This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCOMM.2019.2911286, IEEE Communications Letters

Expanded Blaum-Roth Codes with Efficient Encoding and Decoding Algorithms

Mario Blaum, Veera Deenadhayalan and Steven Hetzler

***Abstract*—An expansion of Blaum-Roth codes by incorporating a vertical parity on $p \times p$ arrays, p a prime number, is described. The vertical parity allows for local recovery of a symbol within a column without invoking the remaining columns. Efficient encoding and erasure decoding procedures are presented, in particular, for the case of two column parities.**

***Index Terms*—Erasure-correcting codes, Blaum-Roth (BR) codes, Reed-Solomon (RS) codes, MDS codes, array codes.**

I. Introduction

Blaum-Roth (BR) codes [2] with r parity columns consist of $(p-1) \times p$ (binary) arrays, p a prime number, such that each line of slope i, $0 \leqslant i \leqslant r-1$, has even parity (the lines taken toroidally). For example, if $p=5$ and $r=3$, below are the lines of slope 0 (i.e., horizontal lines), slope 1 and slope 2 respectively, where we are adding a fifth row to facilitate the description:

♣	♣	♣	♣	♣
△	△	△	△	△
♠	♠	♠	♠	♠
◊	◊	◊	◊	◊
♡	♡	♡	♡	♡

♣	△	♠	◊	♡
△	♠	◊	♡	♣
♠	◊	♡	♣	△
◊	♡	♣	△	♠
♡	♣	△	♠	◊

♣	♠	♡	△	◊
△	◊	♣	♠	♡
♠	♡	△	◊	♣
◊	♣	♠	♡	△
♡	△	◊	♣	♠

.

The following is a 4×5 array in a BR code where each of the lines depicted above contains an even number of ones:

1	0	0	0	1
1	1	1	0	1
0	1	0	0	1
0	1	0	0	1
0	0	0	0	0

.

The last row is an imaginary row of zeros, which is not written.

The main application of array codes is its use in RAID type of architectures [4], where RS codes involving operations in a finite field are replaced by XOR operations. Many other codes in literature used the idea of parities along different lines to obtain certain desired properties, like independent parities and minimum number of encoding operations [1], [3], [5], [7].

An equivalent description of a BR code (and a very convenient one for decoding purposes) is given by an algebraic formulation of the problem. In effect, consider the ring of polynomials modulo $M_p(x)=1+x+x^2+\cdots+x^{p-1}$. Each column in the array is considered as an element in this ring, and let $M_p(\alpha)=0$. Then, a parity-check matrix of the code is the Reed-Solomon type of matrix

$$H_{p,r} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{r-1} & \alpha^{2(r-1)} & \dots & \alpha^{(r-1)(p-1)} \end{pmatrix}. \quad (1)$$

We will expand BR codes by adding a vertical parity (i.e., slope infinity) to the r parities of slope i, $0 \leqslant i \leqslant r-1$, making the elements of the code $p \times p$ arrays, as opposed to the $(p-1) \times p$ arrays of a BR code. From a practical point of view, like in a RAID architecture in which each column represents a storage device, the vertical parity allows for local recovery of a page or a sector in the storage device without invoking the other devices (columns). We also present an algebraic description that facilitates the decoding when multiple columns are erased. Specifically:

Definition 1. Let p be a prime, $\mathcal{R}_p$ be the ring of polynomials modulo $1+x^p$, and let $\mathcal{R}_p^{(0)} \subset \mathcal{R}_p$ be the ideal of polynomials modulo $1+x^p$ of even weight. If $\alpha^p=1$, an Expanded Blaum-Roth code $EBR(p,r)$ is the $[p, p-r]$ code over $\mathcal{R}_p^{(0)}$ whose parity-check matrix is given by (1). $\square$

Notice that the definition of EBR codes is similar to the one of BR codes. They both share the parity-check matrix (1), but while the columns of an array in a BR code are in the ring of polynomials modulo $M_p(x)$ and $M_p(\alpha)=0$, in an EBR code such columns are in the ideal $\mathcal{R}_p^{(0)}$ and $\alpha^p=1$. Geometrically, as stated above, the elements of an $EBR(p,r)$ code are $p \times p$ arrays such that each column has even parity, as well as any line of slope j for $0 \leqslant j \leqslant r-1$.

For example, consider $EBR(5,3)$. The following is an element in $EBR(5,3)$, i.e., each column as well as each line of slope 0, 1 and 2 has even parity:

1	0	0	1	0
1	1	1	0	1
0	1	1	0	0
0	1	1	0	0
0	1	1	1	1

.

The authors are with the IBM Research Division, Almaden Research Center, San Jose, CA 95120, USA (e-mail: Mario.Blaum@ibm.com, veerad,hetzler@us.ibm.com).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCOMM.2019.2911286, IEEE Communications Letters

The advantage of using codes over the ring $\mathcal{R}_p^{(0)}$ is that multiplications by α^i consist of rotations, no operations in a finite field nor XORs are involved. Codes over this ring have already been used, for example, in the context of Regenerating Codes [6].

II. Encoding and Decoding of EBR Codes

For any integer ℓ, denote by $\langle \ell \rangle_p$ the unique integer m such that $0 \leqslant m \leqslant p-1$ and $m \equiv \ell \pmod p$. Moreover, when there is no confusion, we will denote $\langle \ell \rangle_p$ simply as $\langle \ell \rangle$.

The following lemma (see [5], Lemma 5, or [6], Lemma 13) gives a recursion that will be used in the decoding of EBR codes.

Lemma 1. Let $\underline{x}(\alpha) = \bigoplus_{i=0}^{p-1} x_i\alpha^i \in \mathcal{R}_p^{(0)}$, $\alpha^p = 1$, p a prime number and $1 \leqslant j \leqslant p-1$. Then, the recursion $(1 \oplus \alpha^j)\underline{z}(\alpha) = \underline{x}(\alpha)$ has a unique solution in $\mathcal{R}_p^{(0)}$. Specifically, if $\underline{z}(\alpha) = \bigoplus_{i=0}^{p-1} z_i\alpha^i$, then

$$z_0 = \bigoplus_{u=1}^{(p-1)/2} x_{\langle 2uj \rangle} \tag{2}$$

$$z_{\langle ij \rangle} = z_{\langle (i-1)j \rangle} \oplus x_{\langle ij \rangle} \quad \text{for} \quad 1 \leqslant i \leqslant p-1. \tag{3}$$

The next example illustrates Lemma 1:

Example 1. Let $p=7$, $\underline{x}(\alpha) = 1 \oplus \alpha^3 \oplus \alpha^4 \oplus \alpha^6$, i.e., $x_0 = 1$, $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$, $x_5 = 0$ and $x_6 = 1$. Assume that we want to solve the recursion $(1 \oplus \alpha^3)\underline{z}(\alpha) = \underline{x}(\alpha)$. According to (2) and (3), since $j = 3$, $\langle 2j \rangle_7 = 6$, so

$$\begin{aligned}
z_0 &= x_6 \oplus x_{\langle 12 \rangle_7} \oplus x_{\langle 18 \rangle_7} = x_4 \oplus x_5 \oplus x_6 = 0 \\
z_3 &= z_0 \oplus x_3 = 1 \\
z_6 &= z_3 \oplus x_6 = 0 \\
z_2 &= z_6 \oplus x_2 = 0 \\
z_5 &= z_2 \oplus x_5 = 0 \\
z_1 &= z_5 \oplus x_1 = 0 \\
z_4 &= z_1 \oplus x_4 = 1,
\end{aligned}$$

so $\underline{z}(\alpha) = \alpha^3 \oplus \alpha^4$.

The recursion in Lemma 1 involves $\frac{3p-5}{2}$ XORs. We show next how to correct up to r erased columns by adapting the method in [2]. Assume that columns $i_0, i_1, \ldots, i_{\rho-1}$ have been erased, where $\rho \leqslant r$, and we denote by E_s the (erased) value of column i_s. Define the polynomial $G(x)$ of degree ρ as

$$G(x) = \prod_{s=1}^{\rho-1}(x \oplus \alpha^{i_s}) = \bigoplus_{s=0}^{\rho-1} g_s x^s. \tag{4}$$

Notice that

$$G(\alpha^{i_0}) = \prod_{s=1}^{\rho-1}(\alpha^{i_0} \oplus \alpha^{i_s}) \text{ and } G(\alpha^{i_j}) = 0 \text{ for } j \neq 0. \tag{5}$$

Denote the columns of the array by C_u, where $0 \leqslant u \leqslant p-1$. Assuming that the erased columns are zero, we compute the syndrome vectors

$$S^{(j)} = \bigoplus_{u=0}^{p-1} \alpha^{ju} C_u \quad \text{for} \quad 0 \leqslant j \leqslant \rho-1. \tag{6}$$

Hence, from the parity-check matrix (1) and (6), we also have

$$S^{(j)} = \bigoplus_{s=0}^{\rho-1} \alpha^{ji_s} E_s \quad \text{for} \quad 0 \leqslant j \leqslant \rho-1. \tag{7}$$

From (4), (5) and (7), we obtain

$$\begin{aligned}
\alpha^{-(\rho-1)i_0} \bigoplus_{j=0}^{\rho-1} g_j S^{(j)} &= \alpha^{-(\rho-1)i_0} \bigoplus_{j=0}^{\rho-1} g_j \bigoplus_{s=0}^{\rho-1} \alpha^{ji_s} E_s \\
&= \alpha^{-(\rho-1)i_0} \bigoplus_{s=0}^{\rho-1} E_s \bigoplus_{j=0}^{\rho-1} g_j (\alpha^{i_s})^j \\
&= \alpha^{-(\rho-1)i_0} \bigoplus_{s=0}^{\rho-1} E_s G(\alpha^{i_s}) \\
&= \alpha^{-(\rho-1)i_0} \left(\prod_{s=1}^{\rho-1} (\alpha^{i_0} \oplus \alpha^{i_s}) \right) E_0 \\
&= \left(\prod_{s=1}^{\rho-1} (1 \oplus \alpha^{i_s - i_0}) \right) E_0.
\end{aligned} \tag{8}$$

So, from (8), after computing $\alpha^{-(\rho-1)i_0} \bigoplus_{j=0}^{\rho-1} g_j S_j$, E_0 can be obtained by applying the recursion given by (2) and (3) in Lemma 1 $\rho-1$ times. Once E_0 is obtained, we are left with $\rho-1$ erasures, and we proceed by induction.

We illustrate next the decoding with an example.

Example 2. Consider $EBR(5,3)$, and assume that we want to decode the following array, where the blank spaces correspond to erasures:

		0		
1		1		
0		1		
0				
0		1		

.

We can see that columns 1, 3 and 4 are erased while in columns 0 and 2 there is one erased element. Since the columns have even parity, the first step is obtaining the erased elements in columns 0 and 2. Once this is done, we obtain

1		0		
1		1		
0		1		
0		1		
0		1		

.

By (4), since $\alpha^5 = 1$,

$$G(x) \;=\; (x\oplus\alpha^3)(x\oplus\alpha^4) \;=\; \alpha^2\oplus(\alpha^3\oplus\alpha^4)x\oplus x^2.$$

Also, assuming that the erased columns are zero when computing the syndromes, by (6), we obtain

$$\begin{aligned} S_0 &= 1\oplus\alpha^2\oplus\alpha^3\oplus\alpha^4 \\ S_1 &= \alpha^3\oplus\alpha^4 \\ S_2 &= \alpha^2\oplus\alpha^3. \end{aligned}$$

Next we compute

$$g_0S_0\oplus g_1S_1\oplus g_2S_2 \;=\; 1\oplus\alpha^4.$$

Now, from (8) and since $\rho=3$, we have to solve the double recursion

$$(1\oplus\alpha^2)(1\oplus\alpha^3)E_0 \;=\; \alpha^{-2}(1\oplus\alpha^4)=\alpha^2\oplus\alpha^3.$$

Let $(1\oplus\alpha^3)E_0=V_0$, then we have to solve first

$$(1\oplus\alpha^2)V_0 \;=\; \alpha^2\oplus\alpha^3,$$

Applying the recursion given by (2) and (3) as illustrated in Example 1, we obtain

$$V_0 \;=\; 1\oplus\alpha^3.$$

Next we have to solve

$$(1\oplus\alpha^3)E_0 \;=\; 1\oplus\alpha^3.$$

This gives,

$$E_0 \;=\; \alpha\oplus\alpha^2\oplus\alpha^3\oplus\alpha^4.$$

Recomputing the syndromes,

$$\begin{aligned} S_0 &= S_0\oplus E_0 \;=\; 1\oplus\alpha \\ S_1 &= S_1\oplus \alpha E_0 \;=\; 1\oplus\alpha^2. \end{aligned}$$

Repeating the procedure for two erasures, we now have

$$G(x) \;=\; \alpha^4\oplus x$$

and

$$g_0S_0\oplus g_1S_1 \;=\; \alpha^2\oplus\alpha^4.$$

Now $\rho=2$, and by (8), we have to solve the recursion

$$(1\oplus\alpha)E_1 \;=\; \alpha^{-3}(\alpha^2\oplus\alpha^4)=\alpha\oplus\alpha^4.$$

Solving this recursion using Lemma 1, we obtain,

$$E_1 \;=\; 1\oplus\alpha^4.$$

Finally, we recompute

$$S_0 \;=\; S_0\oplus E_1 \;=\; \alpha\oplus\alpha^4 \;=\; E_2.$$

The final decoded array is then

1	0	0	1	0
1	1	1	0	1
0	1	1	0	0
0	1	1	0	0
0	1	1	1	1

.

The encoding is a special case of the decoding. For example, we may use the last row and the last r columns to store the parities. In this case, since we know where the erasures are, the coefficients of $G(x)=\prod_{j=1}^{r-1}(x\oplus\alpha^{p-r+j})$ may be precomputed, making the process faster.

The number of columns does not need to be a prime number. For example, in the description above, we have $k=p-r$ columns carrying information, where p is prime and $1\leqslant r< p$. But we can have $1\leqslant k\leqslant p-r$ data columns, and if $k<p-r$, we can pad the array with $p-r-k$ 0 columns to encode the data into a $p\times(k+r)$ array. The zero columns are ignored when writing the final $p\times(k+r)$ encoded array.

In the next section we examine the encoding of the important case $r=2$ (which corresponds to RAID 6 in RAID architectures).

III. Encoding of $EBR(p,2)$

We will show how to encode the specific case of $EBR(p,2)$, first following the procedure described in Section II, and then making a modification that saves on the total number of XORs required.

Assume that we want to encode in $EBR(p,2)$ a $(p-1)\times k$ array with entries $a_{u,v}$, where p is prime, $k\leqslant p-2$, $0\leqslant u\leqslant p-2$ and $0\leqslant v\leqslant k-1$.

The first step is obtaining the symbols $a_{p-1,j}$ for $0\leqslant j\leqslant k-1$. Hence,

$$a_{p-1,j} \;=\; \bigoplus_{u=0}^{p-2} a_{u,j}. \tag{9}$$

Each of the computations in (9) requires $p-2$ XORs, and since we have k of them, computing the $a_{p-1,v}$s for $0\leqslant v\leqslant k-1$ takes $k(p-2)$ XORs. We now have a $p\times k$ array that we pad with $p-k-2$ zero columns, giving a $p\times(p-2)$ array.

Next we follow the decoding method described in Section II assuming that columns $p-2$ and $p-1$ have been erased. Taking the data columns $C_j=\bigoplus_{i=0}^{p-1}a_{i,j}\alpha^i$ for $0\leqslant j\leqslant k-1$, we want to obtain columns C_{p-2} and C_{p-1}.

By (4), $G(x)=\alpha^{p-1}\oplus x$, and by (6),

$$S^{(0)} = \bigoplus_{j=0}^{k-1} C_j \quad \text{and} \quad S^{(1)} = \bigoplus_{j=0}^{k-1} \alpha^j C_j. \tag{10}$$

If $S^{(v)} = \bigoplus_{i=0}^{p-1} s_i^{(v)} \alpha^i$ for $0 \leqslant v \leqslant 1$, then, by (10),

$$s_i^{(0)} = \bigoplus_{j=0}^{k-1} a_{i,j} \text{ and } s_i^{(1)} = \bigoplus_{j=0}^{k-1} a_{\langle i-j \rangle, j} \text{ for } 0 \leqslant i \leqslant p-1. \tag{11}$$

Thus, the computation of the syndrome vectors $S^{(0)}$ and $S^{(1)}$, according to (10) and (11), requires $2(k-1)p$ XORs.

By (8), since $E_0 = C_{p-2} = \bigoplus_{u=0}^{p-1} a_{u,p-2}\alpha^u$, $i_0 = p-2$, $i_1 = p-1$ and $\rho = 2$, we have to solve

$$(1 \oplus \alpha)\, C_{p-2} \;=\; \alpha S^{(0)} \oplus \alpha^2 S^{(1)}. \tag{12}$$

Computing the right hand side of equation (12) requires p XORs, and, as we have seen in Section II, solving this recursion takes $(3p-5)/2$ XORs.

Finally, we obtain $C_{p-1} = C_{p-2} \oplus S^{(0)}$, which requires another p XORs. Thus, the total number of XORs in the encoding procedure described, that from now on we refer to as Algorithm 1, requires $(3p(2k+1) - 4k - 5)/2$ XORs.

Next we present a method that reduces the number of XORs in Algorithm 1. We will refer to this method as Algorithm 2. Specifically, we concentrate on the recursion given by (12). The right hand side of (12) is given by

$$\alpha S^{(0)} \oplus \alpha^2 S^{(1)} \;=\; \bigoplus_{v=0}^{p-1} \left(s_{\langle v-1 \rangle}^{(0)} \oplus s_{\langle v-2 \rangle}^{(1)} \right) \alpha^v. \tag{13}$$

The bottleneck for solving the recursion given by (12) is in computing the first element $a_{0,p-2}$, which according to (2) and (13), is

$$a_{0,p-2} \;=\; \bigoplus_{u=0}^{(p-3)/2} \left(s_{\langle 2u+1 \rangle}^{(0)} \oplus s_{\langle 2u \rangle}^{(1)} \right). \tag{14}$$

By (11), (14) becomes

$$a_{0,p-2} = \bigoplus_{u=0}^{(p-3)/2} \bigoplus_{j=0}^{k-1} \left(a_{\langle 2u+1 \rangle, j} \oplus a_{\langle 2u-j \rangle, j} \right) = \bigoplus_{j=0}^{k-1} W_j, \tag{15}$$

where

$$W_j = \bigoplus_{u=0}^{(p-3)/2} \left(a_{\langle 2u+1 \rangle, j} \oplus a_{\langle 2u-j \rangle, j} \right) = \bigoplus_{i=0}^{p-j-2} a_{i,j} \tag{16}$$

for $0 \leqslant j \leqslant k-1$ and the second equality is obtained using (9) when j is odd.

Since, by (9) and (16), $a_{p-1,j} = W_j \oplus \bigoplus_{i=p-j-1}^{p-2} a_{i,j}$, the W_js can be stored when computing the $a_{p-1,j}$s. Therefore, obtaining $a_{0,p-2}$ in (15) requires $k-1$ XORs.

Once $a_{0,p-2}$ is obtained from (15), the remaining symbols $a_{i,p-1}$ and $a_{i,p-2}$ may be computed by the following recursion:

$$a_{i,p-1} \;=\; s_i^{(0)} \oplus a_{i,p-2} \text{ for } 0 \leqslant i \leqslant p-1 \tag{17}$$

$$a_{i,p-2} \;=\; s_{\langle i-2 \rangle}^{(1)} \oplus a_{i-1,p-1} \text{ for } 1 \leqslant i \leqslant p-1. \tag{18}$$

The encoding is completed by eliminating the padding columns to obtain a $p \times (k+2)$ encoded array.

Solving (17) and (18) requires $2p-1$ XORs. Thus, the total number of XORs in Algorithm 2 is $(3p-1)k-2$, a number that is smaller than $(3p(2k+1) - 4k - 5)/2$, the number of XORs in Algorithm 1.

Table I compares the number of XORs of Algorithms 1 and 2. We can see that Algorithm 2 always needs less XORs than Algorithm 1, but the savings are more significant when $k << p$.

		Algorithm 1	Algorithm 2	
p	k	$\frac{3p(2k+1)-4k-5}{2}$	$(3p-1)k-2$	Improvement %
17	8	415	398	4.1%
17	15	758	748	1.3%
127	8	3220	3038	5.7%
127	50	19138	18998	.7%
127	125	47563	47498	.1%
257	8	6535	6158	5.8%
257	50	38833	38498	.9%
257	255	196478	196348	.07%

TABLE I
NUMBER OF XORS OF ALGORITHMS 1 AND 2.

IV. CONCLUSIONS

Blaum-Roth codes consist of $(p-1) \times p$ arrays with r parity columns, such that lines of slope j (with a toroidal topology), $0 \leqslant j \leqslant r-1$, have even parity. These codes have been expanded to $p \times p$ arrays such that each column has even parity. This property allows for local recovery of an erased symbol within the column without invoking any of the other $p-1$ columns, while the MDS property is preserved. The local property makes the codes attractive for RAID applications in which sectors may degrade in time (like in SSDs), occasionally exceeding the correction power of the sector ECC.

REFERENCES

[1] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, "The EVEN-ODD Code and its Generalization," in "High Performance Mass Storage and Parallel I/O: Technologies and Applications," edited by H. Jin, T. Cortes, and R. Buyya, IEEE & Wiley Press, New York, Chapter 14, pp. 187–208, 2001.

[2] M. Blaum and R. M. Roth, "New Array Codes for Multiple Phased Burst Correction," IEEE Trans. on Information Theory, vol. IT-39, pp. 66-77, January 1993.

[3] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," Proc. 3rd Conf. File and Storage Technologies - FAST'04, San Francisco, CA, March/April 2004.

[4] G. A. Gibson, "Redundant Disk Arrays," MIT Press, 1992.

[5] H. Hou, Y. S. Han, K. W. Shum, and H. Li, "A Unified Form of EVEN-ODD and RDP Codes and Their Efficient Decoding," IEEE Trans. on Communications, vol. 66, pp. 5053–66, November 2018.

[6] H. Hou, K. W. Shum, M. Chen, and H. Li, "BASIC codes: Low-complexity regenerating codes for distributed storage systems," IEEE Trans. on Information Theory, vol. IT-62, pp. 3053-69, June 2016.

[7] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," IEEE Transactions on Information Theory, vol. IT-45, pp. 272–76, January 1999.