0$.

The indeterminate of generalized polynomials in monoid rings $\mathbb{F}_2[x;a\mathbb{Z}_0]$ and $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ are given by x^a and $x^{\frac{a}{b^j}}$ for each $1\leq j\leq m$, where m is a fix positive integer, and they behave like an indeterminate x in $\mathbb{F}_2[x]$. The arbitrary elements in $\mathbb{F}_2[x;a\mathbb{Z}_0]$ and $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ are written, respectively, as $f(x^a)=1+(x^a)+(x^a)^2+\cdots+(x^a)^n$ and $f(x^{\frac{a}{b^j}})=1+(x^{\frac{a}{b^j}})+(x^{\frac{a}{b^j}})^2+\cdots(x^{\frac{a}{b^j}})^n$. The monoids $a\mathbb{Z}_0$ and $\frac{a}{b^j}\mathbb{Z}_0$ are totally ordered, so degree and order of elements of $\mathbb{F}_2[x;a\mathbb{Z}_0]$ and $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ are defined.

As $\mathbb{F}_2[x;a\mathbb{Z}_0]\subset\mathbb{F}_2[x]$, it follows that the construction of BCH code in the factor ring $\mathbb{F}_2[x;a\mathbb{Z}_0]_n$ is similar to the construction of a BCH code in $\mathbb{F}_2[x]_n$. Whereas the construction of BCH codes in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]/((x^{\frac{a}{b^j}})^{b^jn}-1)\equiv\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]_{b^jn}$ is based on the values of b. BCH codes of length b^jn in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]_{b^jn}$, corresponding to a BCH code C_n of length n don't exist for all values of b, where $b=a+i$, such that $1\leq i\leq m$, and a, m are positive integers. For that, consider the following map $p(x^a)=p_0+p_1x^a+\cdots+p_{n-1}(x^a)^{n-1}\mapsto p_0+p_1(x^{\frac{a}{b^j}})^{b^j}+\cdots+p_{n-1}(x^{\frac{a}{b^j}})^{b^j(n-1)}=p(x^{\frac{a}{b^j}})$, which convert a primitive polynomial $p(x^a)$ of degree s in $\mathbb{F}_2[x;a\mathbb{Z}_0]$ to a polynomial $p(x^{\frac{a}{b^j}})$ of degree b^js in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$. This converted polynomial is found to be non-primitive, but is irreducible for some values of b. Therefore, for the construction of a non-primitive BCH code in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]_{b^jn}$, only this specific value of b is selected for which there is an irreducible polynomial $p(x^{\frac{a}{b^j}})$ in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$. Thus, for positive integers c_j, d_j and b^jn such that $2\leq d_j\leq b^jn$ with b^jn is relatively prime to 2, there exists a non-primitive binary BCH code C_{b^jn} of length b^jn, where b^jn is order of an element $\alpha\in\mathbb{F}_{2^{b^js}}$.

In Table 1, we present a list of irreducible polynomials of degree b^js in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ corresponding to primitive irreducible polynomial of degree s in $\mathbb{F}_2[x;a\mathbb{Z}_0]$, where for $p(x^a)\in\mathbb{F}_2[x;a\mathbb{Z}_0]$, $p(x^{\frac{a}{b}})\in\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$, $p(x^{\frac{a}{b^2}})\in\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$, replace $x^a,x^{\frac{a}{b}}$ and $x^{\frac{a}{b^2}}$ by x,y and z, respectively.

Proposition 1. *[3, Proposition 2] If $p(x^a)\in\mathbb{F}_2[x;a\mathbb{Z}_0]$ is a primitive irreducible polynomial of degree $s\in\{2l,3l,4l,6l\}$, where $l\in\mathbb{Z}_0$, then the corresponding generalized polynomial $p(x^{\frac{a}{b^j}})$ of degree b^js in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ is non-primitive irreducible for $b\in\{3,7,\{3,5\},\{3,7\}\}$, respectively.*

Theorem 2. *[3, Theorem 3] Let $n=2^s-1$ be the length of a primitive BCH code C_n, where $p(x^a)\in\mathbb{F}_2[x;a\mathbb{Z}_0]$ is a primitive irreducible polynomial of degree s such that $p(x^{\frac{a}{b^j}})\in\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ is a non-primitive irreducible polynomial of degree b^js.*

1. *For positive integers c_j,d_j,b^jn such that $2\leq d_j\leq b^jn$ and b^jn are relatively prime to 2, there exist a non-primitive binary BCH code C_{b^jn} of length b^jn, where b^jn is the order of an element $\alpha\in\mathbb{F}_{2^{b^js}}$.*

Table 1: Irreducible polynomials of degree $b^j s$ in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ corresponding to primitive irreducible polynomial of degree s in $\mathbb{F}_2[x;a\mathbb{Z}_0]$.

$p(x^a)\in\mathbb{F}_2[x;a\mathbb{Z}_0]$	$p(x^{\frac{a}{b}})\in\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$	$p(x^{\frac{a}{b^2}})\in\mathbb{F}_2[x;\frac{a}{b}\mathbb{Z}_0]$
$1+(x^a)+(x^a)^3$	$1+(x^{\frac{a}{7}})^7+(x^{\frac{a}{7}})^{21}$	$1+(x^{\frac{a}{49}})^{49}+(x^{\frac{a}{49}})^{147}$
$1+(x^{\frac{a}{3}})^3+(x^{\frac{a}{3}})^{12}$	$1+(x^{\frac{a}{9}})^9+(x^{\frac{a}{9}})^{36}$	
$1+(x^{\frac{a}{5}})^5+(x^{\frac{a}{5}})^{20}$	$1+(x^{\frac{a}{25}})^{25}+(x^{\frac{a}{25}})^{100}$	
$1+(x^a)^3+(x^a)^4$	$1+(x^{\frac{a}{3}})^9+(x^{\frac{a}{3}})^{12}$	$1+(x^{\frac{a}{9}})^{27}+(x^{\frac{a}{9}})^{36}$
	$1+(x^{\frac{a}{5}})^{15}+(x^{\frac{a}{5}})^{20}$	$1+(x^{\frac{a}{25}})^{75}+(x^{\frac{a}{25}})^{100}$
$1+(x^a)+(x^a)^6$	$1+(x^{\frac{a}{3}})^3+(x^{\frac{a}{3}})^{18}$	$1+(x^{\frac{a}{9}})^9+(x^{\frac{a}{9}})^{54}$
	$1+(x^{\frac{a}{7}})^7+(x^{\frac{a}{7}})^{42}$	$1+(x^{\frac{a}{49}})^{49}+(x^{\frac{a}{49}})^{294}$
$1+(x^a)+(x^a)^3$	$1+(x^{\frac{a}{3}})^3+(x^{\frac{a}{3}})^9+$	$1+(x^{\frac{a}{9}})^9+(x^{\frac{a}{9}})^{27}+$
	$(x^{\frac{a}{3}})^{15}+(x^{\frac{a}{3}})^{24}$	$(x^{\frac{a}{9}})^{45}+(x^{\frac{a}{9}})^{72}$
$+(x^a)^5+(x^a)^8$	$1+(x^{\frac{a}{5}})^5+(x^{\frac{a}{5}})^{15}+$	$1+(x^{\frac{a}{25}})^{25}+(x^{\frac{a}{25}})^{75}+$
	$(x^{\frac{a}{5}})^{25}+(x^{\frac{a}{5}})^{40}$	$(x^{\frac{a}{25}})^{125}+(x^{\frac{a}{25}})^{200}$
$1+(x^a)^4+(x^a)^9$	$1+(x^{\frac{a}{7}})^{28}+(x^{\frac{a}{7}})^{63}$	
	$1+(x^{\frac{a}{49}})^{196}+(x^{\frac{a}{49}})^{441}$	
$\vdots$	$\vdots$	$\vdots$

2. *The non-primitive BCH code $C_{b^j n}$ of length $b^j n$ is defined as*

$$C_{b^j n}=\{v(x^{\frac{a}{b^j}})\in\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]_{b^j n}: v(\alpha^i)=0 \text{ for all } i=c_j,c_j+1,\cdots,c_j+d_j-2.$$

Equivalently, $C_{b^j n}$ is the null space of the matrix

$$H=\begin{bmatrix} 1 & \alpha^{c_j} & \alpha^{2c_j} & \cdots & \alpha^{(bn-1)c_j} \\ 1 & \alpha^{c_j+1} & \alpha^{2(c_j+1)} & \cdots & \alpha^{(bn-1)(c_j+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{c_j+d_j-2} & \alpha^{2(c_j+d_j-2)} & \cdots & \alpha^{(bn-1)(c_j+d_j-2)} \end{bmatrix}.$$

The following example illustrates the construction of a non-primitive BCH code of length $3^2 n$ through $\mathbb{F}_2[x;\frac{2}{3^2}\mathbb{Z}_0]$.

Example 2.1. *Corresponding to a primitive polynomial $p(x^2)=1+(x^2)+(x^2)^4$ in $\mathbb{F}_2[x;2\mathbb{Z}_0]$ there is a non-primitive irreducible polynomial $p(x^{\frac{2}{3^2}})=1+(x^{\frac{2}{9}})^9+(x^{\frac{2}{9}})^{36}$ in $\mathbb{F}_2[x;\frac{2}{3^2}\mathbb{Z}_0]$ (Table*

1). Let $\alpha \in GF(2^{36})$ such that α satisfies the relation $\alpha^{36}+\alpha^9+1=0$. Using this relation, in the Table 2, we obtain the distinct elements of $GF(2^{36})$.

By Table 2, it follows that $b^2n = 3^2 \times 15 = 135$. Now, to calculate the generating polynomial $g(x^{\frac{2}{9}})$, first calculate the minimal polynomials. By [8, Theorem 4.4.2], the following roots

$$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}, \alpha^{64}, \alpha^{128}, \alpha^{121}, \alpha^{107}, \alpha^{79}, \alpha^{23}, \alpha^{46}, \alpha^{92}, \alpha^{49}, \alpha^{98}, \alpha^{61}, \alpha^{122},$$
$$\alpha^{109}, \alpha^{83}, \alpha^{31}, \alpha^{62}, \alpha^{124}, \alpha^{113}, \alpha^{91}, \alpha^{47}, \alpha^{94}, \alpha^{53}, \alpha^{106}, \alpha^{77}, \alpha^{19}\alpha^{38}, \alpha^{76}, \alpha^{17}, \alpha^{34}, \alpha^{68}$$

have same minimal polynomial $m_1(x^{\frac{2}{9}}) = p(x^{\frac{2}{9}}) = 1+(x^{\frac{2}{9}})^9+(x^{\frac{2}{9}})^{36}$. It $m_3(x^{\frac{2}{9}})$ is the minimal polynomial for α^3, then $\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{48}, \alpha^{96}, \alpha^{57}, \alpha^{114}, \alpha^{93}, \alpha^{51}, \alpha^{102}$ and α^{69} all are roots for $m_3(x^{\frac{2}{9}})$. Therefore, it follows that $m_3(x^{\frac{2}{9}}) = (x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^3+1$. Similarly,

$$\begin{aligned}
m_5(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^{18}+(x^{\frac{2}{9}})^9+1, m_7(x^{\frac{2}{9}}) = (x^{\frac{2}{9}})^{36}+(x^{\frac{2}{9}})^{27}+1\\
m_9(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^4+(x^{\frac{2}{9}})+1,\ m_{15}(x^{\frac{2}{9}}) = (x^{\frac{2}{9}})^6+(x^{\frac{2}{9}})^3+1\\
m_{21}(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^9+1\\
m_{27}(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^4+(x^{\frac{2}{9}})^3+(x^{\frac{2}{9}})^2+(x^{\frac{2}{9}})+1\\
m_{45}(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^2+(x^{\frac{2}{9}})+1,\ m_{63}(x^{\frac{2}{9}}) = (x^{\frac{2}{9}})^4+(x^{\frac{2}{9}})^3+1.
\end{aligned}$$

The BCH code with $d_2 = 3$ has generator polynomial $g(x^{\frac{2}{9}}) = 1+(x^{\frac{2}{9}})^9+(x^{\frac{2}{9}})^{36}$. It corrects up to 1 error and its code rate is $\frac{99}{135} = 0.733$. The BCH code with $d_2 = 5$ has generator polynomial

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}})) = ((x^{\frac{2}{9}})^{36}+(x^{\frac{2}{9}})^9+1)((x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^3+1)\\
&= (x^{\frac{2}{9}})^{48}+(x^{\frac{2}{9}})^{39}+(x^{\frac{2}{9}})^{36}+(x^{\frac{2}{9}})^{21}+(x^{\frac{2}{9}})^9+(x^{\frac{2}{9}})^3+1.
\end{aligned}$$

It corrects up to 3 errors and its code rate is $\frac{87}{135} = 0.644$. Similarly, the BCH codes with $d_2 = 7, 9, 15, 21, 27, 45, 63$ and 135 have generator polynomials

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))\\
&= (x^{\frac{2}{9}})^{66}+(x^{\frac{2}{9}})^{54}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{36}+(x^{\frac{2}{9}})^{30}+(x^{\frac{2}{9}})^{27}+(x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^3+1.
\end{aligned}$$

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))\\
&= (x^{\frac{2}{9}})^{102}+(x^{\frac{2}{9}})^{93}+(x^{\frac{2}{9}})^{90}+(x^{\frac{2}{9}})^{57}+(x^{\frac{2}{9}})^{48}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^3+1.
\end{aligned}$$

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))(m_9(x^{\frac{2}{9}}))\\
&= (x^{\frac{2}{9}})^{106}+(x^{\frac{2}{9}})^{103}+(x^{\frac{2}{9}})^{102}+(x^{\frac{2}{9}})^{97}+(x^{\frac{2}{9}})^{93}+(x^{\frac{2}{9}})^{91}+(x^{\frac{2}{9}})^{90}+(x^{\frac{2}{9}})^{61}\\
&\quad +(x^{\frac{2}{9}})^{58}+(x^{\frac{2}{9}})^{57}+(x^{\frac{2}{9}})^{52}+(x^{\frac{2}{9}})^{48}+(x^{\frac{2}{9}})^{46}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{16}+(x^{\frac{2}{9}})^{13}\\
&\quad +(x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^7+(x^{\frac{2}{9}})^3+(x^{\frac{2}{9}})+1.
\end{aligned}$$

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))(m_9(x^{\frac{2}{9}}))(m_{15}(x^{\frac{2}{9}}))\\
&= (x^{\frac{2}{9}})^{112}+(x^{\frac{2}{9}})^{108}+(x^{\frac{2}{9}})^{105}+(x^{\frac{2}{9}})^{102}+(x^{\frac{2}{9}})^{100}+(x^{\frac{2}{9}})^{99}+(x^{\frac{2}{9}})^{94}+(x^{\frac{2}{9}})^{91}\\
&\quad +(x^{\frac{2}{9}})^{90}+(x^{\frac{2}{9}})^{67}+(x^{\frac{2}{9}})^{63}+(x^{\frac{2}{9}})^{60}+(x^{\frac{2}{9}})^{57}+(x^{\frac{2}{9}})^{55}+(x^{\frac{2}{9}})^{54}+(x^{\frac{2}{9}})^{49}\\
&\quad +(x^{\frac{2}{9}})^{46}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{22}+(x^{\frac{2}{9}})^{18}+(x^{\frac{2}{9}})^{15}+(x^{\frac{2}{9}})^{12}+(x^{\frac{2}{9}})^{10}+(x^{\frac{2}{9}})^9\\
&\quad +(x^{\frac{2}{9}})^4+(x^{\frac{2}{9}})+1.
\end{aligned}$$

Table 2: Distinct elements of $GF(2^{36})$.

$\alpha^{36}=1+\alpha^{9}$	$\alpha^{62}=\alpha^{26}+\alpha^{35}$	$\alpha^{88}=\alpha^{16}+\alpha^{34}$	$\alpha^{114}=\alpha^{6}+\alpha^{15}+\alpha^{24}+\alpha^{33}$
$\alpha^{37}=\alpha+\alpha^{10}$	$\alpha^{63}=1+\alpha^{9}+\alpha^{27}$	$\alpha^{89}=\alpha^{17}+\alpha^{35}$	$\alpha^{115}=\alpha^{7}+\alpha^{16}+\alpha^{25}+\alpha^{34}$
$\alpha^{38}=\alpha^{2}+\alpha^{11}$	$\alpha^{64}=\alpha+\alpha^{10}+\alpha^{28}$	$\alpha^{90}=1+\alpha^{9}+\alpha^{18}$	$\alpha^{116}=\alpha^{8}+\alpha^{17}+\alpha^{26}+\alpha^{35}$
$\alpha^{39}=\alpha^{3}+\alpha^{12}$	$\alpha^{65}=\alpha^{2}+\alpha^{11}+\alpha^{29}$	$\alpha^{91}=\alpha+\alpha^{10}+\alpha^{19}$	$\alpha^{117}=1+\alpha^{18}+\alpha^{27}$
$\alpha^{40}=\alpha^{4}+\alpha^{13}$	$\alpha^{66}=\alpha^{3}+\alpha^{12}+\alpha^{30}$	$\alpha^{92}=\alpha^{2}+\alpha^{11}+\alpha^{20}$	$\alpha^{118}=\alpha+\alpha^{19}+\alpha^{28}$
$\alpha^{41}=\alpha^{5}+\alpha^{14}$	$\alpha^{67}=\alpha^{4}+\alpha^{13}+\alpha^{31}$	$\alpha^{93}=\alpha^{3}+\alpha^{12}+\alpha^{21}$	$\alpha^{119}=\alpha^{2}+\alpha^{20}+\alpha^{29}$
$\alpha^{42}=\alpha^{6}+\alpha^{15}$	$\alpha^{68}=\alpha^{5}+\alpha^{14}+\alpha^{32}$	$\alpha^{94}=\alpha^{4}+\alpha^{13}+\alpha^{22}$	$\alpha^{120}=\alpha^{3}+\alpha^{21}+\alpha^{30}$
$\alpha^{43}=\alpha^{7}+\alpha^{16}$	$\alpha^{69}=\alpha^{6}+\alpha^{15}+\alpha^{33}$	$\alpha^{95}=\alpha^{5}+\alpha^{14}+\alpha^{23}$	$\alpha^{121}=\alpha^{4}+\alpha^{22}+\alpha^{31}$
$\alpha^{44}=\alpha^{8}+\alpha^{17}$	$\alpha^{70}=\alpha^{7}+\alpha^{16}+\alpha^{34}$	$\alpha^{96}=\alpha^{6}+\alpha^{15}+\alpha^{24}$	$\alpha^{122}=\alpha^{5}+\alpha^{23}+\alpha^{32}$
$\alpha^{45}=\alpha^{9}+\alpha^{18}$	$\alpha^{71}=\alpha^{8}+\alpha^{17}+\alpha^{35}$	$\alpha^{97}=\alpha^{7}+\alpha^{16}+\alpha^{25}$	$\alpha^{123}=\alpha^{6}+\alpha^{24}+\alpha^{33}$
$\alpha^{46}=\alpha^{10}+\alpha^{19}$	$\alpha^{72}=1+\alpha^{18}$	$\alpha^{98}=\alpha^{8}+\alpha^{17}+\alpha^{26}$	$\alpha^{124}=\alpha^{7}+\alpha^{25}+\alpha^{34}$
$\alpha^{47}=\alpha^{11}+\alpha^{20}$	$\alpha^{73}=\alpha+\alpha^{19}$	$\alpha^{99}=\alpha^{9}+\alpha^{18}+\alpha^{27}$	$\alpha^{125}=\alpha^{8}+\alpha^{26}+\alpha^{35}$
$\alpha^{48}=\alpha^{12}+\alpha^{21}$	$\alpha^{74}=\alpha^{2}+\alpha^{20}$	$\alpha^{100}=\alpha^{10}+\alpha^{19}+\alpha^{28}$	$\alpha^{126}=1+\alpha^{27}$
$\alpha^{49}=\alpha^{13}+\alpha^{22}$	$\alpha^{75}=\alpha^{3}+\alpha^{21}$	$\alpha^{101}=\alpha^{11}+\alpha^{20}+\alpha^{29}$	$\alpha^{127}=\alpha+\alpha^{28}$
$\alpha^{50}=\alpha^{14}+\alpha^{23}$	$\alpha^{76}=\alpha^{4}+\alpha^{22}$	$\alpha^{102}=\alpha^{12}+\alpha^{21}+\alpha^{30}$	$\alpha^{128}=\alpha^{2}+\alpha^{29}$
$\alpha^{51}=\alpha^{15}+\alpha^{24}$	$\alpha^{77}=\alpha^{5}+\alpha^{23}$	$\alpha^{103}=\alpha^{13}+\alpha^{22}+\alpha^{31}$	$\alpha^{129}=\alpha^{3}+\alpha^{30}$
$\alpha^{52}=\alpha^{16}+\alpha^{25}$	$\alpha^{78}=\alpha^{6}+\alpha^{24}$	$\alpha^{104}=\alpha^{14}+\alpha^{23}+\alpha^{32}$	$\alpha^{130}=\alpha^{4}+\alpha^{31}$
$\alpha^{53}=\alpha^{17}+\alpha^{26}$	$\alpha^{79}=\alpha^{7}+\alpha^{25}$	$\alpha^{105}=\alpha^{15}+\alpha^{24}+\alpha^{33}$	$\alpha^{131}=\alpha^{5}+\alpha^{32}$
$\alpha^{54}=\alpha^{18}+\alpha^{27}$	$\alpha^{80}=\alpha^{8}+\alpha^{26}$	$\alpha^{106}=\alpha^{16}+\alpha^{25}+\alpha^{34}$	$\alpha^{132}=\alpha^{6}+\alpha^{33}$
$\alpha^{55}=\alpha^{19}+\alpha^{28}$	$\alpha^{81}=\alpha^{9}+\alpha^{27}$	$\alpha^{107}=\alpha^{17}+\alpha^{26}+\alpha^{35}$	$\alpha^{133}=\alpha^{7}+\alpha^{34}$
$\alpha^{56}=\alpha^{20}+\alpha^{29}$	$\alpha^{82}=\alpha^{10}+\alpha^{28}$	$\alpha^{108}=1+\alpha^{9}+\alpha^{18}+\alpha^{27}$	$\alpha^{134}=\alpha^{8}+\alpha^{35}$
$\alpha^{57}=\alpha^{21}+\alpha^{30}$	$\alpha^{83}=\alpha^{11}+\alpha^{29}$	$\alpha^{109}=\alpha+\alpha^{10}+\alpha^{19}+\alpha^{28}$	$\alpha^{135}=1$
$\alpha^{58}=\alpha^{22}+\alpha^{31}$	$\alpha^{84}=\alpha^{12}+\alpha^{30}$	$\alpha^{110}=\alpha^{2}+\alpha^{11}+\alpha^{20}+\alpha^{29}$	
$\alpha^{59}=\alpha^{23}+\alpha^{32}$	$\alpha^{85}=\alpha^{13}+\alpha^{31}$	$\alpha^{111}=\alpha^{3}+\alpha^{12}+\alpha^{21}+\alpha^{30}$	
$\alpha^{60}=\alpha^{24}+\alpha^{33}$	$\alpha^{86}=\alpha^{14}+\alpha^{32}$	$\alpha^{112}=\alpha^{4}+\alpha^{13}+\alpha^{22}+\alpha^{31}$	
$\alpha^{61}=\alpha^{25}+\alpha^{34}$	$\alpha^{87}=\alpha^{15}+\alpha^{33}$	$\alpha^{113}=\alpha^{5}+\alpha^{14}+\alpha^{23}+\alpha^{32}$	

$$\begin{aligned} g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))(m_9(x^{\frac{2}{9}}))(m_{15}(x^{\frac{2}{9}}))(m_{21}(x^{\frac{2}{9}})) \\ &= (x^{\frac{2}{9}})^{124}+(x^{\frac{2}{9}})^{121}+(x^{\frac{2}{9}})^{120}+(x^{\frac{2}{9}})^{109}+(x^{\frac{2}{9}})^{106}+(x^{\frac{2}{9}})^{105}+(x^{\frac{2}{9}})^{94}+(x^{\frac{2}{9}})^{91} \\ &\quad +(x^{\frac{2}{9}})^{90}+(x^{\frac{2}{9}})^{79}+(x^{\frac{2}{9}})^{76}+(x^{\frac{2}{9}})^{75}+(x^{\frac{2}{9}})^{64}+(x^{\frac{2}{9}})^{61}+(x^{\frac{2}{9}})^{60}+(x^{\frac{2}{9}})^{49} \\ &\quad +(x^{\frac{2}{9}})^{46}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{34}+(x^{\frac{2}{9}})^{31}+(x^{\frac{2}{9}})^{30}+(x^{\frac{2}{9}})^{19}+(x^{\frac{2}{9}})^{16}+(x^{\frac{2}{9}})^{15} \\ &\quad +(x^{\frac{2}{9}})^{4}+(x^{\frac{2}{9}})+1. \end{aligned}$$

$$\begin{aligned} g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))(m_9(x^{\frac{2}{9}}))(m_{15}(x^{\frac{2}{9}}))(m_{21}(x^{\frac{2}{9}}))(m_{27}(x^{\frac{2}{9}})) \\ &= (x^{\frac{2}{9}})^{128}+(x^{\frac{2}{9}})^{127}+(x^{\frac{2}{9}})^{126}+(x^{\frac{2}{9}})^{124}+(x^{\frac{2}{9}})^{120}+(x^{\frac{2}{9}})^{113}+(x^{\frac{2}{9}})^{112}+(x^{\frac{2}{9}})^{111} \\ &\quad +(x^{\frac{2}{9}})^{109}+(x^{\frac{2}{9}})^{105}+(x^{\frac{2}{9}})^{98}+(x^{\frac{2}{9}})^{97}+(x^{\frac{2}{9}})^{96}+(x^{\frac{2}{9}})^{94}+(x^{\frac{2}{9}})^{90}+(x^{\frac{2}{9}})^{83}+(x^{\frac{2}{9}})^{82} \\ &\quad +(x^{\frac{2}{9}})^{81}+(x^{\frac{2}{9}})^{79}+(x^{\frac{2}{9}})^{75}+(x^{\frac{2}{9}})^{68}+(x^{\frac{2}{9}})^{67}+(x^{\frac{2}{9}})^{66}+(x^{\frac{2}{9}})^{64}+(x^{\frac{2}{9}})^{60}+(x^{\frac{2}{9}})^{53} \\ &\quad +(x^{\frac{2}{9}})^{52}+(x^{\frac{2}{9}})^{51}+(x^{\frac{2}{9}})^{49}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{38}+(x^{\frac{2}{9}})^{37}+(x^{\frac{2}{9}})^{36}+(x^{\frac{2}{9}})^{34}+(x^{\frac{2}{9}})^{30} \\ &\quad +(x^{\frac{2}{9}})^{23}+(x^{\frac{2}{9}})^{22}+(x^{\frac{2}{9}})^{21}+(x^{\frac{2}{9}})^{19}+(x^{\frac{2}{9}})^{15}+(x^{\frac{2}{9}})^{8}+(x^{\frac{2}{9}})^{7}+(x^{\frac{2}{9}})^{6}+(x^{\frac{2}{9}})^{4}+1. \end{aligned}$$

$$\begin{aligned} g(x^{\frac{2}{9}}) &= (m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))(m_9(x^{\frac{2}{9}}))(m_{15}(x^{\frac{2}{9}}))(m_{21}(x^{\frac{2}{9}} \\ &\quad))(m_{27}(x^{\frac{2}{9}}))(m_{45}(x^{\frac{2}{9}}))=(x^{\frac{2}{9}})^{130}+(x^{\frac{2}{9}})^{128}+(x^{\frac{2}{9}})^{125}+(x^{\frac{2}{9}})^{124}+(x^{\frac{2}{9}})^{122} \\ &\quad +(x^{\frac{2}{9}})^{121}+(x^{\frac{2}{9}})^{120}+(x^{\frac{2}{9}})^{115}+(x^{\frac{2}{9}})^{113}+(x^{\frac{2}{9}})^{110}+(x^{\frac{2}{9}})^{109}+(x^{\frac{2}{9}})^{107} \\ &\quad +(x^{\frac{2}{9}})^{106}+(x^{\frac{2}{9}})^{105}+(x^{\frac{2}{9}})^{100}+(x^{\frac{2}{9}})^{98}+(x^{\frac{2}{9}})^{95}+(x^{\frac{2}{9}})^{94}+(x^{\frac{2}{9}})^{92} \\ &\quad +(x^{\frac{2}{9}})^{91}+(x^{\frac{2}{9}})^{90}+(x^{\frac{2}{9}})^{85}+(x^{\frac{2}{9}})^{83}+(x^{\frac{2}{9}})^{80}+(x^{\frac{2}{9}})^{79}+(x^{\frac{2}{9}})^{77}+(x^{\frac{2}{9}})^{76} \\ &\quad +(x^{\frac{2}{9}})^{75}+(x^{\frac{2}{9}})^{70}+(x^{\frac{2}{9}})^{68}+(x^{\frac{2}{9}})^{65}+(x^{\frac{2}{9}})^{64}+(x^{\frac{2}{9}})^{62}+(x^{\frac{2}{9}})^{61}+(x^{\frac{2}{9}})^{60} \\ &\quad +(x^{\frac{2}{9}})^{55}+(x^{\frac{2}{9}})^{53}+(x^{\frac{2}{9}})^{50}+(x^{\frac{2}{9}})^{49}+(x^{\frac{2}{9}})^{47}+(x^{\frac{2}{9}})^{46}+(x^{\frac{2}{9}})^{45}+(x^{\frac{2}{9}})^{40} \\ &\quad +(x^{\frac{2}{9}})^{38}+(x^{\frac{2}{9}})^{35}+(x^{\frac{2}{9}})^{34}+(x^{\frac{2}{9}})^{32}+(x^{\frac{2}{9}})^{31}+(x^{\frac{2}{9}})^{30}+(x^{\frac{2}{9}})^{25} \\ &\quad +(x^{\frac{2}{9}})^{23}+(x^{\frac{2}{9}})^{20}+(x^{\frac{2}{9}})^{19}+(x^{\frac{2}{9}})^{17}+(x^{\frac{2}{9}})^{16}+(x^{\frac{2}{9}})^{15}+(x^{\frac{2}{9}})^{10}+(x^{\frac{2}{9}})^{8} \\ &\quad +(x^{\frac{2}{9}})^{5}+(x^{\frac{2}{9}})^{4}+(x^{\frac{2}{9}})^{2}+(x^{\frac{2}{9}})+1. \end{aligned}$$

$$\begin{aligned} g(x^{\frac{2}{9}}) &= m_1(x^{\frac{2}{9}}))(m_3(x^{\frac{2}{9}}))(m_5(x^{\frac{2}{9}}))(m_7(x^{\frac{2}{9}}))(m_9(x^{\frac{2}{9}}))(m_{15}(x^{\frac{2}{9}})) \\ &\quad (m_{21}(x^{\frac{2}{9}}))(m_{27}(x^{\frac{2}{9}}))(m_{45}(x^{\frac{2}{9}}))(m_{63}(x^{\frac{2}{9}})) \\ &= (x^{\frac{2}{9}})^{134}+(x^{\frac{2}{9}})^{133}+\cdots+(x^{\frac{2}{9}})^{2}+(x^{\frac{2}{9}})+1. \end{aligned}$$

Finally, errors like $3,4,7,10,13,22,31$ *and* 67 *are corrected.*

From Example 2.1 and [3, Example 1], it follows that the code generated through $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ corrects more errors and has better code rate than the code generated through $\mathbb{F}_2[x;a\mathbb{Z}_0]$. Now, we are in position to develop a link between a primitive $(n,n-r)$ binary BCH code C_n and a non-primitive (b^jn,b^jn-r_j) binary BCH code C_{b^jn}, where r and r_j are, respectively, the degrees of their generating polynomials $g(x^a)$ and $g(x^{\frac{a}{b^j}})$. From Theorem 2, it follows that the generalized polynomial $g(x^{\frac{a}{b^j}})$ in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$ divides $(x^{\frac{a}{b^j}})^{b^jn}-1$ in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]$. So, there is a non-primitive BCH code C_{b^jn} generated by $g(x^{\frac{a}{b^j}})$ in $\mathbb{F}_2[x;\frac{a}{b^j}\mathbb{Z}_0]_{b^jn}$. By the same argument, as b^jn divides

$n_j = 2^{b^j s} - 1$, it follows that $(x^{\frac{a}{b^j}})^{b^j n} - 1$ divides $(x^{\frac{a}{b^j}})^{n_j} - 1$ in $\mathbb{F}_2[x; \frac{a}{b^j}\mathbb{Z}_0]$. Therefore, $((x^{\frac{a}{b^j}})^{n_j} - 1) \subset ((x^{\frac{a}{b^j}})^{b^j n} - 1)$. Consequently, from third isomorphism theorem for rings, it follows that

$$\frac{\mathbb{F}_2[x; \frac{a}{b^j}\mathbb{Z}_0]/((x^{\frac{a}{b^j}})^{n_j} - 1)}{((x^{\frac{a}{b^j}})^{b^j n} - 1)/((x^{\frac{a}{b^j}})^{n_j} - 1)} \simeq \frac{\mathbb{F}_2[x; \frac{a}{b^j}\mathbb{Z}_0]}{((x^{\frac{a}{b^j}})^{b^j n} - 1)} \simeq \frac{\mathbb{F}_2[x; a\mathbb{Z}_0]}{((x^a)^n - 1)}.$$

Thus, there are embeddings $C_n \hookrightarrow C_{b^j n} \hookrightarrow C_{n_j}$ of codes, whereas $C_n, C_{b^j n}, C_{n_j}$ are, respectively, primitive BCH, non-primitive BCH and primitive BCH codes. Whereas the embeddings $C_n \hookrightarrow C_{b^j n}$ are defined as $a(x^a) = a_0 + a_1(x^a) + \cdots + a_{n-1}(x^a)^{n-1} \mapsto a_0 + a_1(x^{\frac{a}{b^j}})^{b^j} + \cdots + a_{n-1}(x^{\frac{a}{b^j}})^{b^j(n-1)} = a(x^{\frac{a}{b^j}})$, where $a(x^a) \in C_n$ and $a(x^{\frac{a}{b^j}}) \in C_{b^j n}$. Also, if $g(x^{\frac{a}{b^{j-1}}})$ is the generator polynomial of a binary non-primitive BCH code $C_{b^{j-1}n}^{j-1}$ in $\mathbb{F}_2[x; \frac{a}{b^{j-1}}\mathbb{Z}_0]_{b^{j-1}n}$, then $g(x^{\frac{a}{b^j}})$ is the generator polynomial of a binary non-primitive BCH code $C_{b^j n}$ in the monoid ring $\mathbb{F}_2[x; \frac{a}{b^j}\mathbb{Z}_0]_{b^j n}$. Thus, a non-primitive BCH code $C_{b^{j-1}n}$ is embedded in a non-primitive BCH code $C_{b^j n}$ under the monomorphism defined as $a(x^{\frac{a}{b^{j-1}}}) \mapsto a(x^{\frac{a}{b^j}})$.

With the above discussion it follows the following theorem.

Theorem 3. *[3, Theorem 6] Let C_n be a primitive binary BCH code of length $n = 2^s - 1$ generated by a polynomial $g(x^a)$ of degree r in $\mathbb{F}_2[x; a\mathbb{Z}_0]$.*

1. *There exists a binary non-primitive BCH code $C_{b^j n}$ of length $b^j n$ generated by a polynomial $g(x^{\frac{a}{b^j}})$ of degree $b^j r$ in $\mathbb{F}_2[x; \frac{a}{b^j}\mathbb{Z}_0]$.*
2. *The binary primitive BCH code C_n is embedded in a binary non-primitive BCH code $C_{b^j n}$, for each $j \geq 1$.*
3. *The binary BCH codes of the sequence $\{C_{b^j n}\}_{j \geq 1}$ have the following embedding*

$$C_{bn} \hookrightarrow C_{b^2 n} \hookrightarrow \cdots \hookrightarrow C_{b^j n} \hookrightarrow \cdots$$

Hence,

$$\begin{array}{ccccccccc}
\mathbb{F}_2[x; a\mathbb{Z}_0] & \subset & \mathbb{F}_2[x; \frac{a}{b}\mathbb{Z}_0] & \subset & \mathbb{F}_2[x; \frac{a}{b^2}\mathbb{Z}_0] & \subset & \cdots & \mathbb{F}_2[x; \frac{a}{b^j}\mathbb{Z}_0] \\
\frac{\mathbb{F}_2[x; a\mathbb{Z}_0]}{((x^a)^n - 1)} & \backsimeq & \frac{\mathbb{F}_2[x; \frac{a}{b}\mathbb{Z}_0]}{((x^{\frac{a}{b}})^{bn} - 1)} & \backsimeq & \frac{\mathbb{F}_2[x; \frac{a}{b^2}\mathbb{Z}_0]}{((x^{\frac{a}{b^2}})^{b^2 n} - 1)} & \backsimeq & \cdots & \frac{\mathbb{F}_2[x; \frac{a}{b^2}\mathbb{Z}_0]}{((x^{\frac{a}{b^j}})^{b^j n} - 1)} \\
\cup & & \cup & & \cup & & \cdots & \cup \\
C_n & \hookrightarrow & C_{bn} & \hookrightarrow & C_{b^2 n} & \hookrightarrow & \cdots & C_{b^j n}
\end{array}$$

and a non-primitive BCH code $C_{b^{j-1}n}$ is embedded in a non-primitive BCH code $C_{b^j n}$ under the monomorphism defined by $a(x^{\frac{a}{b^{j-1}}}) \mapsto a(x^{\frac{a}{b^j}})$.

Remark 4. *The polynomial $g(x^{\frac{a}{b}})$ can be obtained from $g(x^{\frac{a}{b^j}})$ by substituting $x^{\frac{a}{b^j}} = y$ and replacing y by $y^{b^j - 1} = x^{\frac{a}{b}}$.*

Example 2.2. *By [3, Example 2] and Example 2.1, it follows that the BCH codes with designed distance $d = 2, 3, \cdots, 9$ have generator polynomials $g(x^{\frac{2}{3}})$, $g(x^{\frac{2}{9}})$ with the same minimum distance, error correction capability and code rate, where the differences are degree, data bits, code*

length and check sum of the code C_{135} which is three times that of code C_{45} and nine times of the code C_{15}. Similarly, on letting $(x^{\frac{2}{9}}) = y$, that is $x^{\frac{2}{3}} = y^3$, we get

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^{106} + (x^{\frac{2}{9}})^{103} + (x^{\frac{2}{9}})^{102} + (x^{\frac{2}{9}})^{97} + (x^{\frac{2}{9}})^{93} + (x^{\frac{2}{9}})^{91} + (x^{\frac{2}{9}})^{90} + \\
&\quad (x^{\frac{2}{9}})^{61} + (x^{\frac{2}{9}})^{58} + (x^{\frac{2}{9}})^{57} + (x^{\frac{2}{9}})^{52} + (x^{\frac{2}{9}})^{48} + (x^{\frac{2}{9}})^{46} + (x^{\frac{2}{9}})^{45} + (x^{\frac{2}{9}})^{16} + \\
&\quad (x^{\frac{2}{9}})^{13} + (x^{\frac{2}{9}})^{12} + (x^{\frac{2}{9}})^{7} + (x^{\frac{2}{9}})^{3} + (x^{\frac{2}{9}}) + 1, \\
g(y) &= (y)^{106} + (y)^{103} + (y)^{102} + (y)^{97} + (y)^{93} + (y)^{91} + (y)^{90} + (y)^{61} + (y)^{58} + \\
&\quad (y)^{57} + (y)^{52} + (y)^{48} + (y)^{46} + (y)^{45} + (y)^{16} + (y)^{13} + (y)^{12} + (y)^{7} + \\
&\quad (y)^{3} + y + 1, \\
g(y^3) &= (y^3)^{106} + (y^3)^{103} + (y^3)^{102} + (y^3)^{97} + (y^3)^{93} + (y^3)^{91} + (y^3)^{90} + (y^3)^{61} + \\
&\quad (y^3)^{58} + (y^3)^{57} + (y^3)^{52} + (y^3)^{48} + (y^3)^{46} + (y^3)^{45} + (y^3)^{16} + (y^3)^{13} + \\
&\quad (y^3)^{12} + (y^3)^{7} + (y^3)^{3} + (y^3) + 1, \\
g(x^{\frac{2}{3}}) &= (x^{\frac{2}{3}})^{106} + (x^{\frac{2}{3}})^{103} + (x^{\frac{2}{3}})^{102} + (x^{\frac{2}{3}})^{97} + (x^{\frac{2}{3}})^{93} + (x^{\frac{2}{3}})^{91} + (x^{\frac{2}{3}})^{90} + (x^{\frac{2}{3}})^{61} \\
&\quad + (x^{\frac{2}{3}})^{58} + (x^{\frac{2}{3}})^{57} + (x^{\frac{2}{3}})^{52} + (x^{\frac{2}{3}})^{48} + (x^{\frac{2}{3}})^{46} + (x^{\frac{2}{3}})^{45} + (x^{\frac{2}{3}})^{16} + (x^{\frac{2}{3}})^{13} \\
&\quad + (x^{\frac{2}{3}})^{12} + (x^{\frac{2}{3}})^{7} + (x^{\frac{2}{3}})^{3} + (x^{\frac{2}{3}}) + 1 \\
&= (x^{\frac{2}{3}})^{16} + (x^{\frac{2}{3}})^{13} + (x^{\frac{2}{3}})^{12} + (x^{\frac{2}{3}})^{7} + (x^{\frac{2}{3}})^{3} + (x^{\frac{2}{3}}) + 1 \in \mathbb{F}_2[x; \frac{2}{3}\mathbb{Z}_0]_{45}.
\end{aligned}$$

Similarly, for

$$\begin{aligned}
g(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^{112} + (x^{\frac{2}{9}})^{108} + (x^{\frac{2}{9}})^{105} + (x^{\frac{2}{9}})^{102} + (x^{\frac{2}{9}})^{100} + (x^{\frac{2}{9}})^{99} + (x^{\frac{2}{9}})^{94} + \\
&\quad (x^{\frac{2}{9}})^{91} + (x^{\frac{2}{9}})^{90} + (x^{\frac{2}{9}})^{67} + (x^{\frac{2}{9}})^{63} + (x^{\frac{2}{9}})^{60} + (x^{\frac{2}{9}})^{57} + (x^{\frac{2}{9}})^{55} + \\
&\quad (x^{\frac{2}{9}})^{54} + (x^{\frac{2}{9}})^{49} + (x^{\frac{2}{9}})^{46} + (x^{\frac{2}{9}})^{45} + (x^{\frac{2}{9}})^{22} + (x^{\frac{2}{9}})^{18} + (x^{\frac{2}{9}})^{15} + \\
&\quad (x^{\frac{2}{9}})^{12} + (x^{\frac{2}{9}})^{10} + (x^{\frac{2}{9}})^{9} + (x^{\frac{2}{9}})^{4} + (x^{\frac{2}{9}}) + 1,
\end{aligned}$$

it follows that

$$g(x^{\frac{2}{3}}) = (x^{\frac{2}{3}})^{22} + (x^{\frac{2}{3}})^{18} + (x^{\frac{2}{3}})^{15} + (x^{\frac{2}{3}})^{12} + (x^{\frac{2}{3}})^{10} + (x^{\frac{2}{3}})^{9} + (x^{\frac{2}{3}})^{4} + (x^{\frac{2}{3}}) + 1,$$

which is the generating polynomial of a BCH code $(45, 23)$ having design distance $d_2 = 6, 7$. In this way. we can obtain a non-primitive binary BCH code C_{15} from a non-primitive binary BCH code C_{45}. On writing the corresponding code vectors of the generating polynomials

$$\begin{aligned}
g(x^{\frac{2}{3}}) &= (x^{\frac{2}{3}})^{16} + (x^{\frac{2}{3}})^{13} + (x^{\frac{2}{3}})^{12} + (x^{\frac{2}{3}})^{7} + (x^{\frac{2}{3}})^{3} + (x^{\frac{2}{3}}) + 1 \\
g(x^{\frac{2}{9}}) &= (x^{\frac{2}{9}})^{106} + (x^{\frac{2}{9}})^{103} + (x^{\frac{2}{9}})^{102} + (x^{\frac{2}{9}})^{97} + (x^{\frac{2}{9}})^{93} + (x^{\frac{2}{9}})^{91} + \\
&\quad (x^{\frac{2}{9}})^{90} + (x^{\frac{2}{9}})^{61} + (x^{\frac{2}{9}})^{58} + (x^{\frac{2}{9}})^{57} + (x^{\frac{2}{9}})^{52} + (x^{\frac{2}{9}})^{48} + (x^{\frac{2}{9}})^{46} + \\
&\quad (x^{\frac{2}{9}})^{45} + (x^{\frac{2}{9}})^{16} + (x^{\frac{2}{9}})^{13} + (x^{\frac{2}{9}})^{12} + (x^{\frac{2}{9}})^{7} + (x^{\frac{2}{9}})^{3} + (x^{\frac{2}{9}}) + 1,
\end{aligned}$$

it follows that

$$\begin{aligned}
v^1 &= (11010001000011001) \\
v^2 &= (11010001000011001000000000000000000000000000011010001 \\
&\quad 00001100100000000000000000000000000011010001000011001)
\end{aligned}$$

The binary bits of v^1 are properly overlapped on the bits of v^2, in fact, they are repeated three times after a particular pattern. Hence, the generating matrix G_2 of $g(x^{\frac{2}{9}})$ contains the generating matrix G_1 of $g(x^{\frac{2}{3}})$ such that $G_2 = \oplus_1^3 G_1$.

3 ALGORITHM

In this section, we propose an algorithm to calculate a non-primitive BCH code of length $b^j n$ using a primitive BCH code of length n, where the encoding and decoding of the code are carried out in Matlab. The process of developing algorithm is divided into two major steps, i.e., encoding and decoding of a non-primitive BCH code of length $b^j n$.

3.1 Encoding of a non-primitive BCH code of length $b^j n$.

In encoding, we first calculate a primitive polynomial of degree s by invoking Matlab's built in command "*bchgenpoly*". After this operation, a non-primitive polynomial of degree bs is calculated. With the help of a root, say α', the elements of the Galois field $GF(2^{bn})$ are calculated such that $(\alpha')^{b^j n} = 1$ and after we obtain the minimal polynomials of these elements. Finally, we get a non-primitive polynomial with the help of these minimal polynomials. Thus, these design distances are calculated through which the number of errors that can be corrected in each BCH code are determined. Some methods are developed in order to achieve a specified result. In Table 3, we show a list of these methods and its description.

Table 3: Encoding modulation description.

	Module	Input	Output
a	Elements of Galois Field	$b*n$	$\alpha'[i]$
b	Bchgenpoly	n,k	$g[i]$
c	Cyclotomic_cosets	$\alpha'[i]$, b, k	$c[i]$
d	Design_distance	$c[i]$	d
e	error	d	t

The steps of the algorithm are explained as follows:

a. Elements of Galois field (alpha array) $\alpha'[i]$).

This step calculates all the elements of the Galois field $GF(2^{bn})$ using a root, say α', of a non-primitive polynomial of degree bs such that $(\alpha')^{bn} = 1$. The element α' is called of alpha array and denoted by $\alpha'[i]$. Now, denote the index array by $A[index]$. For a given input $b*n$, the non-primitive polynomial of degree bs gives the first element of the array $\alpha'[i]$. By increasing its power, each element of the array $\alpha'[i]$ is calculated in outer loop. Then in nested while loop their corresponding values are determined in such a way that if the value of any element in the array exceed bs, we take the remainder $rem(index, bs)$ (step 9 of the algorithm). The loop breaks when the condition for identity is met.

Algorithm 1:

```
1  begin
2      INPUT b and n
3      bn ← b ∗ n
4      Initialize A[index] from 1 to bn − 1
5      Initialize α′[i] ← 0
6      Initialize index ← 1
7      WHILE index ≠ 0
8        mark A[index] ← 0
9        Initialize v ← index
10       Calculate next candidate, p_i, by rem(index, bs)
11       α′[i] ← v
12       WHILE current position of A[index] = 0
13         IF current position < A[index] size
14           i++
15         ELSEIF
16           i ← 0
17           BREAK
18         ENDIF
19       ENDWHILE
20     ENDWHILE
21 end
```

b. BCH generator polynomial ($g[i]$).

The Matlab builds a function and its complete documentation is found under http://www.mathworks.com/help/comm/ref/bchgenpoly.html. In this module, we get the generator polynomial of primitive BCH code of length n.

1 INPUT n and k

2 OUTPUT $g[i]$

c. Cyclotomic cosets ($c[i]$).

Given $\alpha'[i]$, dimension of code k and positive integer b, cyclotomic cosets $c[i]$ are calculated. Length of $c[i]$ is initialized to at max $b*k$, in short all elements should not exceed the max length. Loop start from 2 to max length and calculate unique values in given $\alpha'[i]$. The process stops when we get sum of two elements $= 2$. Once cyclotomic cosets are calculated, we can calculate the minimal polynomials for BCH codes.

d. Non-primitive generating polynomial $g'[i]$.

Given primitive polynomial $p[i]$ and b. First find the highest degree of $p[i]$ i.e., s and initial degree of non-primitive polynomial $p'[i]$ i.e., bs. Initialize each element of $p'[i]$ of length bs to 0. Iterate loop from 2 to bs in order to initialize coefficient array to 1. Finally, iterate each element if coeft_array and modify the value if $p'[i]$ at position i to the value of coeft_array at i. Finally insert 1 at position 0 of $p'[i]$, i.e., when its first element becomes 0. The output of this module

Algorithm 2:

begin
  INPUT $\alpha'[i]$, b, k
  Initialize cal_coset $\leftarrow 0$
  Initialize len_cal_coset $\leftarrow 0$
  Initialize code_length $\leftarrow b*k$
  Initialize len_coset $\leftarrow$ length of $c[i]$
  Initialize code $\leftarrow b*k$
  FOR $i \leftarrow 2$ to len_coset
    cal_coset $\leftarrow c[i]$ at position i
      IF $i \neq 2$ THEN
        code_length $\leftarrow bk$ cal_coset
      ENDIF
    PRINT $c[i]$
  END FOR
end

play an integral role for calculating non-primitive BCH generating polynomial. It is denoted by g' in our algorithm. We are interested in rows of obtained matrix. Using the matrix we obtained, the code for non-primitive generating polynomial of length b. Finally these values are printed out and saved in file for further usage.

Algorithm 3:

begin
  INPUT $p[i]$, b
  Initialize len $\leftarrow$ length of $p[i]$
  Initialize size_array $\leftarrow$ len $\times$ b
  Initialize $p'[i]$ of length size_array i.e. $p\prime[i] \leftarrow 0$
  Initialize index $\leftarrow 1$
  Initialize len_coef_array $\leftarrow 0$
    FOR i taking values from 2 to len
      IF $p'[i]$ at $i \neq 0$ THEN
        Initialize coef_array $\leftarrow b*(i-1)$
        increment index
      ENDIF
    ENDFOR
    len_coef_array $\leftarrow$ length of coef_array
    FOR j taking values from 1 to len_coef_array
      $g\prime(i)$ at position j of coef_array $\leftarrow 1$
    ENDFOR
    $g\prime[i] \leftarrow 1$ to $p\prime[i]$
    PRINT $g\prime[i]$
end

e. Designed distance (d).

Here design distance is calculated from $g'[i]$. The length of coset array cl is determined and then iterate index from 2 to cl, calculate next index ni by increment current index. If $ni \leq cl$, then next element of coset is calculated to the value of coset array at position of next index. Otherwise the bn is assigned to next coset. The whole process iterates to lencoset coset and finally stops at the last coset. Design distance d is calculated from the last value of coset array at position 1.

Algorithm 4:

```
1  begin
2  |       INPUT c[i]
3  |       Initialize lencoset ← cl
4  |       Initialize ni ← 0
5  |       Initialize next_coset ← 0
6  |       FOR index from 2 to cl
7  |         ni ← increment index
8  |         IF ni ≤ cl THEN
9  |           next_coset ← c[i] at position ni
10 |         ELSEIF
11 |               next_coset ← bn
12 |           ENDIF
13 |       ENDFOR
14 |       d ← next_coset at position 1.
15 end
```

f. Error correction capability.

For given designed distance d, the error correction capability of a code t is calculated.

Algorithm 5:

```
1  begin
2  |       INPUT d.
3  |           error t ← (d − 1)/2.
4  end
```

3.2 Error correction in received polynomial (decoding)

In decoding step for a received polynomial, we first calculate the syndrome matrix $S[i]$. Then the $D_$matrix is calculated that should be invertible. After this error locator polynomial is determined whose roots give the exact position of errors in the received polynomial. Finally the received polynomial is corrected. To find the error vector and obtain the corrected codeword following scheme is used. Table 4 shows list of the following steps for error correction.

Table 4: Encoding modulation description.

	Module	Input	Output
a	Syndrome_Matrix	d, bn, bk, α', $r[i]$	$S'[i]$
b	Calculate D_matrix	t, $S'[i]$	D_matrix
c	Is D invertible	t, D_matrix	$\mid D \mid \neq 0$
d	error_locator_poly	t, D_matrix, $S'[i]$	$f[i]$
e	error_position	$f[i]$	$e[i]$
f	error_values	t, $S[i]$, D_matrix, $e[i]$, bn, $pm[i]$	$ev[i]$
h	Correct_recieved polynomial	t, bn, $e[i]$, $ev[i]$, $r[i]$, α'	$v[i]$

a. **Syndrome matrix** $S'[i]$).

Given design distance d, bn, message length bk and received polynomial $r[i]$, syndrome matrix $S'[i]$ can be calculated. The length of $S'[i]$ is initialize to the difference of bn and bk, i.e., $bn-bk$. Furthermore, $S'[i]$ is initialized by element of Galois field at position i i.e. $GF[i]$. Nested loop are used to calculate $S'[i]$. Upper loop is limited to the length of $bn+d-2$, where $S'[i]$ equalizes to power of α' and in nested loop $S'[i]$ equalize to power of $S'[i]$ and the iterator. Once the values of $S'[i]$ are filled with the above values of $S'[i]$, it follows that these $S'[i]$ can be calculated as product of $r[i]$ and $(S'[i])^t$.

Algorithm 6:

```
1  begin
2      INPUT d, bn, bk, α and r[i]
3      Initialize lenSyndrome ← bn − bk
4      Initialize S'[i] ← GF[i] of len Syndrome
5      Initialize valueSynd ← 0
6      Initialize valueSynd ← GF[i] length bn
7      Initialize loopLimit ← bn + d − 2
8      FOR i ← bn to loop limit
9          valueSynd ← α^i
10         FOR j ← 1 to bn
11             evalSynd ← valueSynd powers j
12         ENDFOR
13         S'[i] ← received_poly ∗ transpose of evalSynd
14     ENDFOR
15     PRINT S'[i]
16 end
```

b. D **matrix** ($D[i]$).

Given error t and $S'[i]$, D matrix is calculated and then double loops operation on syndrome matrix is carried out and suitable values from syndrome matrix is scanned out. The process is as

follows: calculate $GF[i]$ of length t and D-matrix is initialized to that value. $D[i]$ in nested loop for loops. Both loops iterates from 1 to t. $D[i]$ values are $S'[i]$ values at position i of sum of loops iterators to 1.

Algorithm 7:

```
1  begin
2      INPUT t, S'[i]
3      Calculate GF[i] of length t
4      Initialize D_matrix ← GF[i]
5      FOR index i1 taking from 1 to t
6          FOR index i2 taking from 1 to t
7              D_matrix ← S'[i] at position i1 + i2 − 1
8          ENDFOR
9      ENDFOR
10 end
```

c. Is D invertible.

This module check if $D[i]$ is invertible. If $D[i]$ is invertible, then the algorithm works, otherwise error t is decremented and D matrix is again calculated. These operations are carried out till error t becomes 0. If t becomes 0, then algorithm will exit i.e panic condition occurs as in step 10.

Algorithm 8:

```
1  begin
2      INPUT t and D[i]
3      IF D[i] is invertible THEN
4          continue algo // goto step 5.
5      ELSEIF
6          decrement t;
7          IF t equals 0
8              goto STEP 3
9          ENDIF
10     ENDIF
11     // Panic Condition
12     IF t equals 0
13     Print ERROR cannot be corrected.
14     EXIT algo
15     ENDIF
16 end
```

d. **Error locator polynomial** ($\mathbf{f}[\mathbf{i}]$).

Given input t, $D[i]$ and $S'[i]$, error locator polynomial $f[i]$ is calculated. First, initialize product matrix $pm[i]$ and $f[i]$ to $\alpha''[i]$. Then, iterate the loop from 1 to t, $pm[i]$ is filled with the value of $S'[i]$ at $t+i$. After the loops ends, the value of $(S'[i] * pm[i])^{-1}$ get equals to temporary matrix

$T''[i]$. Once the temporary matrix $T''[i]$ is achieved $f[i]$ is taken as the transpose of that temporary matrix $(T''[i])^t$ discuss in step 10.

Algorithm 9:

begin
    INPUT t, $D[i]$ and $S[i]$
    Create $\alpha''[i]$
    Initialize $pm[i] \leftarrow \alpha''[i]$
    Initialize $f[i] \leftarrow \alpha''[i]$
    Initialize temporary_matrix $T''[i]$ of size $t \leftarrow 0$
    FOR index i taking values from 1 to t
        $pm[i] \leftarrow S[i]$ at position $t+i$
    ENDFOR
    $T''[i] \leftarrow (S[i])^{-1} * pm[i]$
    $f[i] \leftarrow (T''[i])^t$
    $f[t+1] \leftarrow 1$ // coefficient of $f[t+1] = 1$
end

e. Error position matrix $(e[i])$.

Based on $f[i]$ we can determine error position. First, the roots of $f[i]$ is calculated and then we take its inverse. The values we obtain are in matrix form and these manifest error position.

Algorithm 10:

begin
    INPUT $f[i]$
    Initialize error_pos_matrix $e[i] \leftarrow 0$
    Initialize root_matrix $\leftarrow 0$
    root_matrix $\leftarrow$ roots of $f[i]$
    $e[i] \leftarrow$ inverse of roots of $f[i]$
    PRINT $e[i]$
end

f. Error values $(ev[i])$.

Once error position $e[i]$ is determined we can easily calculate their respective values. The nested for loops are used to determine error values. Both of the loops iterate from 1 to t, in the first loop values from D_matrix can be taken while in the next loop value of product matrix $pm[i]$ can be taken along with $S'[i]$. Finally, $ev[i]$ get equated to $(D\text{\_matrix} * pm[i])^{-1}$ as discusses on line 9.

g. Correct received polynomial $r[i]$.

Once we have calculated error positions $e[i]$ and error values $ev[i]$ the received polynomial $r[i]$ can be corrected. Here input parameters are $e[i], ev[i]$, $r[i]$ and bn. First estimated codeword denoted by est_code is calculated using various operations i.e., taking loops to t, bn and power of

Algorithm 11:

begin
  INPUT t, $S[i]$, $D[i]$, $e[i]$, bn and $pm[i]$
  Initialize $ev[i] \leftarrow 0$
  FOR $1 \leq i1 \leq t$
    FOR $1 \leq i2 \leq t$
      $D[i]$ at $i1$ and $i2 \leftarrow e[i]$ at $i2 * (i1 + bn - 1)$
    ENDFOR
    $pm[i]$ at $i1 \leftarrow S[i1]$
  ENDFOR
  $ev[i] \leftarrow (D\_\text{matrix} * pm[i])^{-1}$
  PRINT $ev[i]$
end

Galois field. The received polynomial is corrected by subtracting error polynomial and we get the corrected codeword $v[i]$.

Algorithm 12:

begin
  INPUT t, bn, $e[i]$, $ev[i]$, $r[i]$ and α'
  calculate $GF[i]$ of length bn.
  Initialize est_error $\leftarrow GF[i]$
  Initialize est_code $\leftarrow GF[i]$
  Initialize alpha_val $\leftarrow 0$
  FOR $1 \leq i \leq t$
    FOR $1 \leq j \leq bn$
      alpha_val $\leftarrow (\alpha')^{j-1}$
      IF alpha_val = element $e[i]$ at i THEN
        est_error position $j \leftarrow$ est_error at position $j + ev[i]$ at i
      ENDIF
    ENDFOR
    est_code $\leftarrow r[i]+$ est_error
    PRINT est_code
    elements of $pm[i]$ at $i1 \leftarrow S'[i]$ elements at $i1$
  ENDFOR
  $ev[i] \leftarrow D\_\text{matrix} * pm[i]$
  PRINT $ev[i]$
end

Example 3.1. *For the code of length* 45 *simulation is carried out as follows: in this case* $b = 3$ *and* $n = 15$*. Using* $n = 15$ *and* $k = 11$*, Matlab's build in function* ***genpoly*** *is invoked in order to find primitive polynomial, i.e.,* $p(i) = x^4 + x + 1$*, as explain in Table 3. With* $b = 3$ *and* $p(i) = x^4 + x + 1$*,* ***non_primitive_poly*** *function is invoked, as described in Table 3 step 4, here non-primitive polynomial named as* $p\prime(i)$ *is obtained. Output for* $p\prime(i)$ *is* $x^{12} + x^9 + 1$*. Cyclotomic cosets, i.e.,* ***coset_array*** *values are also calculated. First elements of Galois field in step 1 of*

Table 1, is invoked to find the power of alpha till $\alpha^{45} = 1$. *With coset_array in hand, the designed distance d can be calculated, which is the first element of next coset_array. Last but not the least error t is calculated against the given designed distance d. Code rate R is also calculated against each* k_1 *and bn but is not mentioned in previous section. The output are as follows: Cyclotomic cosets for* $(45,33) = [1\ 2\ 4\ 8\ 16\ 32\ 19\ 38\ 31\ 17\ 34\ 23]$, $t_1 = 1$ *and* $R_1 = (0.73333)$. *Cyclotomic cosets for* $(45,29) = [3\ 6\ 12\ 24]$, $t_1 = 2$ *and* $R_1 = (0.64444)$. *Cyclotomic cosets for* $(45,23) = [5\ 10\ 20\ 40\ 35\ 25]$, $t_1 = 3$ *and* $R_1 = (0.51111)$. *Cyclotomic cosets for* $(45,11) = [7\ 14\ 28\ 11\ 22\ 44\ 43\ 41\ 37\ 29\ 13\ 26]$, $t_1 = 4$ *and* $R_1 = (0.24444)$. *Cyclotomic cosets for* $(45,5) = [15\ 30]$, $t_1 = 10$ *and* $R_1 = (0.11111)$. *Now comes error correction in received polynomial. In this the code* $(45,29)$ *is taken under consideration, with designed distance* $d_1 = 5$ *and* $t_1 = 2$. *Let the received polynomial be*

$$x^{44} + x^{16} + x^{13} + x^{12} + x^{11} + x^7 + x^3 + x + 1.$$

With the given values d_1, k_1, *and received polynomial, syndrome_matrix is calculated. The output for syndromes are* $S_1 = \alpha^2$, $S_2 = \alpha^4$, $S_3 = \alpha^{30}$, $S_4 = \alpha^8$. *Next, we arrange syndrome values in linear equation form that is* $Ax = B$. *Where* $A = [S_1,\ S_2;\ S_2,\ S_3]$ *and* $B = [S_3,\ S_4]$. *Matrix A is named as t_matrix of* $t \times t$ *dimension. Thus, we find the whether the t_matrix is singular or not. If the determinant of t_matrix is nonzero then error locator polynomial is calculated. Next error position is calculated from sigma_matrix which is obtained from the coefficients of error locator polynomial. For the given values, error_positions are* 44 *and* 11. *Hence the error polynomial is* $x^{44} + x^{11}$. *On subtracting error polynomial from received polynomial the following code polynomial*

$$x^{16} + x^{13} + x^{12} + x^7 + x^3 + x + 1$$

is obtained. All of the above equations are obtained by using Matlab symbolic toolbox.

With the help of the above discussed algorithm many examples on non-primitive BCH codes of length bn, b^2n, b^3n are constructed corresponding to primitive BCH code of length n. The parameters for all binary non-primitive BCH codes of length bn, b^2n, b^3n, where $n \leq 2^6 - 1$ and b is either 3 or 7 are given in Table 5.

Table 5 manifests the error and code rate values against some selected codes which we have obtained after simulating our algorithm. These codes are of length bn, b^2n and b^3n, where $n \leq 2^6 - 1$, and $b = 3, 7$. k_1, k_2 and k_3 are dimensions of the codes C_{bn}, C_{b^2n} and C_{b^3n}, respectively.

Interleaved Codes

From Table 5, it is observed that corresponding to a primitive (n,k) code there are (bn,bk), (b^2n,b^2k_2), $(b^3n,\ b^3k)$ codes with same error correction capability and code rate. These codes are found to be *interleaved codes* (*Interleaving* is a periodic and reversible reordering of codes of L transmitted bits) of depth b, b^2 and b^3. Hence along with random error correction capability these codes can correct burst of error of length b, b^2 and b^3, respectively. The term burst of error means that two or more bits in the received word has changed from 1 to 0 or from 0 to 1. The length of the burst is measured from the first corrupted bit to the last corrupted bit. Similarly, for the code (bn,bk) the codes (b^2n,b^2k) and (b^3n,b^3k) are interleaved codes of depth b^2 and b^3,

Table 5: Parameters of binary non-primitive BCH codes.

n	**bn**	$\mathbf{k}_1$	$\mathbf{t}_1$	$\mathbf{R}_1$	$\mathbf{b^2n}$	$\mathbf{k}_2$	$\mathbf{t}_2$	$\mathbf{R}_2$	$\mathbf{b^3n}$	$\mathbf{k}_3$	$\mathbf{t}_3$	$\mathbf{R}_3$
15	**45**	33	1	0.733	**135**	99	1	0.733	**405**	297	1	0.733
		29	2	0.644		87	2	0.644		261	2	0.644
		23	3	0.511		69	3	0.511		207	3	0.511
		11	4	0.244		33	4	0.244		99	4	0.244
		7	7	0.155		29	7	0.214		87	7	0.214
		5	10	0.111		23	10	0.170		69	10	0.170
		1	22	0.022		11	13	0.081		33	13	0.081
						7	22	0.051		29	22	0.071
						5	31	0.037		23	31	0.056
						1	67	0.007		11	40	0.027
										7	67	0.017
										5	94	0.012
										1	202	0.002
63	**189**	171	1	0.904	**567**	513	1	0.904	**1701**	1539	1	0.904
		165	2	0.873		495	2	0.873		1485	2	0.873
		147	3	0.777		441	3	0.777		1323	3	0.777
		129	4	0.682		387	4	0.682		1161	4	0.682
		123	5	0.650		381	5	0.672		1143	5	0.671
		105	6	0.555		327	6	0.576		981	6	0.576
		87	7	0.460		273	7	0.481		819	7	0.481
		81	10	0.428		255	10	0.449		765	10	0.449
63	**189**	75	11	0.3968	**567**	237	11	0.4179	**1701**	711	11	0.417
		57	13	0.3015		183	13	0.3227		549	13	0.322
		54	15	0.2857		177	15	0.3121		543	15	0.319
		36	16	0.1904		123	16	0.2169		381	16	0.224
		30	19	0.1587		105	19	0.1851		327	19	0.192
		24	22	0.1269		87	22	0.1534		273	22	0.160
		18	31	0.0952		81	31	0.1428		255	31	0.149
		16	34	0.0846		75	34	0.1322		237	34	0.1393
		10	40	0.0529		57	40	0.1005		183	40	0.1075
		7	46	0.0370		54	46	0.0952		177	46	0.1040
		1	94	0.0053		36	49	0.0634		123	49	0.0723
						30	58	0.0529		105	58	0.0617
						24	67	0.0423		87	67	0.0511
						18	94	0.0317		81	94	0.0476
						16	103	0.0282		75	103	0.0440
						10	121	0.0176		57	121	0.0335
						7	139	0.0123		54	139	0.0317
						1	283	0.0017		36	148	0.0211
										30	175	0.0176
										10	364	0.0058
										1	850	0.0005

respectively. The code $(49,28)$ is interleaved code of depth 7, which is formed by interleaving the following 7 codewords from $(7,4)$ code that is (0000000), (1101000), (0000000), (0000000), (0000000), (0000000) and (0000000). on writing them column by column it gives

$$v^1 = (0100000010000000000000100000000000000000000000000) \in C_{49}.$$

In a similar way, codeword of $(343,196)$ and $(2401,1372)$ are obtained by writing column by column 7 codeword of C_{49} and C_{343}. Therefore, for decoding a received polynomial in C_{343} one can easily reverse the process and correct errors in the codeword of either C_{49} or C_7.

ACKNOWLEDMENT

Acknowledgment to FAPESP by financial support 2013/25977-7. The authors would like to thank the anonymous reviewers for their intuitive commentary that signicantly improved the worth of this work.

RESUMO. Neste trabalho, apresentamos um método que estabelece como uma sequência de códigos BCH não primitivos pode ser obtida através de um dado código BCH primitivo. Para isso, utilizamos uma técnica de construção diferente da técnica rotineira de códigos BCH e usamos a estrutura de anéis monoidais em vez de anéis de polinômios. Consequentemente, mostramos que existe uma sequência $\{C_{b^j n}\}_{1\leq j\leq m}$, onde $b^j n$ é o comprimento do código $C_{b^j n}$, de códigos BCH binários não primitivos em vez de um dado código binário BCH C_n de comprimento n. Algoritmos simulados via Mathlab para codificação e decodificação para este tipo de códigos são introduzidos. O algoritmo via o Matlab fornece rotinas para a construção de um código BCH primitivo, mas impõe várias restrições, como por exemplo, o grau s de um polinômio irredutível primitivo deve ser menor que 16. Este trabalho trata-se de polinômios não-primitivos irredutíveis com grau bs, que são maiores do que 16.

Palavras-chave: Anel monoidal, códigos BCH, polinômio primitivo, polinômio não-primitivo.

REFERENCES

[1] A.A. Andrade & P. Jr. Linear codes over finite rings. *TEMA – Trends in Applied and Computational Mathematics*, **6**(2) (2005), 207–217.

[2] A.A. Andrade, T. Shah & A. Khan. A note on linear codes over semigroup rings. *TEMA – Trends in Applied and Computational Mathematics*, **12**(2) (2011), 79–89.

[3] A.S. Ansari & T. Shah. An association between primitive and non-primitive BCH codes using monoid rings. *EURASIP - Journal on Wireless Communications and Networking*, **38** (2016).

[4] J. Cazaran, A.V. Kelarev, S.J. Quinn & D. Vertigan. An algorithm for computing the minimum distances of extensions of BCH codes embedded in semigroup rings. *Semigroup Forum*, **73**(3) (2006), 317–329.

[5] A.V. Kelarev. "Ring constructions and applications". World Scientific Publishing Co. Inc., New York (2002).

[6] A.V. Kelarev & P. Solé. Error-correcting codes as ideals in group rings. *Contemporary Mathematics*, **273** (2001), 11–18.

[7] S. Lin & J. D. J. Costello. "Error control coding: fundamentals and applications". Prentice Hall Professional Technical Reference, New York (1994).

[8] S.R. Nagpaul & S.K. Jain. "Topics in applied abstract algebra", The Brooks/Cole Series in Advanced Mathematics. New York (2005).

[9] V.S. Pless, W.C. Huffman & R.A. Brualdi. "Handbook of coding theory". Elsevier, New York (1998).

[10] A. Poli & L. Huguet. "Error-correcting codes: theory and applications". Prentice-Hall, New York (1992).

[11] T. Shah, Amanullah & A.A. de Andrade. A decoding procedure which improves code rate and error corrections. *Journal of Advanced Research in Applied Mathematics*, **4**(4) (2012), 37–50.

[12] T. Shah, Amanullah & A.A. de Andrade. A method for improving the code rate and error correction capability of a cyclic code. *Computational and Applied Mathematics*, **32**(2) (2013), 261–274.

[13] T. Shah & A.A. de Andrade. Cyclic codes through $B[X]$, $B[X;\frac{1}{kp}Z_0]$ and $B[X;\frac{1}{p^k}Z_0]$: a comparison. *Journal of Algebra and its Applications*, **11**(4) (2012), 1250078, 19.

[14] T. Shah & A.A. de Andrade. Cyclic codes through $B[X;\frac{a}{b}\mathbb{Z}_0]$, with $\frac{a}{b} \in \mathbb{Q}^+$ and $b = a+1$, and encoding. *Discrete Mathematics, Algorithms and Applications*, **4**(4) (2012), 1250059, 14.

[15] T. Shah, A. Khan & A.A. Andrade. Encoding through generalized polynomial codes. *Comp. Appl. Math.*, **30**(2) (2011), 349–366.

[16] T. Shah, M. Khan & A.A. de Andrade. A decoding method of an n length binary BCH code through $(n+1)n$ length binary cyclic code. *Anais da Academia Brasileira de Ciências*, **85**(3) (2013), 863–872.

[17] T. Shah & A. Shaheen. Cyclic codes as ideals in $F_2[x;aN_0]_n$, $F_2[x]_{an}$, and $F_2[x;\frac{1}{b}N_0]_{abn}$: a linkage. *U.P.B. Sci. Bull., Series A*, **78**(3) (2016), 205–220.