ЮБИЛЕЙНА НАУЧНА СЕСИЯ – 30 години ФМИ
ПУ "Паисий Хилендарски", Пловдив, 3-4.11.2000

MULTIPLE ERROR CORRECTING ALGORITHM FOR SOFTWARE PROTECTION OF ARCHIVE DATA

Mairtin O'Droma, Ivan Ganchev Ivanov

A self-contained error correcting code (ECC) algorithm is proposed. Based on the use of BCH codes it envisaged as part of an adaptive data protection (ADP) application programme. It is designed to operate (decode and error correct) for a range of BCH codes providing increasing level of protection against error in accordance with user choices. While such codes cause loss of compression the improvement in error probability is dramatic, e.g. from 10^{-5} to10^{-29} approximately for BCH(8,8), with a 20% loss of compression.

Keywords: BCH codes; CD-ROMs; DVD; archives; SKHNE algorithm; error protection

1 Introduction

Coding to protect against errors arising in the electronic archival writing or retrieval process or in the archive itself, fig. 1, is an important field [1]. Driven by user expectations and the potential for frustration, damage and cost in the event of archived data corruption many schemes have been designed to reduce the probability of error, e.g. c.f. [1, 2, 3, 4 and 5]. Some schemes are built into the archiving software. Some of these are error detecting only, e.g. CRC-32 being a popular and powerful one. Others attempt to recover corrupted data e.g. the ARJ program [4] can be made to repair up to 4 damaged 1kbyte long sections in a 1Mbyte block. Other schemes are embedded into the hardware. In CD-ROM and DVD hardware implemented (right of 'A', fig.1), powerful, Reed Solomon product code (RS-PC) and cross-interleaved RS code (CIRC) are used resp., e.g. [5], [6], [7] and [8]. The latter can cater for 6.0mm burst error and both add c. 23% and 13% overhead resp., [9]. However regardless of the in-built error protection power in any archive system, requirements will continue to change as a function of service application, user needs and so forth, e.g. [5], [10]. Thus there is room for an optional, additional, user-friendly, software-based scheme for multiple error protection, the level of protection being chosen by the user and traded against increased storage requirements. Such an adaptive data protection (ADP) application has been developed based on BCH codes[1]. It would be inserted at 'A' in fig. 1 and may include its own compression. The error protection system to the right of 'A' if already present, as part of the recording device (embedded firmware) e.g. DVD, becomes a second layer of protection. This paper will describe the error-correcting algorithm used in this scheme. The algorithm is based on recent developments in the understanding of BCH decoding mechanisms and contains some innovations of its own.

[1] Telecom applications using efficient binary BCH codes have emerged. Examples are the USA TDMA cellular system employing a BCH(48,36,5) scheme and the paging protocol specification POCSAG employing BCH(31,21,5); [in (n,k,d) notation, the latter being equivalent to BCH(5,2) in the (m,t) notation used in this paper]. These are fixed systems as compared to the user-controlled, flexible, multilevel error protection reliability offered by the archival system proposed here.

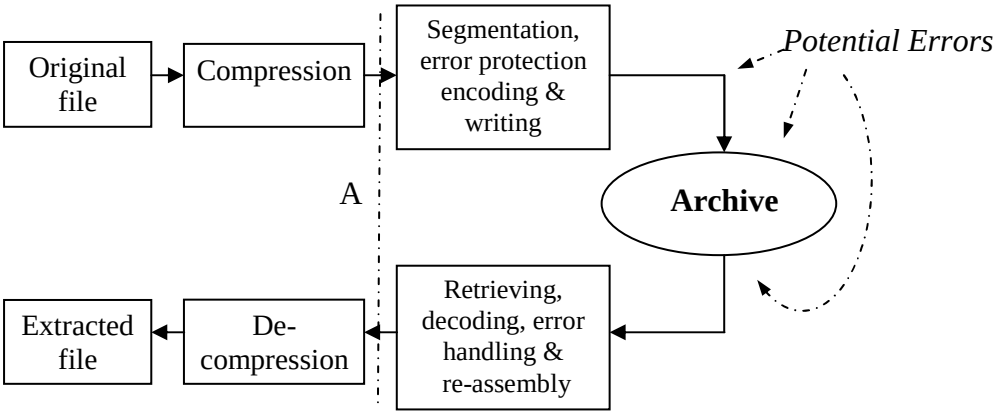

Figure 1. A block diagram of the typical archival-retrieval process

2 A BCH error correction archival system

Before describing the error correcting algorithm some brief comments are presented on BCH encoding.

2.1 BCH Encoding

The Bose-Chaudhuri-Hocqueghen (BCH) codes are a generalization of Hamming codes which allow multiple error correction [11], [12]. They have the attractions for this application of flexibility in the choice of those parameters which dictate their error correcting power. Also at block lengths of a few hundred or less, many of these codes are among the most efficient known, [13]. Releveant background is briefly presented here to facilitate explaining the BCH ECC algorithm.

Since BCH codesets are cyclic linear block (n,k) codes any valid n-tuple member can be represented by a polynomial of degree ***(n-1)*** or less which has a characteristic unique generator polynomial ***G(x)*** as a factor. The polynomial coefficients are required to be elements of a finite Galois field, designated *GF(**q**)* where ***q*** is the order of the field. *GF(**q**)* is a prime field when ***q*** is a prime and is an extension field over a prime field when ***q*** is a power of a prime. In the former, the field has as elements the integers ***0,… , q-1***, and the operations are simply modulo ***q*** multiplication and addition. In the latter (say $q = p^m$ where ***p*** is prime), then the field elements are all possible polynomials of degree ***m-1*** where the coefficients are from the prime field *GF(**p**)*. The number ***p*** is called the characteristic of $GF(p^m)$. Multiplication and addition are executed by multiplying and adding these polynomials in the usual way and then reducing the result modulo a special degree-*m* polynomial ***p(x)*** which cannot be factored using only polynomials with coefficients from *GF(**p**)*.

In all finite fields there exists at least one element α, called a generator or primitive element, such that every other nonzero element in the field can be expressed as a power of this element. A primitive element α in a field *GF(**q**)* can easily be found by simply checking the powers of each field element. For this it must satisfy the following rule: α is a primitive element of *GF(**q**)* iff the powers $\alpha, \alpha^2, \ldots, \alpha^{q-1} \{= \alpha^0\}$ are distinct and produce all the non-zero elements of *GF(**q**)*.

The BCH codes are most easily defined in terms of the roots of the generator polynomial, ***G(x)***. Thus a primitive *t*-error-correcting *BCH(**m,t**)* code over *GF(**p**^{m})* of block length ***n* = (*q*^{m} -1)** has $\alpha^{m_0}, \alpha^{m_0+1}, \ldots, \alpha^{m_0+2t-1}$ as roots of ***G(x)*** for any m_0, where α is a primitive element of *GF(**p**^{m})*. Those codes with $m_0 = 1$ are called "narrow-sense BCH codes" and those defined over *GF(**2**^{m})* are called binary codes.

In designing a binary *BCH(**m,t**)* code, the choice of the primitive element of *GF(**2**^{m})* is immaterial, [14]. Thus, in this respect, the encoder and decoder should use the same primitive element together with matching encoding and decoding techniques. The archiving algorithm proposed for the ADP application uses BCH generator polynomials taken from Clark & Cain, [13].

The encoding process: This follows a straight forward implementation e.g. [14] of the standard systematic *BCH(**m,t**)* cyclic code generation procedure:

$$c(x) = b(x)x^{n-k} + h(x), \quad \text{where } h(x) = \{b(x)\, x^{n-k}\} \text{modulo} \{G(x)\};$$

c(x) and ***b(x)*** represent the ***n*** and ***k*** tuple codeword and message word resp. Letting ***d*** be the code's minimum Hamming distance, the *BCH(**m,t**)* and *BCH(**n,k,d**)* designations of these codes are identical when

$$n = 2^m - 1; \qquad d = 2t+1; \qquad \text{and} \quad k = n - \{\text{degree of } G(x)\}$$

2.2 BCH Decoding and Error Correction

The error correcting decoding process implemented operates on principle of choosing the valid codeword vector, ***c***, which is at the minimum Hamming distance (i.e. MDD) from the retrieved data vector, ***r.*** This MDD choice is also the maximum likelihood, ML, one for equally likely source symbols at the encoding stage.

The BCH decoding algorithm operates by seeking that error vector ***e*** of minimum Hamming weight such that ***c=r+e***. This it does by computing the syndrome (an *n-k* tuple) of the retrieved vector through multiplying it by the parity check matrix. For BCH codes, there are several forms of the syndrome. In the form used here, [14], the syndromes S_i of the retrieved sequence vector ***r, =r$_j$, j=0,…,(n-1)***, are defined as the values

$$S_i = \sum_{j=0}^{n-1} r_j \alpha^{ij}, \qquad i = 1, 2, \ldots, 2t, \tag{1}$$

where α is the primitive element of *GF(**2**^{m})*; it can be shown that $S_{2i} = S_i^2$. If $S_i = 0$ for ***i=1,…, 2t***, then the retrieved sequence is a valid code word, (and hopefully the original code word produced by the BCH encoder). The syndrome polynomial can be written:

$$S(z) = S_1 + S_2 z + S_3 z^2 + \ldots + S_{2t} z^{2t-1} = \sum_{i=0}^{2t-1} S_{i+1} z^i. \tag{2}$$

Error locator Λ(z) and error evaluator Ω(z) polynomials: If the decoder's retrieved sequence has $\nu \le 2t$, errors in error locations $j_1, j_2, \ldots, j_\nu$, then its ***Λ(z)*** and ***Ω(z)*** polynomials can be written [14]:

$$\Lambda(z) = \prod_{k=1}^{\nu} (1 - \alpha^{j_k} z), \text{ and } \Omega(z) = \sum_{k=1}^{\nu} e_{j_k} \alpha^{j_k} \prod_{l=1 \text{and } l \ne k}^{\nu} (1 - \alpha^{i_l} z). \tag{3}$$

For ν errors the degree of ***Λ(z)*** is ν and of ***Ω(z),*** less than ν. When ***t*** or less errors occur in the retrieved sequence, then the following equality holds, [14]:

(4) $$\Lambda(z)S(z) \equiv \Omega(z) \bmod (z^{2t})$$

To pinpoint the bits in error in the retrieved sequence $\boldsymbol{\Lambda(z)}$, as its name implies, must be found. For this eq. 4 is the "key equation" to be solved. The reciprocals of the roots of $\boldsymbol{\Lambda(z)}$ correspond to the error locations. Inverting the retrieved vector symbols (bits) in these locations effects the error correction.

Finding $\Lambda(z)$: Berlekamp [15] was first to develop an algorithm to find the minimum-degree $\boldsymbol{\Lambda(z)}$ which satisfies eq.4. It is quite an efficient one, which Massey [16] later improved. However, the technique used here adapts, and implements, a conceptually simpler technique based on the recently reported Sugiyama-Kasahara-Hirasawa-Namekawa (SKHN) algorithm, [17], which in turn is based on Euclid's algorithm; for brevity it is called here the SKHNE algorithm.

***SKHNE Algorithm**:* Euclid's recursive technique for finding the greatest common divisor of two polynomials (or two integers), can be stated as follows: if $\boldsymbol{a}$ and $\boldsymbol{b}$ are two polynomials where $\boldsymbol{deg(a) \geq deg(b)}$, then $\boldsymbol{b}$ is the greatest common divisor if $\boldsymbol{amod(b) = \eta = 0}$. If $\boldsymbol{\eta \neq 0}$ then $\boldsymbol{a}$ is replaced with $\boldsymbol{b}$ and $\boldsymbol{b}$ with $\boldsymbol{\eta}$, and the operation is repeated until the condition $\boldsymbol{\eta = 0}$ is satisfied. In the process of finding this greatest common divisor, $\boldsymbol{d}$, the algorithm calculates two polynomials $\boldsymbol{f}$ and $\boldsymbol{g}$ such that $\boldsymbol{fa+gb=d}$.

For the purposes of the SKHNE algorithm the useful aspect of this process is not in the final answer but in the partial results, as shown in the following. At each iteration $\boldsymbol{i}$ it can be readily shown that a set of polynomials $\boldsymbol{f_i}$, $\boldsymbol{g_i}$ and $\boldsymbol{\eta_i}$ are generated such that

(5a) $$f_i a + g_i b = \eta_i$$

In noting that eq.5(a) may also be written

(5b) $$g_i(z)b(z) \equiv \eta_i(z) \bmod a(z)$$

if $a(z)$ is set to z^{2t} and $b(z)$ to $S(z)$, such that

(5c) $$g_i(z)S(z) \equiv \eta_i(z) \bmod (z^{2t})$$

then at some stage in the iterative process, say stage $\boldsymbol{i = n}$, when $\boldsymbol{deg[\eta_n(z)]<t}$, this equation becomes identical to the 'key equation', eq.4, with $\boldsymbol{\eta_n(z) = \Omega(z)}$ and $\boldsymbol{g_n(z)}$ being the desired polynomial $\boldsymbol{\Lambda(z)}$.

This part of the algorithm can be implemented by the three steps:

i) Apply Euclid's algorithm to $\boldsymbol{a(z)=z^{2t}}$ and $\boldsymbol{b(z)=S(z)}$, using the following initial conditions:

$$g_{-1}(z) = 0, \quad g_0(z) = 1, \quad \eta_{-1}(z) = a(z) = z^{2t} \quad \text{and} \quad \eta_0(z) = b(z) = S(z).$$

ii) In each iteration i (i = 1, 2, …) divide $\boldsymbol{\eta_{i-2}(z)}$ by $\boldsymbol{\eta_{i-1}(z)}$ and obtain the quotient $\boldsymbol{q_i(z)}$ and the remainder $\boldsymbol{\eta_i(z)}$. Obtain polynomial $\boldsymbol{g_i(z)}$ by the formula: $g_i(z) = g_{i-2}(z) - q_i(z).g_{i-1}(z)$.

iii) Stop the recursive iterations when $\boldsymbol{deg[\eta_n(z)]<t}$, and set $\boldsymbol{\Lambda(z) = g_n(z)}$.

Termination of the algorithm: The algorithm will terminate properly (step iii) if no more than t errors occurred in the retrieved words because only in this case does the key equation holds. Otherwise (i.e. if more than t errors occur) the algorithm may produce an incorrect code vector (incorrect correction), or it may 'fail'. The latter failure, which has two modes, is detectable and is used to signal that the assumption that at most t errors have occurred is false. The two modes, [14], are:

Failure mode (a): It does not terminate properly. This occurrs iff $\boldsymbol{z^t | S(z)}$.

Failure mode (b): It terminates but produces a faulty $\Lambda(z)$. This may happen if (i) $\boldsymbol{0}$ is a root of $\boldsymbol{\Lambda(z)}$ or (ii) $\boldsymbol{\Lambda(z)}$ does not split into linear factors. Polynomial $\boldsymbol{\Lambda(z)}$ is constructed to split into linear factors with non-zero roots.

Error Correction: Having found $\boldsymbol{\Lambda(z)}$ the actual error locations can now be pinpointed. Since $\boldsymbol{\Lambda(z)}$ has roots $\alpha^{-j_1}, \alpha^{-j_2}, ..., \alpha^{-j_v}$ (eq. 3a), where $\boldsymbol{\nu \leq 2t}$ and $\boldsymbol{j_1, j_2, \ldots, j_v}$ are the error locations [i.e. an error occurs in position $\boldsymbol{j}$ *iff* $\boldsymbol{\Lambda(\alpha^j)=0}$]. The roots of $\boldsymbol{\Lambda(z)}$ are thus found by simply checking which elements $\boldsymbol{y}$ of $GF(\boldsymbol{2^m})$ satisfy the equation $\boldsymbol{\Lambda(y) = 0}$. If $\boldsymbol{\alpha^{p(k)}}$ is such a root, then the location $\boldsymbol{j_k}$ of the corresponding error location in the retrieved vector $\boldsymbol{r}$ is

(6) $$j_k = 2^m - 1 - p(k),$$

counting from the right, starting with $\boldsymbol{0}$, and where $\boldsymbol{k}$ will always be $\leq \boldsymbol{\nu}$. Thus the MDD error vector $\boldsymbol{e}$ is constructed and added to $\boldsymbol{r}$ to yield the corrected retrieved code vector $\boldsymbol{c}$ (i.e. a section of the archive data, which has yet to be de-compressed).

3. Implementation

Table 1 shows the steps in this SKHNE error location and correction algorithm and fig. 2 shows a block diagram of the time domain decoder structure as implemented. The language used was C++. This structure is akin to the general structure for time domain decoders, [13]. In fig. 2 apart from the retrieved sequence buffer and error correcting summer, the decoder contains:

a) a transform computer, which produces the syndrome components $\boldsymbol{S_i}$ via eq. 1, and syndrome polynomial $\boldsymbol{S(z)}$, via eq. 2,

b) the algorithm, "Solve key equation", to find $\boldsymbol{\Lambda(z)}$, and

c) the search algorithm to find the error locations and construct the error vector, $\boldsymbol{e}$ which is added to the original receive vector $\boldsymbol{r}$ thus executing the corrections.

4. Conclusion

This algorithm is the corner stone of a flexible, user controlled, efficient archival scheme providing multiple levels of reliability of the integrity of archived data in respect of errors. It provides effective capabilities for the correction of errors which have occurred in the archive itself, in the archive writing or the retrieval process. A reasonable approximation for the error rate, ***AER***, provided by the *BCH(**m,t**)* codes, for raw random probability of error $\boldsymbol{p}$, is

$$\mathrm{AER} = \binom{2^m - 1}{t+1} p^{t+1}$$

For example if using a *BCH(**8,8**)* code, AER is 10^{-29} for $p=10^{-5}$. The *BCH(**m,t**)* encoding is applied to the source file after it has been compressed, (and perhaps after any other software application-specific error protection scheme is included). The final file will be a little larger than the compressed file, e.g. 20% for *BCH(**8,8**);* i.e. if the compressed file is 12.5% of the original file, the final file would be 15% of the original file.

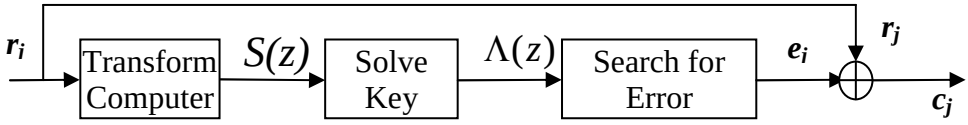

Figure 2. Binary BCH Decoder Block Diagram

Table 1: Seven Step (S) SKHNE error location and correction algorithm

S	*Action*	S	*Action*
1	Retrieve ***r.*** Calculate S_i (eq. 1). Write *S(z)* as eq.2. If *S(z)=0*, ***r*** is error free: set ***e*** to zero and go to *step 7;* otherwise go to *step2*.	**4**	Check for failure mode (b) of SKHNE algorithm: If positive then *stop* and issue message 1. Otherwise, go to *step 5*.
2	Check for failure mode (a) of SKHNE algorithm: if positive *stop* and issue message 1; otherwise, go to *step 3.*	**5**	Find: (i) the roots, $\alpha^{p(k)}$, of ***Λ(z);*** (ii) error locations j_k in *r*, eq.6.
3	Apply SKHNE algorithm to find $\Lambda(z)$. If the first remainder $\eta_i(z)$, which has degree < ***t*** is zero then there are more than ***t*** errors in the word: s*top* and issue message 1. Otherwise go to s*tep4.*	**6**	Construct error vector ***e*** with 1's in positions j_k and 0's elsewhere.
		7	Obtain the recovered corrected code vector c, via *e + r = c.* Return to *step 1* & repeat the process until decoding of all codewords is completed.

Message 1: "*Error Message – more errors have occurred in the archival retrieval than is possible to correct. Attempt retrieval with another device drive as the errors may not be inherent to the archive.*"

REFERENCES

[1] K.A.S. Immink. Coding Techniques for digital recorders. Englewood Cliffs, New Jersey, Prentice Hall Int. (U.K.) Ltd., 1991.

[2] J. Taylor. DVD demystified: the guidebook for DVD-video and DVD-ROM. McGraw-Hill. (ISBN 0-07-064841-7). 1997.

[3] DVD Forum web site. http://www.dvdforum.org. To date.

[4] R. Jung. ARJ program 2.60. http://www.arjsoftware.com/ ARJ.DOC. 1998

[5] S.B. Wicker. Error control systems for digital communication and storage. Prentice Hall. 1995.

[6] R.C. Chang and C.B. Shung. A (208,192;8) Reed-Solomon decoder for DVD application. *Proc. of IEEE International Conference Communications*, ICC 98 (ISBN: 0-7803-4788-9). Vol. **2**. (1998). Pp.957-960.

[7] P. Benachour, B. Honary, and G. Markarian. Improved decoding technique for the DVD. *IEEE Colloquium on Audio and Music Technology: The Challenge of Creative DSP* (Ref. No. 1998/470). (1998). Pp 17/1-17/5.

[8] K. Oh; W. Sung. An efficient Reed-Solomon decoder VLSI with erasure correction. *Proc. of IEEE Workshop on Signal Processing Systems, 1997*. SIPS 97 - Design and Implementation. (Editor(s): Ibrahim, M.K., Pirsch, P., McCanny, J.; ISBN: 0-7803-3806-5). (1997). Pp. 193-201.

[9] D.J. Parker. Defining DVD. *IEEE Multimedia*. (ISSN: 1070-986X) Vol.**6**, Issue:**1**. Jan-Mar. (1999). Pp. 80-84
[10] J.W. Einberger. CD-ROM as a mass storage device. *Proc. 9th IEEE Symposium on Mass Storage Systems*. Boulder, USA. (Editors: Friedman, K., O'Leary, B.T.; ISBN: 0-8186-8880-7). (1988). Pp.125 - 129.
[11] A. Hocquenghem. "Codes correcteurs d'erreurs", *Chiffres*, **2**, (1959). Pp.147-156.
[12] R.C. Bose, and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes, *Inf. Control*, **3**, (1960). Pp.68-79.
[13] G.C. Clark, Jr. and J. B. Cain. Error-Correcting Codes for Digital Communications. Plenum Press, 1981.
[14] O. Pretzel. Error-Correcting Codes and Finite Fields. Clarendon Press, 1992.
[15] E. R. Berlekamp. On decoding binary Bose-Chaudhuri-Hocquenghem codes, *IEEE Trans. Info. Theory*, **11**, (1965). Pp.577-9.
[16] J.L. Massey. Shift-register synthesis and BCH decoding, *IEEE Trans. Info. Theory*, **15**, (1969). Pp.122-7.
[17] Y. Sugiyama, M. Kasahara, S. Hirasawa, T. Namekawa. A method for solving key equation for decoding Goppa codes, *Inf. Control*, **27**, (1975). Pp.87-99.

Dr. Mairtin O'Droma,
Electronic and Computer Engineering,
University of Limerick, Ireland.
Mairtin.ODroma@ul.ie

Dr. Ivan Ganchev,
Dept. of Computer Systems,
University of Plovdiv, Bulgaria.
ivgan@pu.acad.bg