Reed-Solomon Codes: Mechanism, Improvement and Application

Fan Jue (A0221578B), Tian Xiao (A0220592L)

Abstract

In coding theory, Reed-Solomon codes are a family of codes with superior theoretical properties and extensive empirical usage. This report provides a detailed yet intuitive explanation of Reed-Solomon codes in three aspects: the mechanism behind Reed-Solomon codes, the improvements made to Reed-Solomon codes to address their limitations and some real-life applications of Reed-Solomon codes.

1 Introduction

Reed-Solomon codes are an important and well-studied family of error-correcting codes that can detect and correct errors in received codewords. Reed-Solomon codes were first introduced by Irving S. Reed and Gustave Solomon in 1960 [25] and they have a number of desirable information-theoretic properties that make them superior to other codes. Throughout the years, many advancements have been forwarded in this field and Reed-Solomon codes have also proven their usefulness in real-life applications, from the ubiquitous QR codes to the Hubble Space Telescope in the outer space. In this report, we first explain the mechanism behind Reed-Solomon codes, including some important mathematical foundations and detailed encoding and decoding procedures. We also evaluate Reed-Solomon codes based on their desirable properties and possible limitations. Then we introduce several important improvements made to Reed-Solomon codes in order to address these limitations. Lastly, we exemplify some major applications of Reed-Solomon codes to illustrate why Reed-Solomon codes are exceptionally useful in real life.

2 Mechanism

The intuition behind Reed-Solomon codes is simple: Every polynomial of degree up to $k-1$ can be uniquely determined by any k points that it passes through. Therefore, to encode a message of length k, we can interpret it as the coefficients of a polynomial and use the coordinates of the k points it passes through as the codeword. Alternatively, we may also do this in a reversed manner where messages are interpreted as coordinates of points while coefficients of the polynomial are used as codewords. With this intuition in mind, we shall now formally study the mechanism of Reed-Solomon codes.

2.1 Galois Field

Before we introduce Reed-Solomon codes, it is important to explain the definition and some desirable properties of a mathematical structure that is used extensively in Reed-Solomon codes: the Galois field $\mathrm{GF}(n^k)$, where n is a **prime** number and $k \leq n$ is a positive integer.

2.1.1 The Galois Field $\mathrm{GF}(n)$ and Its Primitive Element

We start from a simple Galois field $\mathrm{GF}(n)$:

Definition 1. A Galois field $\mathrm{GF}(n)$ is a finite set S of size n associated with two defined operations: addition $(+)$ and multiplication $(\cdot)$, where the two operations satisfy the Field axioms [21].

For example, $\mathrm{GF}(2)$ can be defined as the set $\{0, 1\}$ with addition defined as $a + b \coloneqq (a + b) \bmod 2$ and multiplication defined as $a \cdot b \coloneqq (a \cdot b) \bmod 2$, where the $+$, $\cdot$ and mod to the right of $\coloneqq$ refer to the normal arithmetic operators on integers. For example, in $\mathrm{GF}(2)$, $1 + 1 = 0$ and $1 \cdot 1 = 1$.

Every Galois field $\mathrm{GF}(n)$ where n is a prime number has a *primitive element* [8], that is, there exists an element $\rho \in \mathrm{GF}(n)$ such that for any non-zero element $x \in \mathrm{GF}(n)$, $x = \rho^i$ for some positive integer $i < n$. Note that the definition of exponentiation here corresponds to the definition of multiplication in $\mathrm{GF}(n)$: $\rho^i \coloneqq \rho \cdot \rho \cdot \cdots \cdot \rho$ for i times. Since one non-zero element can be generated each time, it will take $n - 1$ times for all non-zero elements to be generated, hence $\rho^{n-1} = 1$ and $\rho^n = \rho$. This also implies that $0, \rho, \rho^2, \cdots, \rho^{n-1} = 1$ are pairwise distinct since otherwise a loop will be formed. For example, 2 is a primitive element for $\mathrm{GF}(3)$ since $2^1 = 2$ and $2^2 = 2 \cdot 2 = 1$. However, 2 is not a primitive element for $\mathrm{GF}(7)$ since $2^i, i \in \mathbb{Z}^+$ will be trapped in the cycle $\{2, 4, 1, 2, 4, 1, \cdots\}$. Therefore, any $\mathrm{GF}(n)$ can be written alternatively as the set $\{0, \rho, \rho^2, \cdots, \rho^{n-1} = 1\}$ given a primitive element ρ.

2.1.2 Polynomials over Galois Field: $\mathrm{GF}(n^k)$

Informally, we define $\mathrm{GF}(n^k)$ over a set of polynomials of order up to $k - 1$: $P(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}$, where $m_0, m_1, \cdots, m_{k-1} \in \mathrm{GF}(n)$ and $x \in \mathrm{GF}(n)$. For example, $x^2 + 1$ is an instance of $\mathrm{GF}(5^3)$; $0.5x^2 + 0.5$ is not an instance of $\mathrm{GF}(5^3)$. We then proceed to define addition and multiplication over the set of polynomials:

- Addition $(+)$: The addition of two polynomials in $\mathrm{GF}(n^k)$ is simply to sum up the coefficients to the same term following the addition defined in $\mathrm{GF}(n)$. For example, in $\mathrm{GF}(2^3)$, $(x^2 + x) + (x + 1) = (1 + 0)x^2 + (1 + 1)x + (0 + 1) = x^2 + 1$.
- Multiplication $(\cdot)$: To define the multiplication of two polynomials $P_1(x) \cdot P_2(x)$ in $\mathrm{GF}(n^k)$, we specify two cases:
    1. **The sum of order of P_1 and order of P_2 does not exceed $k-1$.** In this case, the coefficient of ρ $(p \leq k-1)$ in the product is equal to $\sum\limits_{i=0}^{p} (c_{1,i} \cdot c_{2,p-i})$, where $c_{1,i}$ refers to the coefficient to x_i in P_1, $c_{2,p-i}$ refers to the coefficient to x_{p-i} in P_2 and certainly the summation $\sum$ is based on addition in $\mathrm{GF}(n)$ and multiplication $\cdot$ is that of $\mathrm{GF}(n)$. This seemingly complicated process is actually what we are familiar with. For example, in $\mathrm{GF}(5^3)$, $(x+3)\cdot(x+3) = (1\cdot1)x^2 + (1\cdot3+3\cdot1)x + (3\cdot3) = x^2 + x + 4$.
    2. **The sum of order of P_1 and order of P_2 exceeds $k - 1$.** The issue with this case is that the order of the product may be larger than $k - 1$, which contradicts the definition of $\mathrm{GF}(n^k)$.

To resolve this, we need to introduce a property called irreducible: A polynomial in GF(n^k) is said to *irreducible* if it cannot be written as the product of any two polynomials that are not 0 or 1. For example, the polynomial $x^2 + x + 4$ in the above example is not irreducible in GF(5^3) since it can be written as the product of $x + 3$ and $x + 3$. Let $D(x)$ be an irreducible polynomial of order $k - 1$. We can now define the product of $P_1(x)$ and $P_2(x)$ as the result from Case 1 modulo $D(x)$, where the remainder can be found through techniques such as long division. For example, $D(x) = x^2 + x + 1$ is an irreducible polynomial in GF(5^3) and we have $(x^2 + 1) \cdot (x + 1) = (x^3 + x^2 + x + 1) \operatorname{\texttt{mod}} (x^2 + x + 1) = 1$.

It has been proven that GF(n^k) with the above definitions of addition and multiplication is indeed a Galois field [16]. There is a particular reason why Galois fields GF(n) and GF(n^k) are particularly suitable for error-correcting codes: Let ρ be a primitive element of GF(n) and consider a particular polynomial $P(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1} \in$ GF(n^k). We know that x can take values from the set $S = \{0, \rho, \rho^2, \cdots, \rho^{n-1}\}$. When we plug in the values $0, \rho, \rho^2, \cdots, \rho^{n-1}$ into the polynomial $P(x)$, we have

$$\begin{aligned} P(0) &= m_0; \\ P(\rho) &= m_0 + m_1\rho + m_2\rho^2 + \cdots + m_{k-1}\rho^{k-1}; \\ P(\rho^2) &= m_0 + m_1\rho^2 + m_2\rho^4 + \cdots + m_{k-1}\rho^{2k-2}; \\ &\cdots \\ P(\rho^{n-1}) &= m_0 + m_1\rho^{n-1} + m_2\rho^{2n-2} + \cdots + m_{k-1}\rho^{(k-1)(n-1)}. \end{aligned} \tag{1}$$

Note that the above n equations are all **distinct** since $0, \rho, \rho^2, \cdots, \rho^{n-1}$ are pairwise distinct by definition of primitive elements. Suppose that we choose any k distinct instances $\alpha_0, \alpha_1, \cdots, \alpha_{k-1} \in$ GF(n) and evaluate $P(\alpha_i)$ for $i = 0, 1, \cdots, k - 1$, then these k evaluations will correspond to k rows from Equation (1). We can write the k rows in the form of a linear system:

$$\begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{k-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{k-1} & \alpha_{k-1}^2 & \cdots & \alpha_{k-1}^{k-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{k-1} \end{bmatrix} = \begin{bmatrix} P(\alpha_0) \\ P(\alpha_1) \\ \vdots \\ P(\alpha_{k-1}) \end{bmatrix}. \tag{2}$$

We know that the above equation has a unique solution if and only if the first matrix is non-singular, and it is indeed non-singular because it is in the form of a Vandermonde matrix [17]. Therefore, whenever we evaluate a predetermined polynomial using each of $0, \rho, \rho^2, \cdots, \rho^{n-1}$ as input and suppose we are evaluating correctly, we can always reversely find the polynomial from any k pairs of inputs and evaluations. Moreover, even if some of the evaluations are wrong, as long as at least k out of the n evaluations are correct, we still have a chance to reversely find the polynomial. This important property fuels the invention of the encoding and decoding process of Reed-Solomon codes in the sections below.

2.2 Encoding

Let $\mathcal{X}$ denote the set of all symbols (i.e. alphabet) that we are going to use. Let n be the smallest prime number that is larger than $\mathcal{X}$ and let $\mathrm{GF}(n)$ be a Galois field. Let ρ be a primitive element of $\mathrm{GF}(n)$ as defined in Section 2.1.1. For each message block $m = (m_0, m_1, \cdots, m_{k-1}) \in \mathrm{GF}(n)^k$ (i.e. m is a k-tuple defined over $\mathrm{GF}(n)$), we can define a polynomial in $\mathrm{GF}(n^k)$ based on it:

$$P(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}. \tag{3}$$

Note that $P(x)$ is a polynomial of degree at most $k-1$. We can then encode the message m by evaluating the polynomial $P(x)$ at $\alpha_0 = 0, \alpha_1 = \rho, \alpha_2 = \rho^2, \cdots, \alpha_{n-1} = \rho^{n-1}$, where $\{\alpha_0, \alpha_1, \cdots, \alpha_{n-1}\}$ is known as the set of *evaluation points*[1]. The encoding of m is the tuple of evaluations of $P(\alpha)$ at all evaluation points, i.e.

$$\mathrm{RS}(m) = \big(P(\alpha_0), P(\alpha_1), ..., P(\alpha_{n-1})\big). \tag{4}$$

Alternatively, we can also represent Reed-Solomon codes as a result of matrix multiplication:

$$\mathrm{RS}(m)^\top = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{k-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{k-1} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{k-1} \end{bmatrix}, \tag{5}$$

where the first matrix is commonly known as the Vandermonde matrix, denoted as

$$V_{\alpha_0,\alpha_1,\cdots,\alpha_{n-1}} = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{k-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{k-1} \end{bmatrix}. \tag{6}$$

For simplicity, let V denote $V_{\alpha_0,\alpha_1,\cdots,\alpha_{n-1}}$ for a fixed Galois field. It is clear that $V^\top$ is the generator matrix of Reed-Solomon codes since for any message block m we have $\mathrm{RS}(m) = mV^\top$.

2.2.1 Properties of Reed-Solomon Codes

Proposition 1. *Reed-Solomon codes are linear codes.*

Proof. Linear codes are defined such that any linear combination of codewords is also a codeword. Since Galois fields are closed under addition and scalar multiplication, we have for any $m, m' \in \mathrm{GF}(n)^k$, $m + m' \in \mathrm{GF}(n)^k$ and for any $a \in \mathbb{R}$, $am \in \mathrm{GF}(n)^k$. Hence it follows that $\mathrm{RS}(m) + \mathrm{RS}(m') = mV^\top + m'V^\top = (m+m')V^\top = \mathrm{RS}(m+m')$, which is the codeword for $m + m'$. We also have for any $a \in \mathbb{R}$, $a\mathrm{RS}(m) = a\left(mV^\top\right) = (am)V^\top = \mathrm{RS}(am)$, which is the codeword for am. Therefore, Reed-Solomon codes are linear codes of dimension k over $\mathrm{GF}(n)$.

[1]Note that some Reed-Solomon codes also allow arbitrary values of n and evaluation points and their correctness has been proven. We focus on the basic case in this report.

Proposition 2. *Reed-Solomon codes have minimum distance $d = n - k + 1$.*

Proof. The Fundamental Theorem of Algebra states that a non-zero polynomial of degree $k - 1$ over field $\mathrm{GF}(n)$ has at most $k - 1$ distinct roots in $\mathrm{GF}(n)$ [6]. From Equation (3), $P(x)$ is a polynomial of degree at most $k - 1$. Following the Fundamental Theorem of Algebra, it must have at most $k - 1$ roots. In other words, there are at most $k - 1$ zeros in the Reed-Solomon codeword $\mathrm{RS}(m)$ for any message block m.

Recall that the *Hamming distance* between two codewords refers to the number of positions in which they differ and the *Hamming weight* of a codeword refers to the Hamming distance between the codeword and an all-zero codeword [27]. With the reasons above, the Hamming weight of a Reed-Solomon codeword is at least $n-(k-1) = n-k+1$. Recall also that the *minimum distance* of a codebook refers to the minimum Hamming distances between any pair of codeword in the codebook [27]. As Reed-Solomon Codes are linear codes by Proposition 1, the minimum distance equals the minimum Hamming weight, hence is at least $n-k+1$.

The Singleton bound for linear codes states that every linear code of minimum distance d and dimension k satisfy $d \leq n - k + 1$ [28]. Therefore the minimum distance d is bounded both above and below by $n - k + 1$. Hence we complete the proof.

Since Reed-Solomon codes achieve equality in the Singleton bound, they are a type of maximum distance separable (MDS) codes [28]. This characteristic makes them the codes with the optimal error-detecting and error-correcting capabilities for fixed n and k. However, the use of Reed-Solomon codes can be limited by its large alphabet size in real applications and this will be further explained in Subsection 2.4.

2.2.2 Encoding with Lagrange Interpolation

The original encoding procedure as described above does not embed the symbols of input message block $m = (m_0, m_1, \cdots, m_{k-1})$ in the output codeword $\mathrm{RS}(m)$, thus it is a *non-systematic* code. Encoding with Lagrange interpolation, on the other hand, provides an alternative way to produce Reed-Solomon codes that are systematic. Recall that we have a set of evaluation points $\{\alpha_0, \alpha_1, \cdots, \alpha_{n-1}\}$. We now define the polynomial $P(x)$ alternatively as the unique polynomial of degree less than k such that

$$P(\alpha_i) = m_i, \; \forall i \in \{0, 1, ..., k-1\}. \tag{7}$$

The existence and uniqueness of such a polynomial are proved in [29]. We can thus find the unique $P(x)$ passing through points $(\alpha_0, m_0), (\alpha_1, m_1), ..., (\alpha_{k-1}, m_{k-1})$ using Lagrange interpolation. Note that Equation (7) uses only the first k out of n evaluation points. We then evaluate the polynomial $P(x)$ at the remaining evaluation points $\alpha_k, \alpha_{k+1}, \cdots, \alpha_{n-1}$. With these evaluations, the message block m is encoded as

$$\mathrm{RS}(m) = \big(m_0, m_1, ..., m_{k-1}, P(\alpha_k), P(\alpha_{k+1}), \cdots, P(\alpha_{n-1})\big). \tag{8}$$

This encoding procedure is systematic because the first k entries of each codeword correspond to the message m. Similar to the original encoding procedure, we can write the generator matrix as follows:

$$V_{\alpha_1,\alpha_2,\ldots,\alpha_n} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & \alpha_k & \alpha_k^2 & \cdots & \alpha_k^{k-1} \\ 1 & \alpha_{k+1} & \alpha_{k+1}^2 & \cdots & \alpha_{k+1}^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{k-1} \end{bmatrix}. \tag{9}$$

Thus, again for any message block m we have $\text{RS}(m) = mV^\top$.

2.3 Decoding

Suppose that we are given the codeword $\text{RS}(m) = (c_0, c_1, ..., c_{n-1})$ with possible errors and the length of original message block k. Suppose also that the set of evaluation points $S = \{\alpha_0, ..., \alpha_{n-1}\}$ is known. Now we would like to reversely find out the polynomial $P(x)$ and the original message block m. Note that the following decoders are applicable to both original encoding and encoding with Lagrange interpolation.

2.3.1 Theoretical Decoder

Theoretically, we can repeatedly take k out of the n evaluations and perform Lagrange interpolation to produce the polynomial that satisfy the k evaluations. The most frequently occurring polynomial, $P^*(x)$, is believed to eliminate errors in the received codeword [25]. Upon obtaining $P^*(x)$, it becomes straightforward to get the original message block m: For original encoding, we take the coefficients of $P^*(x)$ as m; for encoding with Lagrange interpolation, we take the results of $P^*(\alpha_i), i = 0, 1, \cdots, k-1$ as m.

Suppose the number of errors is ϵ in the n evaluations. The number of k-combinations of evaluations that provides error-free polynomials is thus equal to $\binom{n-\epsilon}{k}$. Meanwhile, the maximum number of a particular erroneous polynomial is at most $\binom{\epsilon+k-1}{k}$, since any erroneous polynomial can pass through at most $k-1$ out of the $n-\epsilon$ error-free evaluations at their corresponding evaluation points. In order for the most frequently occurring polynomial $P^*(x)$ to be the error-free one, we have

$$\begin{aligned} \binom{n-\epsilon}{k} &> \binom{\epsilon+k-1}{k} \\ n-\epsilon &> \epsilon+k-1 \\ \epsilon &< \frac{n-k+1}{2}. \end{aligned} \tag{10}$$

Therefore, a Reed-Solomon code with fixed n and k can correct up to $\lfloor \frac{n-k}{2} \rfloor$ errors.

Despite its technical simplicity and high error-correcting capacity, decoding Reed-Solomon codes require $\binom{n}{k}$ possible combinations of evaluations and thus potential polynomials that we need to compute. This makes the computation prohibitively costly even in simple cases. Therefore, this decoding method is impractical in real applications.

2.3.2 Berlekamp-Welch Decoder

The Berlekamp-Welch algorithm [31] is a more efficient way to decode Reed-Solomon codes. On top of correcting errors and recovering the original polynomial $P(x)$, the Berlekamp-Welch algorithm also produces an error polynomial which can locate the errors in the codeword.

Suppose again that the number of errors is ϵ in the n evaluations. Note that an error occurs at evaluation point α_i if $c_i \neq P(\alpha_i)$. We define an error polynomial $E(x) = e_0 + e_1 x + e_2 x^2 + \cdots + e_\epsilon x^\epsilon$ of order ϵ such that it must satisfy

$$c_i E(\alpha_i) = E(\alpha_i) P(\alpha_i),\ \forall\, i \in \{0, 1, ..., n-1\}. \tag{11}$$

When there is no error at α_i, the above equation obviously holds since $c_i = P(\alpha_i)$; when there is an error at α_i, we need $E(\alpha_i) = 0$ for the above equation to hold. Since all the α_i's are distinct and the polynomial $E(x)$ has at most ϵ roots (because it is of order ϵ), we are certain that whether there is an error at α_i is sufficiently indicated by whether $E(\alpha_i) = 0$.

For simplicity, define a new polynomial $Q(x) = E(x)P(x)$ which is a $(\epsilon + k - 1)$-th order polynomial. Let $Q(x) = q_0 + q_1 x + q_2 x^2 + \cdots + q_{\epsilon+k-1} x^{\epsilon+k-1}$. Note that by Equation (11) $Q(\alpha_i) = 0$ if there is an error at α_i. Additionally, we set the constraint that $e_\epsilon = 1$ and by substituting the terms in Equation (11) we have

$$\begin{aligned} c_i(e_0 + e_1 x + e_2 x^2 + \cdots + e_\epsilon x^\epsilon) &= q_0 + q_1 x + q_2 x^2 + \cdots + q_{\epsilon+k-1} x^{\epsilon+k-1} \\ c_i(e_0 + e_1 x + e_2 x^2 + \cdots + e_{\epsilon-1} x^{\epsilon-1}) - (q_0 + q_1 x + q_2 x^2 + \cdots + q_{\epsilon+k-1} x^{\epsilon+k-1}) &= -c_i x^\epsilon. \end{aligned} \tag{12}$$

Note that we can write such an equation for each of the n evaluations on $\alpha_i, i = 0, 1, \cdots, n-1$. The set of n such equations form a linear system for us to solve the values of e_i's and q_i's and we will know the exact form of $E(x)$ and $P(x)$ from there. Algorithm 1 shows the pseudocode of the Berlekamp-Welch algorithm.

To better demonstrate the algorithm, we provide the following example where $n = 5$ and $k = 3$. Suppose that the message block to be encoded is $(1, 4, 2)$ and the evaluation points are $\{0, 1, 2, 3, 4\}$. Using the original encoding, we use the polynomial $P(x) = 1 + 4x + 2x^2$ and obtain the codeword $(1, 2, 2, 1, 4)$. Assume error occurs at α_3 and the received codeword is $(1, 2, 1, 1, 4)$. Let $\epsilon = 1$ and we have the following linear system based on Equation (12):

$$\begin{bmatrix} c_0 & -1 & -\alpha_0 & -\alpha_0^2 & -\alpha_0^3 \\ c_1 & -1 & -\alpha_1 & -\alpha_1^2 & -\alpha_1^3 \\ c_2 & -1 & -\alpha_2 & -\alpha_2^2 & -\alpha_2^3 \\ c_3 & -1 & -\alpha_3 & -\alpha_3^2 & -\alpha_3^3 \\ c_4 & -1 & -\alpha_4 & -\alpha_4^2 & -\alpha_4^3 \end{bmatrix} \begin{bmatrix} e_0 \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} -c_0\alpha_0 \\ -c_1\alpha_1 \\ -c_2\alpha_2 \\ -c_3\alpha_3 \\ -c_4\alpha_4 \end{bmatrix}. \tag{13}$$

Note that the operations follow the arithmetic of GF(5) (e.g. $-1 = 4$). By substituting the values of

Algorithm 1 Berlekamp-Welch Algorithm

Inputs: The received codeword $c = (c_0, c_1, ..., c_{n-1})$; the set of evaluation points $S = \{\alpha_0, \alpha_1, ..., \alpha_{n-1})$; the length of original message block k.

```
1:  $\epsilon \leftarrow \lfloor \frac{n-k}{2} \rfloor$
2:  while $\epsilon > 0$ do                                   ▷ need to find $\epsilon$ by trail-and-error
3:      $A \leftarrow$ linear system formed by Equation (12) for all $\alpha_i, i = 0, 1, \cdots, n-1$
4:      if $A$ can be solved then
5:          $Q(x), E(x) \leftarrow$ solution to $A$
6:          if $Q(x)/E(x)$ has remainder $\neq 0$ then
                return Uncorrectable error detected!
7:          end if
            $P(x) \leftarrow Q(x)/E(x)$
8:          for $\alpha_i$ where $E(\alpha_i) = 0$ do
9:              $c_i \leftarrow P(\alpha_i)$                     ▷ to correct error at $\alpha_i$
10:         end for
            return $(c_0, c_1, ..., c_{n-1})$
11:     end if
12:     $\epsilon \leftarrow \epsilon - 1$
13: end while
14: return No error!
```

$\alpha_i, i = 0, 1, 2, 3, 4$, we arrive at the following linear system:

$$\begin{bmatrix} 1 & 4 & 0 & 0 & 0 \\ 2 & 4 & 4 & 4 & 4 \\ 1 & 4 & 3 & 1 & 2 \\ 1 & 4 & 2 & 1 & 3 \\ 4 & 4 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} e_0 \\ q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 3 \\ 2 \\ 4 \end{bmatrix}, \tag{14}$$

The solution to the linear system is

$$\begin{bmatrix} e_0 & q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\top = \begin{bmatrix} 3 & 3 & 3 & 0 & 2 \end{bmatrix}^\top. \tag{15}$$

Therefore, we have

$$\begin{aligned} E(x) &= 3 + x, \\ Q(x) &= 3 + 3x + 2x^3 \\ P(x) &= Q(x)/E(x) = 1 + 4x + 2x^2 \text{ with remainder } 0. \end{aligned} \tag{16}$$

As such, we reconstruct the original polynomial $P(x)$. Since $E(\alpha_2) = 3 + 2 = 0$, we can evaluate $P(\alpha_2) = 1 + 4 \cdot 2 + 2 \cdot 2^2 = 2$ to obtain the correct value at α_2: 2. Thus the corrected codeword is $(1, 2, 2, 1, 4)$.

The time complexity of Berlekamp-Welch algorithm depends on the algorithm we use to solve linear systems, which is at most $O(n^3)$. Therefore, it is much better than the theoretical decoder and hence much more common in practice.

2.3.3 Gao Decoder

The Gao decoder [11] is another efficient way to decode Reed-Solomon codes. Recall that we have a set evaluation points $\{\alpha_0, \alpha_1, \cdots, \alpha_{n-1}\}$ and a codeword $(c_0, c_1, \cdots, c_{n-1})$, hence there exists a unique polynomial $G_1(x)$ of order at most $n-1$ such that $G_1(\alpha_i) = c_i$ for $i = 0, 1, \cdots, n-1$. In Section 2.3.2 we define an error polynomial $E(x)$ and another polynomial $Q(x) = E(x)P(x)$, and from Equation (11) we have

$$G_1(\alpha_i)E(\alpha_i) = Q(\alpha_i),\ \forall i \in \{0, 1, \cdots, n-1\}. \tag{17}$$

In other words, the remainder of $G_1(x)E(x)$ divided by $(x - \alpha_i)$ is always equal to the remainder of $Q(x)$ divided by $(x-\alpha_i)$ given any α_i. Note that for any two $i, j \in \{0, 1, \cdots, n-1\}$ and $i \neq j$, $(x-\alpha_i)$ is *co-prime* to $(x - \alpha_j)$, that is, one cannot be written as a multiple of another. Thus, by Chinese Remainder theorem, $G_1(x)E(x)$ must be *congruent* to $Q(x)$ modulo the following polynomial $G_0(x)$, the product of the co-prime factors:

$$G_0(x) = \prod_{i=0}^{n-1}(x - \alpha_i). \tag{18}$$

That is, when we divide $G_1(x)E(x)$ and $Q(x)$ by $G_0(x)$ respectively, we always get the same remainder. Hence, when we divide $G_1(x)E(x) - Q(x)$ by $G_0(x)$, the remainder is equal to 0. In other words, there exists a polynomial $A(x)$, such that

$$\begin{aligned} G_1(x)E(x) - Q(x) &= A(x)G_0(x) \\ A(x)G_0(x) - E(x)G_1(x) &= -Q(x) \end{aligned} \tag{19}$$

It has been shown that $-Q(x)$ is a multiple of the greatest common divisor or $G_0(x)$ and $G_1(x)$ [11]. Hence, we can easily find possible $A(x)$, $E(x)$ and $Q(x)$ in Equation (19) through the Extended Euclidean algorithm. After that, we can compute $P(x) = Q(x)/E(x)$ similar to Section 2.3.2. Algorithm 2 shows the pseudocode of the Gao decoder.

Algorithm 2 Gao Decoder Algorithm

Inputs: The received codeword $c = (c_0, c_1, ..., c_{n-1})$; the set of evaluation points $S = \{\alpha_0, \alpha_1, ..., \alpha_{n-1})$; the length of original message block k.

```
1: $G_0(x) \leftarrow \prod_{i=0}^{n-1}(x - \alpha_i)$
2: $G_1(x) \leftarrow$ unique polynomial such that $G_1(\alpha_i) = c_i$ for $i = 0, 1, \cdots, n-1$    ▷ by Lagrange interpolation
3: while order of remainder $\geq \frac{n+k}{2}$ do    ▷ apply Extended Euclidean algorithm
4:     quotient $\leftarrow G_0(x)/G_1(x)$
5:     remainder $\leftarrow G_0(x) \bmod G_1(x)$
6:     $G_0(x) \leftarrow G_1(x)$
7:     $G_1(x) \leftarrow$ remainder
8: end while
9: $A(x), E(x), Q(x) \leftarrow$ result of Extended Euclidean algorithm
10: if $Q(x)/E(x)$ has remainder $\neq 0$ then
11:     return Uncorrectable error detected!
12: end if
13: $P(x) \leftarrow Q(x)/E(x)$
14: return $P(x)$
```

To better demonstrate this algorithm, we use the same setting as Section 2.3.2 where $n = 5$, $k = 3$, the message block to be encoded is $(1, 4, 2)$ and the evalution points are $\{0, 1, 2, 3, 4\}$. Suppose again that the corrupted codeword is $(1, 2, 1, 1, 4)$. We first find $G_0(x)$:

$$G_0(x) = (x - 0)(x - 1)(x - 2)(x - 3)(x - 4) = x^5 + 4x. \tag{20}$$

We then find $G_1(x)$ by solving the following linear system:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 & 1 \\ 1 & 3 & 4 & 2 & 1 \\ 1 & 4 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 4 \end{bmatrix}. \tag{21}$$

The solution to the linear system is

$$\begin{bmatrix} g_0 & g_1 & g_2 & g_3 & g_4 \end{bmatrix}^\top = \begin{bmatrix} 1 & 2 & 1 & 2 & 1 \end{bmatrix}. \tag{22}$$

Hence,

$$G_1(x) = x^4 + 2x^3 + x^2 + 2x + 1. \tag{23}$$

We then calculate the quotient and remainder of $G_0(x)/G_1(x)$:

$$\begin{array}{r r}
 & x + 3 \\ \cline{2-2}
x^4 + 2x^3 + x^2 + 2x + 1 \big) & x^5 + 0x^4 + 0x^3 + 0x^2 + 4x + 0 \\
 & x^5 + 2x^4 + \ \ x^3 + 2x_2 + \ \ x \quad\ \ \\ \cline{2-2}
 & 3x^4 + 4x^3 + 3x^2 + 3x + 0 \\
 & 3x^4 + \ \ x^3 + 3x^2 + \ \ x + 3 \\ \cline{2-2}
 & 3x^3 + 0x^2 + 2x + 2
\end{array}$$

Hence following the Extended Euclidean algorithm we have

$$\begin{aligned} G_0(x) - (x + 3)G_1(x) &= 3x^3 + 2x + 2 \\ G_0(x) + (4x + 2)G_1(x) &= 3x^3 + 2x + 2. \end{aligned} \tag{24}$$

Note that $(3x^3 + 2x + 2)/(4x + 2) = 1 + 4x + 2x^2$, which is the original polynomial $P(x)$. Following similar procedures to Section 2.3.2, we can get the corrected codeword $(1, 2, 2, 1, 4)$ from there.

The time complexity of Gao decoder is in general $O(n \log^2 n)$ with the help of Fast Fourier Transform (FFT) techniques [11]. However, it is well-known that FFT works well only for large n (e.g. $n \geq 1000$). Therefore, Gao decoders are more useful for large-scale Reed-Solomon codes with more errors while the Belerkamp-Welch algorithm is more suitable for small-scale Reed-Solomon codes (which are more common in practice).

2.4 Limitations

In this subsection, we evaluate the limitations of Reed-Solomon codes in terms of computational complexity and multiple-burst correction capability.

2.4.1 High Computational Complexity

The theoretical decoder of Reed-Solomon codes requires a combinatorial time complexity. Although the Berlekamp-Welch algorithm improves the time complexity to $O(n^3)$, it is still impractical in real applications such as deep-space communication that requires fast and real-time decoding. Gao decoder has a time complexity of $O(n \log^2 n)$, but it is even slower than Berlekamp-Welch algorithm decoding because it uses a very large constant factor for FFT. Moreover, when the message to encode is more complex, we need a larger alphabet size, thus a larger n and a larger Galois field. As a result, the computational cost of performing arithmetic operations in GF(n^k) would also be prohibitively expensive. To better make use of this powerful coding algorithm and enhance its scalability to encode more complex messages, many efforts have been invested in improving the computational complexity. Examples will be explained in Section 3.

2.4.2 Low Multiple-Burst Correction Capability

In real life, there are two types of errors that can occur in data transmission: random errors and burst errors. Random errors can typically be caused by natural noises or interference and are split evenly across the codewords, whereas burst errors refer to errors that occur in many consecutive positions in codewords. For example, it can be caused by physical damage to the data storage devices. Figure 1 illustrates the difference between the two types of errors.

It has been shown that Reed-Solomon codes are good at handling random errors and single burst errors, but not multiple burst errors [2]. However, multiple burst errors are common in some applications of Reed-Solomon codes, such as compact disks. Therefore, there is a need to make improvements to the multiple-burst correction capability of Reed-Solomon codes.

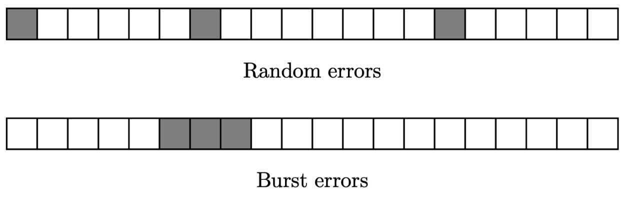

Figure 1: Different types of errors. The shaded squares refer to the positions where errors occur.

3 Improvement on Reed-Solomon Codes

To address the limitations above, a number of improvements were made over the original Reed-Solomon codes. In this section, we focus on three important improvements and discuss their mechanisms and advantages over the original Reed-Solomon codes.

3.1 Bose-Chaudhur-Hocquenghem (BCH) View of Reed-Solomon Codes

Bose-Chaudhur-Hocquenghem (BCH) codes [4, 5, 15] are another type[2] of error-correcting codes whose key idea is to use a *generator polynomial* (which we will further explain later). The **BCH view of Reed-Solomon codes** combines the idea of original Reed-Solomon codes and BCH codes to produce a new type of codes that are more flexible and efficient than the original Reed-Solomon codes.

Similar to the original Reed-Solomon codes, we define a polynomial in GF(n^k) using the message block $m = (m_0, m_1, \cdots, m_{k-1})$ as coefficients:

$$P(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}. \tag{25}$$

Recall that the Galois field GF(n) has a primitive element ρ. We define a generator polynomial as follows:

$$G(x) = (x-\rho)(x-\rho^2)\cdots(x-\rho^t) = g_0 + g_1 x + g_2 x^2 + \cdots + g_t x^t, \tag{26}$$

where $t \leq n-k$ is the error-correcting capacity that we can set. The encoding of m is simply the coefficients of $P(x)G(x)$ which is a polynomial of order up to $(k-1)+(n-k) = n-1$. For any error-free codeword, $G(\rho^i) = 0$ for $\rho = 1, 2, \cdots, t$; otherwise $G(\rho^i) \neq 0$ at some ρ^i and we call it a *syndrome* at ρ^i. We can easily correct the errors and get the original message block by "treating" such syndromes through the Petersen-Gorenstein-Zierler algorithm [13, 23] or Berlekamp-Massey algorithm [3, 20].

Note that the above encoding is not systematic as the input message block $m = (m_0, m_1, \cdots, m_{k-1})$ is not embedded into the codeword. To make it systematic, we wish to have a polynomial $B(x)$ such that some coefficients of $B(x)$ is equal to m_i for $i = 0, 1, \cdots, m-1$ and $B(x)$ should also be a multiple of the generator polynomial $G(x)$ so that we can still correct errors by treating syndromes. To achieve this, we define $B(x)$ as follows:

$$\begin{aligned} P'(x) &= P(x) \cdot x^t; \\ B(x) &= P'(x) - (P'(x) \bmod G(x)). \end{aligned} \tag{27}$$

Note that $P'(x) = m_0 x^t + m_1 x^{t+1} + \cdots + m_{k-1} x^{t+k-1}$ according to the definition above. Since $G(x)$ is of order t, $P'(x) \bmod G(x)$ has order at most $t-1$ which implies that coefficients to the terms $x^i, t \leq i \leq t+k-1$ are the same for $P'(x)$ and $B(x)$, hence the original message block $m = (m_0, m_1, \cdots, m_{k-1})$ exists in the coefficients of $B(x)$. Meanwhile, $P'(x)$ and $P'(x) \bmod G(x)$ are clearly congruent modulo $G(x)$, hence $B(x)$ is a multiple of $G(x)$. Therefore, the coefficients of $B(x)$ is a systematic encoding of the original message block m. Decoding of $B(x)$ is similarly done by treating syndromes with Petersen-Gorenstein-Zierler algorithm or Berlekamp-Massey algorithm.

BCH view of Reed-Solomon codes are more common in practice because of its efficient decoding: the Berlekamp-Massey algorithm takes $O(n^2)$ to decode, which is a significant improvement from $O(n^3)$ of the original view. Moreover, there have been a lot of works to reduce constant factors such as the number of arithmetic operations such as Berlekamp-Massey-Sakata algorithm [26] and Fast Parallel Berlekamp-Massey

[2]Sometimes, BCH codes are also viewed as subfield subcodes of Reed-Solomon codes [7].

algorithm [18]. On the other hand, BCH view of Reed-Solomon codes provides higher flexibility in adjusting the balance between computational complexity and error-correcting capacity. One can do so by adjusting the value of t in Equation (26): A larger t allows us to detect and correct more errors but makes the decoding process more computationally heavy, whereas a smaller t makes the decoding process more computationally efficient in the compensation of some error-correcting capacity.

3.2 Folded Reed-Solomon (FRS) Codes

We have introduced some limitations of Reed-Solomon codes caused by its large block size in Section 2.4.1. To address this issue, Folded Reed-Solomon (FRS) codes reduce block sizes by grouping the original codeword into a number of folds and the total number of folds is much smaller than the original length of codewords.

Suppose again that we have a Galois field $\mathrm{GF}(n)$, where $\rho \in \mathrm{GF}(n)$ is a primitive element. Recall that $\rho^{n-1} = 1$ and $\rho^n = \rho$. Let $t \geq 1$ be the *folding parameter* which represents the number of elements per fold and t divides $n-1$. Let $P(x)$ be the polynomial defined for Reed-Solomon codes in Equation (3) or Equation (7). We have known the codeword given by Reed-Solomon codes is

$$\mathrm{RS}(m) = \big(P(0), P(\rho), P(\rho^2), ..., P(\rho^{n-1})\big). \tag{28}$$

In FRS codes, we drop the evaluation point 0 such that all the remaining evaluation points can be expressed as a power of ρ. By grouping t consecutive elements in $\mathrm{RS}(m)$ together, we can arrive at the t-folded Reed-Solomon codes, $\mathrm{FRS}(m)$, which is a code of block size $\frac{n}{t}$:

$$\mathrm{FRS}(m) = \left(\begin{bmatrix} P(\rho) \\ P(\rho^2) \\ \vdots \\ P(\rho^t) \end{bmatrix}, \begin{bmatrix} P(\rho^{t+1}) \\ P(\rho^{t+2}) \\ \vdots \\ P(\rho^{2t}) \end{bmatrix}, \cdots, \begin{bmatrix} P(\rho^{n-t}) \\ P(\rho^{n-t+1}) \\ \vdots \\ P(\rho^{n-1}) \end{bmatrix} \right). \tag{29}$$

FRS codes can be decoded using an algorithm developed by Guruswami [14] in polynomial time.

Besides computational efficiency, FRS codes are also more capable of correcting burst errors as compared to Reed-Solomon codes because FRS codes correct errors in each sub-block independently.

FRS codes have another significant advantage over original Reed-Solomon codes or even any other codes in general: it has so far the optimal trade-off between coding rate and error correction radius. *Coding rate* in a systematic code refers to the ratio of message length to codeword length; *error correction radius* refers to the number of errors that a decoder can correct up to. Before FRS codes are invented, the best error correction radius is given a fixed coding rate R is equal to $1 - \sqrt{R}$[3], and FRS codes improve this number to $1 - R$ asymptotically.

[3]Actually this error correction radius is also achieveed by Reed-Solomon codes!

3.3 Cross-Interleaved Reed-Solomon Codes (CIRC)

Interleaving is a common technique to convert burst errors into the same patterns as random errors. The idea of interleaving is extremely simple yet practical: It shuffles the original codeword such that the burst errors are spread uniformly the entire codeword. For example, Figure 2 demonstrates how a 5×4 interleaver can redistribute the burst errors.

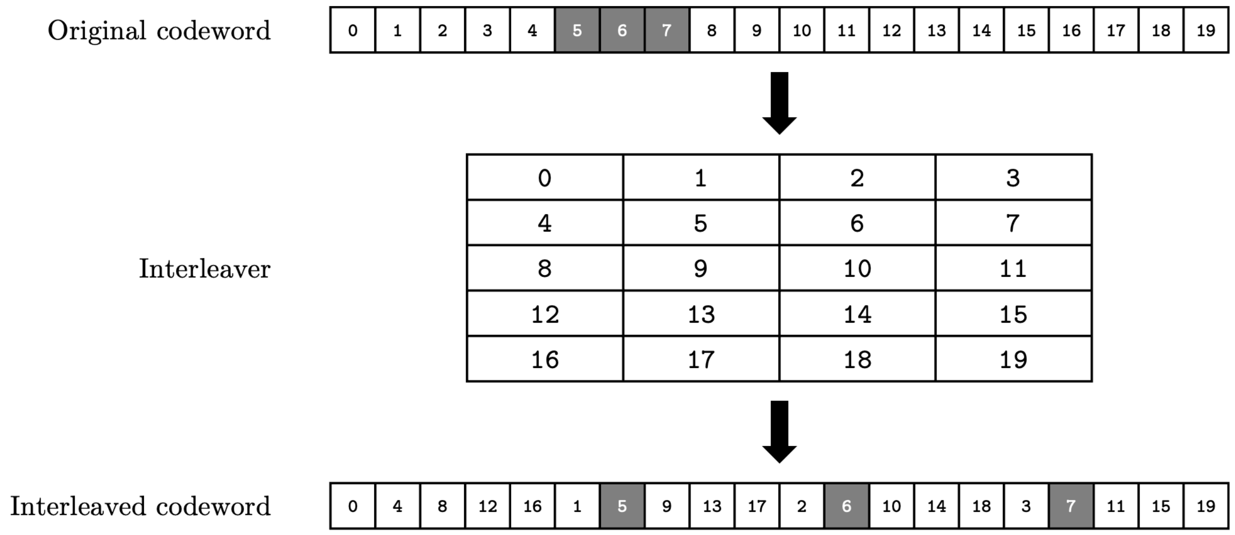

Figure 2: Different types of errors. The shaded squares refer to the positions where errors occur.

Since Reed-Solomon codes are very good at handling random errors, it is likely to perform well on both random errors and burst errors after the interleaving technique is used. Moreover, this technique does not have any significant time complexity to the encoding and decoding process, and hence is very commonly used in practice.

4 Applications of Reed-Solomon Codes

Because of its several superior properties, Reed-Solomon codes and their variants are commonly used in practice. In this section, we focus on three major applications of Reed-Solomon codes [33]. By analyzing the requirements and use cases in each application scenario, we explain why certain specific Reed-Solomon codes are used in each application.

4.1 Compact Discs (CDs)

Reed-Solomon codes play a crucial role in ensuring accurate data retrieval from compact discs (CDs), video compact discs (VCDs) and digital video discs (DVDs). We focus on the simplest one of them, CDs, which only store audio information, where Reed-Solomon codes were introduced in the consumer product for the first time. To produce a CD, sounds are first converted into a binary format through pulse-code modulation. The binary code is then encoded with **Eight-to-Fourteen Modulation (EFM) codes** and **CIRC codes** and stored in CDs. When we play a CD, decoders inside players work to decode the original message and convert it to sounds.

In CDs, data are stored as upwards bumps called *pits* (1) and downwards bumps called *lands* (0) on the disc surface (Figure 3), which can be read by a laser beam. Due to bandwidth limits, only a certain number of pits and lands can be written repetitively within a region. Hence, the raw messages need to be converted to codewords such that **the numbers of consecutive 0's and 1's are bounded**, which can be done using Run-Length Limited (RLL) codes. Therefore, we use a specific type of RLL codes, the **EFM codes** in CDs.

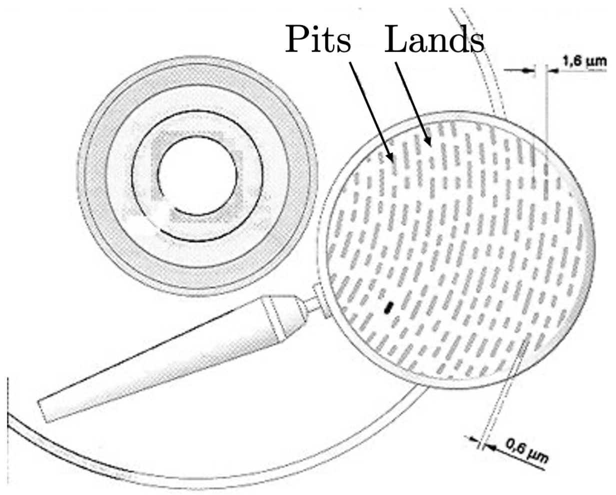

Figure 3: Illustration of a CD with pits and lands [24].

During the production of CDs, electrical noises or interference might occur and cause imperfections in the pits and lands; during the usage of CDs, players may also experience interference and inaccurately read from the pits and lands. Hence there might be random errors in the codewords that decoders receive. Meanwhile, CD owners may leave fingerprints or scratches on CDs which affect a concentrated region of data, leading to possible burst errors in the codewords. Therefore, the codes used in CDs must have a **high random error correctability**, **long burst error correctability** and **good ability to retain as much information as possible** even if the number of errors exceeds the error correction radius. Furthermore, since a disc can store limited data and is expected to be used to play audio fluently, the codes used in CDs should also have **low redundancy** and **efficient decoding algorithm**. As a result, **CIRC codes based on BCH view of Reed-Solomon codes** are chosen to be used CDs because they satisfy all these requirements, as we have explained in Section 3.1 and 3.3.

4.2 Bar Codes and QR Codes

Two-dimensional bar codes encode information in a matrix or grid of black and white squares which can be scanned using an imaging device. One popular example of two-dimensional bar codes is the Quick Response (QR) code, where black dots refer to 1's and white dots refer to 0's. The use of Reed-Solomon codes in QR codes can allow correct reading even if the QR codes are partially physically damaged or some pixels are wrongly captured by imaging devices.

To achieve this, a bar code needs to use some of its storage to store information (messages) and the other to store redundancies. The specific design and parameters of the Reed-Solomon code used in bar codes can vary depending on the specific application and the required level of error correction. In general, the Reed-Solomon codes, when applied in bar codes, work similarly to the parity error correction [27]. However, Reed-Solomon codes can correct much more errors as compared to the parity method based on the trade-off between coding rate and error correction radius introduced in Section 3.2. For example [19], a Version 3 QR code (29×29) can hold 70 bytes of information. A common scheme is to use 34 bytes to store messages and 36 bytes to store redundancies. In other words, the coding rate, in this case, is $\frac{34}{70}$. Based on the trade-off between coding rate and error correction radius introduced in Section 3.2, Reed-Solomon codes can ideally achieve an error correction radius of $1 - \sqrt{\frac{34}{70}} \approx 0.3$. In practice, the messages usually can be correctly retrieved when up to 25% of the codewords are corrupted, which is slightly lower than the ideal bound but still makes the QR codes highly resilient.

4.3 Deep-Space Telecommunication Systems

Reed-Solomon codes have been used extensively in deep-space telecommunication systems since the last century, such as the famous National Aeronautics and Space Administration (NASA) Voyager program [12] and NASA's Hubble Space Telescope [32]. Due to the vast distance between the Earth and spacecrafts, the signals sent from spacecrafts are weakened along the way and are highly susceptible to cosmic radiation and electromagnetic interference. Thus the received data are highly likely to be erroneous. Meanwhile, it is extremely time-expensive (up to months) to repeat the data collection and transmission process if the erroneous data cannot be recovered [10]. This drives the need to use error-correcting codes in deep-space telecommunication systems.

We study the Voyager spacecraft as an example [1]. Voyager adopted an optimized $(7, \frac{1}{2})$ convolutional code and later an additional $(255, 223)$ Reed-Solomon code[4]. A *convolutional code* [30] is another type of error-correcting code that works via convolution, the process of combining neighbouring data into one using a boolean polynomial function called *kernel*. Convolutional codes can be decoded via maximum-likelihood Viterbi decoder. The $(7, \frac{1}{2})$ convolutional code can correct up to $\lfloor \frac{10-1}{2} \rfloor = 4$ errors [22]. The $(255, 223)$ Reed-Solomon code, on the other hand, can correct up to $\lfloor \frac{255-223}{2} \rfloor = 16$ errors and it is actually a standard implementation of Reed-Solomon codes. Note that the Reed-Solomon code is concatenated to the initial convolutional code at a later stage through code concatenation [9], which enables us to append a more advanced code to an older code without major changes to the overall architecture of the system.

In conclusion, Reed-Solomon codes are a powerful tool in real-life applications where the correct information is highly important but data transmission tends to be erroneous and lossy.

[4]Note that $n = 255$ is not a prime number as in our definition. This is common in practice because the primeness of n is not a necessary condition for Reed-Solomon codes to work and our definition is a basic case.

5 Conclusion

Our report has illustrated the information-theoretic and practical advantages of Reed-Solomon codes through an in-depth study of their mechanisms. By reviewing how existing improvements have greatly extended the use cases of Reed-Solomon codes, our report has also demonstrated the potential for further improvements in future. In conclusion, Reed-Solomon codes are a remarkably powerful and important family of error-correcting codes.

References

[1] Kenneth S Andrews, Dariush Divsalar, Sam Dolinar, Jon Hamkins, Christopher R Jones, and Fabrizio Pollara. The development of turbo and ldpc codes for deep-space applications. *Proceedings of the IEEE*, 95(11):2142–2156, 2007.

[2] Elwyn R Berlekamp. Elongated burst trapping, April 10 1990. US Patent 4,916,702.

[3] ER Berlekamp. Binary bch codes for correcting multiple errors. *Algebraic Coding Theory*, 1968.

[4] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. Further results on error correcting binary group codes. *Information and Control*, 3(3):279–290, 1960.

[5] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and control*, 3(1):68–79, 1960.

[6] Lindsay N Childs and Lindsay N Childs. The fundamental theorem of algebra. *A Concrete Introduction to Higher Algebra*, pages 253–276, 1995.

[7] P. Delsarte. On subfield subcodes of modified reed-solomon codes (corresp.). *IEEE Transactions on Information Theory*, 21(5):575–576, 1975.

[8] David R Finston and Patrick J Morandi. *Abstract Algebra: Structure and Application*. Springer, 2014.

[9] G David Forney. Concatenated codes. 1965.

[10] Harold Fredricksen. Error correction for deep space network teletype circuits. Technical report, 1968.

[11] Shuhong Gao. A new algorithm for decoding reed-solomon codes. *Communications, information and network security*, pages 55–68, 2003.

[12] W.A. Geisel, United States. National Aeronautics, Space Administration, and Lyndon B. Johnson Space Center. *Tutorial on Reed-Solomon Error Correction Coding*. NASA technical memorandum. National Aeronautics and Space Administration, Lyndon B. Johnson Space Center, 1990.

[13] Daniel Gorenstein and Neal Zierler. A class of error-correcting codes in pˆm symbols. *Journal of the Society for Industrial and Applied Mathematics*, 9(2):207–214, 1961.

[14] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on information theory*, 54(1):135–150, 2008.

[15] Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffers*, 2:147–156, 1959.

[16] Avi Kak. Lecture notes on computer and network security. *Purdue University*, 2015.

[17] Allen Klinger. The vandermonde matrix. *The American Mathematical Monthly*, 74(5):571–574, 1967.

[18] Ralf Kotter. A fast parallel implementation of a berlekamp-massey algorithm for algebraic-geometric codes. *IEEE Transactions on Information Theory*, 44(4):1353–1368, 1998.

[19] Marcus Stenfert Kroese. Open-sourcing cs education: Computer science field guide 2.0. 2015.

[20] James Massey. Shift-register synthesis and bch decoding. *IEEE transactions on Information Theory*, 15(1):122–127, 1969.

[21] James S Milne. Fields and galois theory. *Courses Notes, Version*, 4, 2003.

[22] Todd K Moon. *Error correction coding: mathematical methods and algorithms.* John Wiley & Sons, 2020.

[23] Wesley Peterson. Encoding and error-correction procedures for the bose-chaudhuri codes. *IRE Transactions on information theory*, 6(4):459–470, 1960.

[24] Quora. What is pits and lands in cd?, 2020. [Online; accessed April 10, 2023].

[25] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[26] Shojiro Sakata. Extension of the berlekamp-massey algorithm to n dimensions. *Information and Computation*, 84(2):207–239, 1990.

[27] Jonathan Scarlett. Lecture notes on introduction to information theory. *National University of Singapore*, 2023.

[28] R. Singleton. Maximum distanceq-nary codes. *IEEE Transactions on Information Theory*, 10(2):116–118, 1964.

[29] Endre Süli and David F Mayers. *An introduction to numerical analysis.* Cambridge university press, 2003.

[30] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[31] L Welch and ER Berlekamp. Error correction for algebraic block codes. In *IEEE Int. Symp. Inform. Theory*, 1983.

[32] S Whitaker, K Cameron, P Owsley, and G Maki. Custom cmos reed solomon coder for the hubble space telescope. In *IEEE Conference on Military Communications*, pages 116–120. IEEE, 1990.

[33] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications.* John Wiley & Sons, 1999.

Appendix: Project Administration

Brief Summary of Contributions

Overall, the two authors have contributed equally to the report.

- The **Abstract**, **Introduction**, **Applications** and **Conclusion** sections were written together by the two authors.
- For the **Mechanism** section, Fan Jue focused on the encoding and decoding (without Gao decoder) of Reed-Solomon codes ; Tian Xiao focused on Galois field and Gao decoder. The **Limitations** subsection are written together by the two authors. However, both authors proofread and made appropriate updates on the scripts written by each other.
- For the **Improvement on Reed-Solomon Codes** section, Fan Jue focused on FRS codes and Tian Xiao focused on BCH view of Reed-Solomon codes and interleaving. However, both authors proofread and made appropriate and significant updates on the scripts written by each other.

Page Count

The page count, excluding the cover page (abstract), reference list and figures/tables, is **14 pages**, which is within the specified range of report length.

Acknowledgement

The authors would like to thank Assistant Professor Jonathan Scarlett for his guidance on information theory throughout the course *CS3236: Introduction to Information Theory* at National University of Singapore.