Chapter 2 X-Code: MDS Array Codes with Optimal Encoding

2.1 Introduction

As stated in Chapter 1, array codes have important applications in storage systems[6] [12] and have been studied extensively[4][5][7][8][15]. A common property of these codes is that the encoding and decoding procedures use only simple XOR and cyclic shift operations, and thus are more efficient than Reed-Solomon codes in terms of computation complexity [6]. In this chapter, we describe the *X-Code*, a new class of array codes of size $n \times n$ over any Abelian group $G(q)$ with an addition operation $+$, where q is the size of the group. When $q = 2^m$, the addition operation is just the usual bit-wise XOR operation. Similar to the codes in [4][7], the error model of the X-Code is that errors or erasures are columns of the array, i.e., if one symbol of a column is an error or erasure, then the whole column is considered to be an error or erasure. The same model is also used for the B-Code in the next chapter. As usual, the dimension of the code is defined to be $k = log_{q^n} N$, where N is the number of its codewords. The code can also be viewed as an (n, k, d) code over $G(q^n)$. Its distance is defined over $G(q^n)$, i.e., over the columns of the array. The X-Code is an *MDS* (*Maximum Distance Separable*) code of distance $d = 3$, i.e., $k = n - 2$, which meets the Singleton bound[19]: $d = n - k + 1$.

Although it was shown[37][8] that for general array codes of distance 3, the lower bound 2 of update complexity is achievable, the code in [37] and later its clearer form [8] are described by *parity check matrix* and not directly as array codes. The new family of array codes, called the X-Codes, has a much simpler and direct *geometrical structure* and has an update complexity of *exactly* 2.

Both the X-Codes and the codes in [37] and [8] combine information and parity symbols within columns in order to achieve optimal update complexity. The redundancy of the X-Code is obtained by adding two parity *rows* rather than two parity columns, which results in a nice property that updating one information symbol affects only *two* parity symbols, i.e., the *update complexity* is always *two*. In addition, the number of operations for computing parity symbols is the same for every column, namely, the computational load is evenly distributed among all the columns, and thus the bottleneck effects of repeated *write* operations are naturally overcome.

In summary, the main contribution of this chapter is a construction for the X-Code, a new class of MDS array codes of distance 3, with the properties of optimal update complexity and balanced computations. The simple geometrical structure of the the X-Code makes its decoding very efficient, both for *two erasures* and for *one error*.

This chapter is organized as follows. In Section 2.2, the encoding scheme of the X-Code is described, and a proof of its *MDS* property is presented. In Section 2.3, an efficient decoding algorithm for correcting two erasures and an efficient algorithm for correcting one error are provided. Section 2.4 concludes the chapter and presents some future research directions.

2.2 X-Code Description

In the X-Code, information symbols are placed in an array of size $(n-2) \times n$. Like other array codes [4][5][7][15], parity symbols are constructed by adding (with the group addition operation $+$) the information symbols along several *parity check lines* or *diagonals* of some given *slopes*. But instead of being put in separate columns, the parity symbols of the X-Code are placed in *two* additional *rows*. So the coded array is of size $n \times n$, with the first $n-2$ rows containing information symbols, and the last two rows containing parity symbols. Notice that each column has information symbols as well as parity symbols, i.e., information symbols and parity symbols are mixed in each column. By the structure of the code, if two columns are erased, the number of remaining symbols is $n(n-2)$, which is equal to the number of original information symbols, making it possible to recover the two column *erasures*, i.e., missing columns.

2.2.1 Encoding Procedure

Let $C_{i,j}$ be the symbol at the ith row and jth column. The parity symbols of the X-Code are constructed according to the following encoding rules:

$$C_{n-2,i} = \sum_{k=0}^{n-3} C_{k,\langle i+k+2\rangle_n}$$

$$C_{n-1,i} = \sum_{k=0}^{n-3} C_{k,\langle i-k-2\rangle_n} \tag{2.1}$$

where $i = 0, 1, \cdots, n-1$, and $\langle x \rangle_n = x \bmod n$. Geometrically speaking, the two parity rows are just the checksums along diagonals of slopes 1 and -1 respectively. The following example gives a construction of the X-Code of size 5×5.

Example 2.1 *X-Code of size 5 × 5*

The first parity row is calculated along the diagonals of slope 1, with the last row being an imaginary 0-row, as follows:

△	♣	♢	♡	♠		1	0	0	1	1
♠	△	♣	♢	♡		0	1	0	1	1
♡	♠	△	♣	♢		0	0	1	0	1
♢	♡	♠	△	♣	←1st parity check row→	0	0	1	1	0
♣	♢	♡	♠	△	←imaginary 0-row →	0	0	0	0	0

The second parity row is calculated along the diagonals of slope -1, as follows:

△	♣	♢	♡	♠		1	0	0	1	1
♣	♢	♡	♠	△		0	1	0	1	1
♢	♡	♠	△	♣		0	0	1	0	1
♡	♠	△	♣	♢	←2nd parity check row→	1	1	0	1	1
♠	△	♣	♢	♡	←imaginary 0-row→	0	0	0	0	0

Then the complete codeword is

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

□

From the construction of the X-Code, it is easy to see that the two parity rows are obtained independently; more specifically, each information symbol affects exactly *one* parity symbol in each parity row. All parity symbols depend only on information symbols, *not* on each other. So, updating one information symbol results in updating only *two* parity symbols. Thus the X-Code has the optimal encoding (or update) property, i.e., its update complexity of 2 matches the lower bound for any code of distance 3.

It is also easy to see that the X-Code is a cyclic code in terms of columns, i.e., cyclically shifting columns of a codeword of the X-Code results in another codeword of the X-Code.

In addition, notice that each column has two parity symbols, each of which is the checksum of $n-2$ information symbols. Thus computing parity symbols at each column needs $2(n-3)$ group additions. This balanced computation property of the X-Code is very useful in applications that require evenly distributed computations.

2.2.2 The MDS Property

In this section, we state and prove the *MDS property* of the X-Code.

Theorem 2.1 (MDS Property)

The X-Code has column distance of 3, i.e., it is MDS, if and only if n *is a prime number.*

Proof: Let us start with the *sufficient* condition, namely, prove that for any prime number n, the X-Code is *MDS*.

First observe that the X-Code is a linear code, thus proving that the code has distance of 3 *is equivalent to* proving that the code has *minimum column weight* w_{min} of 3, i.e., a valid codeword of the X-Code has at least 3 nonzero columns. (A column is called a nonzero column if at least one symbol in the column is nonzero.) We will prove this by contradiction.

From the construction of the X-Code, checksums are obtained along diagonals of slope 1 or slope -1, so it is impossible to have only *one* nonzero column, thus $w_{min} > 1$.

Now suppose $w_{min} = 2$. Without loss of generality, we can assume the nonzero columns are the 0th and kth columns where $1 \leq k \leq n-1$, because of the column cyclic property of the X-Code. Denote the ith symbol of the 0th and kth columns a_i and b_i respectively.

Observe that any one diagonal of slope 1 or -1 only traverses $n-1$ columns, then among the diagonals of slope 1, the diagonal crossing a_{n-1-k} does *not* cross any symbol of the kth column, and the diagonal crossing b_{k-1} does *not* cross any symbol of the 0th column, so $a_{n-1-k} = 0$ and $b_{k-1} = 0$. Because the diagonals of slope -1 have the same property, we can also get $a_{k-1} = 0$ and $b_{n-1-k} = 0$ (or $b_{n-1} = 0$ if $k = 1$).

Starting from $a_{k-1} = 0$, we get $b_{2k-1} = 0$, since they are in the same diagonal of slope 1; then we get $a_{3k-1} = 0$, since it is on the same diagonal of slope 1 with b_{2k-1}, $\cdots$, and so on, we have

$$a_{k-1} = a_{3k-1} = a_{5k-1} = \cdots = a_{(n-2)k-1} = 0$$

and

$$b_{2k-1} = b_{4k-1} = b_{6k-1} = \cdots = b_{(n-1)k-1} = 0$$

where all indices are *mod* n.

Similarly, starting from $a_{n-1-k} = 0$, we have

$$a_{n-1-k} = a_{n-1-3k} = \cdots = a_{n-1-(n-2)k} = 0$$

and

$$b_{n-1-2k} = b_{n-1-4k} = \cdots = b_{n-1-(n-1)k} = 0$$

again, all indices above are *mod* n.

We can describe the above 4 sets of entries in the array as follows. Let $A_0 = \{\langle(2m+1)k-1\rangle_n : m = 0, 1, \cdots, \frac{n-3}{2}\}$, and $A_1 = \{\langle n-(2l+1)k-1\rangle_n : l = 0, 1, \cdots, \frac{n-3}{2}\}$, let $B_0 = \{\langle 2mk-1\rangle_n : m = 1, 2, \cdots, \frac{n-1}{2}\}$, and $B_1 = \{\langle n-2lk-1\rangle_n : l = 1, 2, \cdots, \frac{n-1}{2}\}$, notice that none of the sets includes $n-1$, since n is prime. This can also be seen from the construction of the X-Code, since the $(n-1)$th row is just an imaginary all-0 row and it does not need to be considered. An illustration of the above sets for n = 5 and k = 2 is as follows:

A_0		B_1		
A_0		B_1		
A_1		B_0		
A_1		B_0		

Since n is prime, for any $1 \leq k \leq n-1$, $\gcd(n,k) = 1$, $\|A_0\| = \|A_1\| = \frac{n-1}{2}$, and if there were m and l such that

$$(2m+1)k - 1 \equiv n - (2l+1)k - 1 \bmod n \tag{2.2}$$

then,

$$2(m+l+1)k \equiv 0 \bmod n \tag{2.3}$$

but $1 \leq m+l+1 \leq n-2, \gcd(m+l+1, n) = 1, \gcd(2k, n) = 1$, so it is impossible to have such a pair of m and l, i.e., $\|A_0 \cap A_1\| = 0$. Notice that $n-1 \equiv (2\frac{n-1}{2}+1)k - 1 \bmod n$, so

$$A_0 \cup A_1 = \{0, 1, \cdots, n-2\}$$

Similarly,

$$B_0 \cup B_1 = \{0, 1, \cdots, n-2\}$$

So all the first $n-1$ symbols in the 0th and the kth columns are 0's, obviously the last symbols in the 0th and the kth columns should be also 0's. This is a contradiction. Thus, $w_{min} \geq 3$, but it is easy to see there is a codeword of column weight 3, so $w_{min} = 3$. This concludes the proof for the sufficient condition.

On the other hand, from the equation Eq. (2.3), if n were not a prime number, then it could be factored into two factors n_1 and n_2. Thus we got a solution (k, l, m) for the equation Eq. (2.3) or Eq. (2.2), where $k = n_1$, and $m + l + 1 = n_2$, and $2 \leq k \leq n-1$. This means there is a codeword of weight 2, or equivalently the distance of the code is no greater than 2. This contradicts the fact that the code is of distance 3. So n being a prime number is also a necessary condition to the *MDS* property of the X-Code. $\square$

Remarks:

1. For the sufficient condition, we can always find a diagonal of one slope which traverses only one of the two columns. Thus the traversed symbol must be 0. Starting from this 0-symbol and using the diagonal of the other slope crossing this symbol, we can determine that the symbol crossed by the diagonal in the other column must be also 0. So this saw-like recursive procedure can proceed until it hits a parity symbol at one of the two columns, since a parity symbol can only lie in one diagonal. We call this saw-like recursion a *decoding chain*. Since there are four parity symbols in the two columns, there are at most four decoding chains. (A simple calculation can show that there are two decoding chains when $k = 1$ and four decoding chains otherwise.) If n is prime, the procedure of getting the decoding chains will stop with all the symbols

in the two columns being 0s. Since this procedure is deterministic once the positions of the two columns are given, it also provides an efficient erasure decoding algorithm.

2. In the code construction above, we use diagonals of slope 1 and -1. This choice of slopes is not unique. In fact, for $s = 1, \cdots, \frac{n-1}{2}$, codes constructed by the pair of slopes $(s, -s)$ are *MDS* if and only if n is prime. The proof is similar to the case where the slope pair is (1,$-$1). It seems that other slope pairs do not provide advantages over (1,$-$1), so in this paper we will focus on the X-Codes generated by the slope (1,$-$1).

2.3 Efficient Decoding Algorithms

In this section, we present decoding algorithms for correcting two erasures or one error of the X-Code. Neither the encoding algorithm of the code or decoding algorithms require any finite field operations. Instead, the only operations needed are additions and cyclic shifts, both of which can be implemented very efficiently in software and/or hardware. It is clear how to correct one erasure, since the erasure can be easily recovered along one of the diagonals. So we will proceed with correcting two erasures.

2.3.1 Correcting Two Erasures

First notice that in an array of size $n \times n$, if two columns are erasures, then the key unknown symbols of the two erased columns are the information symbols. So the number of unknown symbols is $2(n-2)$. On the other hand, in the remaining array, there are $2(n-2)$ parity symbols that include all the $2(n-2)$ unknown symbols. Hence, correcting the two erasures is only a problem of solving for $2(n-2)$ unknowns from $2(n-2)$ linear equations. Since the X-Code is of distance 3, it can correct two erasures; thus the $2(n-2)$ linear equations must be linearly independent, i.e., the linear equations are solvable. Now notice that a parity symbol can *not* be affected by more than one information symbol in the same column, so each equation has at most two unknown symbols, with some having only one unknown symbol. This drastically reduces the complexity of solving the equations.

Suppose the erasure columns are the ith and jth $(0 \le i < j \le n-1)$ columns. Since each diagonal traverses only $n-1$ columns, if a diagonal crosses a column at the last row, no symbols of that column are included in this diagonal. This determines the position of the parity symbol that includes only one symbol from the two erasure columns, thus this symbol can be immediately recovered from a simple checksum along this diagonal. From this symbol, we can get a decoding chain as discussed in Remark 1 in Section 2.2. Using this decoding chain and the other one (if $j - i = 1$) or three (if $j - i > 1$), all unknown symbols can be recovered.

Now let us calculate the starting parity symbols of the decoding chains. First consider the diagonals of slope 1. Suppose the xth symbol of the ith column is the only unknown symbol in a diagonal. This diagonal hits the jth column at the $(n-1)$th row, and hits the first parity row at the yth column, i.e., the three points $(x, i), (n-1, j)$ and $(n-2, y)$ are on the same diagonal of slope 1, thus the following equations hold:

$$\begin{cases} (n-1) - x \equiv j - i \bmod n \\ (n-1) - (n-2) \equiv j - y \bmod n \end{cases}$$

Since $1 \leq j - i \leq n - 1$, and $0 \leq j - 1 \leq n - 2$, the solutions for x and y are

$$\begin{cases} x = \langle (n-1) - (j-i) \rangle_n = (n-1) - (j-i) \\ y = \langle j - 1 \rangle_n = j - 1 \end{cases}$$

So from the parity symbol $C_{n-2,j-1}$, we can immediately get the symbol $C_{(n-1)-(j-i),i}$ in the ith column. Similarly, the symbol $C_{(j-i)-1,j}$ in the jth column can be solved directly from the parity symbol $C_{n-2,\langle i-1 \rangle_n}$.

Symmetrically with the diagonals of slope -1, the symbol $C_{(j-i)-1,i}$ in the ith column can be solved from the parity symbol $C_{n-1,\langle j+1 \rangle_n}$, and the symbol $C_{(n-1)-(j-i),j}$ in the jth column can be solved from the parity symbol $C_{n-1,i+1}$.

A formal algorithm for correcting the two erasures ith and jth $(0 \leq i < j \leq n-1)$ columns of the X-Code can be described as follows:

Algorithm 2.1 (Correcting Two Erasures)

Use each of the four parity symbols $C_{n-2,j-1}$, $C_{n-2,\langle i-1 \rangle_n}$, $C_{n-1,\langle j+1 \rangle_n}$ *and* $C_{(n-1)-(j-i),j}$ *as the starting point of a decoding chain; in each decoding chain use the saw-like recursion method to recover unknown symbols until a parity symbol at one of the two erasure columns is hit, then start a new decoding chain, as discussed in Section 2.2.*

A pseudo-code description of the algorithm is as follows:

1. Init_Slope_Set = { 1, 1, -1, -1 }

$Init\_Par\_Col\_Set = \{j-1, \langle i-1\rangle_n, \langle j+1\rangle_n, i+1\}$;

$Init\_Sym\_Col\_Set = \{i, j, i, j\}$;

$Init\_Sym\_Row\_Set = \{(n-1)-(j-i), (j-i)-1, (j-i)-1, (n-1)-(j-i)\}$;

$i = -1$;

2. $i++$;

If $i == 4$ **Then**

Compute $P_0[i], P_0[j], P_1[i], P_1[j]$ according to the encoding rule Eq. (2.1);

Stop;

Else

$Slope = Init\_Slope\_Set[i]$;

$Par\_Col = Init\_Par\_Col\_Set[i]$;

$Sym\_Col = Init\_Sym\_Col\_Set[i]$;

$Sym\_Row = Init\_Sym\_Row\_Set[i]$;

End If

3. **If** $Par\_Col == i$ **Or** $Par\_Col == j$ **Then**

Goto 2;

Else

If $Slope == 1$ **Then**

$C_{Sym\_Row, Sym\_Col} = P_0[Par\_Col] + \sum_{k=0, k\neq Sym\_Row}^{n-3} C_{k, \langle Par\_Col+k+2\rangle_n}$;

Else

$C_{Sym\_Row, Sym\_Col} = P_1[Par\_Col] + \sum_{k=0, k\neq Sym\_Row}^{n-3} C_{k, \langle Par\_Col-k-2\rangle_n}$;

End If

End If

4. $Slope = -Slope$;

$Par\_Col = \langle Sym\_Col - Slope * (Sym\_Row + 2)\rangle_n$;

If $Sym\_Col == i$ **Then**

$Sym\_Col = j$;

Else

$Sym\_Col = i$;

End If

$Sym\_Row = \langle n - 2 - Slope * (Par\_Col - Sym\_Col)\rangle_n$;

Goto 3;

□

Step 1 of the algorithm computes those positions of the four parity symbols that contain only one unknown symbol. Steps 2 through 4 include the saw-like recursive procedure described above. Step 3 is just the checksum calculation along a diagonal of slope *Slope* crossing the parity symbol $P_0[Par\_Col]$ (or $P_1[Par\_Col]$). This recovers the unknown symbol C_{Sym_Row,Sym_Col} if the parity symbol is not in one of the erasure columns; otherwise, it just restarts with another parity symbol obtained in Step 1. Step 4 uses the symbol that was just found to calculate the position of the next unknown symbol.

The correctness of the algorithm can be deduced from the proof of Theorem 1 and Remark 1 in Section 2.2. The complexity of the algorithm is easy to analyze. Each iteration solves one unknown symbol and requires $(n-3)$ additions. So to correct two erasure columns, the decoding algorithm needs $2n(n-3)$ additions, just the same as that of the encoding algorithm.

The following is a simple example to show how the decoding algorithm works. To be more general, we use symbols rather than numerical values.

Example 2.2 *Correcting Two Erasures of a* 5×5 *X-Code*

Without loss of generality, we assume the last (i.e., the 4th) column is one of the erasures, and because of the symmetry of the code, we only need to examine the cases where the other erasure is the 3rd or the 2nd column.

Case 1. $i = 3$, $j = 4$

Then the remaining array is:

a_0	a_1	a_2	$?(a_3)$	$?(a_4)$
b_0	b_1	b_2	$?(b_3)$	$?(b_4)$
c_0	c_1	c_2	$?(c_3)$	$?(c_4)$
$d_0 = a_2 + b_3 + c_4$	$d_1 = a_3 + b_4 + c_0$	$d_2 = a_4 + b_0 + c_1$	$?(d_3)$	$?(d_4)$
$e_0 = a_3 + b_2 + c_1$	$e_1 = a_4 + b_3 + c_2$	$e_2 = a_0 + b_4 + c_3$	$?(e_3)$	$?(e_4)$

After omitting the obvious checksum calculations, the decoding chain for the erasures would be as follows:

$$a_4(d_2) \rightarrow b_3(e_1) \rightarrow c_4(d_0)$$

$$a_3(e_0) \rightarrow b_4(d_1) \rightarrow c_3(e_2)$$

Each chain above represents a recursion starting from a parity symbol, and in each term of the chain, $x(y)$ means that the symbol x can be recovered from the parity symbol y. Obviously, d_3, d_4, e_3 and e_4 can be easily computed after all others are known.

Case 2. $i = 2$, $j = 4$

Then the remaining array is follows:

a_0	a_1	$?(a_2)$	a_3	$?(a_4)$
b_0	b_1	$?(b_2)$	b_3	$?(b_4)$
c_0	c_1	$?(c_2)$	c_3	$?(c_4)$
$d_0 = a_2 + b_3 + c_4$	$d_1 = a_3 + b_4 + c_0$	$?(d_2)$	$d_3 = a_0 + b_1 + c_2$	$?(d_4)$
$e_0 = a_3 + b_2 + c_1$	$e_1 = a_4 + b_3 + c_2$	$?(e_2)$	$e_3 = a_1 + b_0 + c_4$	$?(e_4)$

Now the decoding chain becomes:

$$
\begin{aligned}
&c_2(d_3) \rightarrow a_4(e_1) \\
&b_4(d_1) \\
&b_2(e_0) \\
&c_4(e_3) \rightarrow a_2(d_0)
\end{aligned}
$$

Again, d_2, d_4, e_2 and e_4 are easy to get after all other symbols are obtained.

□

2.3.2 Correcting One Error

To correct one error, the key is to locate the error position. This can be done by computing two *syndrome* vectors from the two parity rows. Since the error is a column error, it is natural to compute the syndromes with respect to *columns* rather than with respect to *rows* as in the encoding procedure. Once the error location is found, the value of the error can be easily computed along the diagonals of either slope.

Suppose $R = [r_{i,j}]_{0 \le i,j \le n-1}$ is the error-corrupted array. Construct two arrays $U = [u_{i,j}]_{0 \le i,j \le n-1}$ and $V = [v_{i,j}]_{0 \le i,j \le n-1}$ from R, where for $0 \le j \le n-1$,

$$u_{i,j} = v_{i,j} = r_{i,j}, \quad 0 \le i \le n-3 \tag{2.4}$$

$$u_{n-2,j} = r_{n-2,j}, v_{n-2,j} = r_{n-1,j} \tag{2.5}$$

$$u_{n-1,j} = v_{n-1,j} = 0 \tag{2.6}$$

i.e., U and V are constructed by copying the $n-1$ information rows and parity rows accordingly from R, then adding an imaginary 0-row at the last row. From U and V, compute two *syndrome* vectors S_0 and S_1 as follows:

$$S_0[i] = \sum_{k=0}^{n-1} u_{i+k,k} \tag{2.7}$$

$$S_1[i] = \sum_{k=0}^{n-1} v_{i-k,k} \tag{2.8}$$

all indices above are mod n.

It is easy to see that the two syndrome vectors are the column checksums along the diagonals of slope 1 and -1 respectively, and that they should be all-zero vectors if there is no error in the array R. If there is one error column in the array R, then the two syndromes are just cyclicly-shifted versions of the error vector with respect to the position of the error column. Thus the location of the error column can be determined simply by a cyclic equivalence test, which tests if one vector is equal to some cyclic shift of another vector. The following example shows how a single error column is reflected in two syndromes for an X-Code of size 5.

Example 2.3 *Syndrome Computation for a* 5×5 *X-Code*

Suppose the 3rd column is an error column, then the two syndrome vectors (S_0 and S_1 respectively) and their corresponding error arrays are as in Fig. 2.1.

$$\begin{array}{|c|c|c|c|c||c|} \multicolumn{6}{r}{S_0} \\ \hline 0 & 0 & 0 & e_0 & 0 & e_3 \\ \hline 0 & 0 & 0 & e_1 & 0 & 0 \\ \hline 0 & 0 & 0 & e_2 & 0 & e_0 \\ \hline 0 & 0 & 0 & e_3 & 0 & e_1 \\ \hline 0 & 0 & 0 & 0 & 0 & e_2 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|c|c||c|} \multicolumn{6}{r}{S_1} \\ \hline 0 & 0 & 0 & e_0 & 0 & e_2 \\ \hline 0 & 0 & 0 & e_1 & 0 & e_4 \\ \hline 0 & 0 & 0 & e_2 & 0 & 0 \\ \hline 0 & 0 & 0 & e_4 & 0 & e_0 \\ \hline 0 & 0 & 0 & 0 & 0 & e_1 \\ \hline \end{array}$$

Figure 2.1: Syndrome computation for a 5×5 X-Code

The two syndromes are actually just the original error column vector (cyclic-)shifted in two different directions for the same number of positions. When they are shifted back, they differ in at most one position; the number of the positions shifted gives the location of the error column. □

The above example almost gives the decoding algorithm for one error correction. A formal algorithm for correcting one error is

Algorithm 2.2 Correcting One Error

Compute two syndrome vectors S_0 and S_1 from the possibly-error-corrupted array R according to the equations Eq. (2.4) through Eq. (2.8). If the two syndromes are both all-zero vectors, then there is no error in the array R; otherwise if there exists such an i that after cyclically down-shifting S_0 by i positions and cyclically up-shifting S_1 by i positions, the first $n-2$ components of the two shifted vectors are equal and the last components of both are zeros then the ith column of the array R is an error column. If no such an i exists, then there is more than one error column in the array R.

A pseudo-code description of the algorithm is as follows:

1. Compute two syndrome vectors S_0 and S_1 from the possibly-error-corrupted array R according to the equations Eq. (2.4) through Eq. (2.8);

2. $i = 0$;

3. **If** $S_0[0..n-3] == S_1[0..n-3]$ **And** $S_0[n-1] == S_1[n-1] == 0$ **Then**

   The error position is the ith column, and the error value is

   $E = (S_0[0], S_0[1], \cdots, S_0[n-3], S_0[n-2], S_1[n-1])$;

   Else If $i == n$ **Then**

   Declare decoding failure : more than one error occurred;

   Else

   $S_0 = S_0^{(1)}, S_1 = S_1^{(-1)}$;

   $i++$;

   Goto 3;

   End If

□

In the above algorithm, for a vector V, denote its transpose V^T; let

$$V = (V[0], V[1], \cdots, V[n-1])^T,$$

then $V^{(1)}$ (or $V^{(-1)}$) is the *down-* (or *up-*) shifted vector from V, i.e.,

$$V^{(1)} = (V[n-1], V[0], \cdots, V[n-2])^T,$$

and

$$V^{(-1)} = (V[1], \cdots, V[n-1], V[0])^T;$$

also $V^{(i)} = (V^{(i-1)})^{(1)}$, $V^{(-i)} = (V^{-(i-1)})^{(-1)}$.

Before proving the correctness of the algorithm, we give a numerical example.

Example 2.4 *Correcting One Error of a* 5×5 *X-Code*

Suppose the possibly-error-corrupted array R is :

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

then U and V, the two constructed arrays from R, and their corresponding syndromes S_0 and S_1 are shown in Fig. 2.2.

U					S_0
0	0	0	1	0	1
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	0	0	0

V					S_1
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	0	0

Figure 2.2: Correcting one error of a 5×5 X-Code

Repeat Step 3 of the algorithm until $i = 3$, then we get $S_0 = (1,0,0,1,0)^T$ and $S_1 = (1,0,0,0,0)^T$, so $S_0[0..2]$ equals to $S_1[0..2]$, and $S_0[4] = S_1[4] = 0$. Thus we declare that the error occurs at the 3rd column and that the error value is $E = (1,0,0,1,0)$, i.e., the uncorrupted array should be an all-zero array. $\Box$

Now we give a correctness proof of the algorithm.

Proof: If one error occurs at the ith column, and its value is $e = (e[0], e[1], \cdots, e[n-2], e[n-1])^T$, then the two syndromes (Eq. (2.4) through Eq. (2.7)) are:

$$S_0 = ((e[0], \cdots, e[n-3], e[n-2], 0)^T)^{(-i)} \tag{2.9}$$

$$S_1 = ((e[0], \cdots, e[n-3], e[n-1], 0)^T)^{(i)} \tag{2.10}$$

thus

$$S_0^{(i)} = (e[0], \cdots, e[n-3], e[n-2], 0)^T \tag{2.11}$$

$$S_1^{(-i)} = (e[0], \cdots, e[n-3], e[n-1], 0)^T \tag{2.12}$$

Since the X-Code is an MDS code of column distance 3, it can correct one error, which means the location of a single column error can always be found unambiguously. A unique i can be found such that the two shifted syndrome vectors may differ only in the second last component, and their last components are both 0 (Eq. (2.11) and Eq. (2.12)). Once the error location i is found, the error value is obtained directly from Eq. (2.11) and Eq. (2.12). $\square$

The above algorithm needs $2n(n-2)$ additions to compute the two syndrome vectors, and on average n cyclic equivalence test operations to get the error location.

2.4 Summary

The X-Code, a new class of $n \times n$ *MDS* array codes of distance 3, is presented in this chapter. The significant difference of these codes from all other known array codes is that the parity (redundancy) symbols are placed in two independent rows rather than columns. Additionally, the X-Code has a very simple geometrical structure. Encoding and decoding of the code may be accomplished using only additions (*XOR*s). We have proven that the X-Code is *MDS if and only if* n is prime. For all prime numbers n, the X-Code achieves the lower bound of the update complexity. It also has balanced computation at each column, which might be very helpful in storage systems and distributed computing systems. Finally decoding algorithms for correcting two erasures or one error are given.

One future research problem is to find new MDS codes with optimal update complexity 1) for each positive integer length rather than only for prime lengths, and 2) for distance greater than 3. Our preliminary research shows that only for a few lengths n can the X-Code be easily extended to have larger distance by simply using more parity rows and taking more slopes; in general this is not the case. Extended diagonals, i.e., a set of symbols not necessarily on a straight line of some fixed slope, might be helpful in extending the X-Code to both more general lengths and distances.