Decoding Algorithms for Reed-Solomon Codes

Annie Cervin

Department of Mathematics and Computer Science
College of the Holy Cross

Advisor: John Little
1 Introduction
   - Coding Theory
   - Reed-Solomon Codes
   - Encoding

2 Unique Decoding Algorithms for Reed-Solomon Codes
   - Decoding Reed-Solomon Codes

3 List Decoding Algorithms for Reed-Solomon Codes
   - Interpolation and Factorization

4 The Lee O’Sullivan Algorithm and the FGLM Algorithm
   - The Lee O’Sullivan Algorithm
   - The FGLM Algorithm
   - Theoretical Comparison
   - Experimental Comparison
Introduced by Claude Shannon in 1948, coding theory tries to eliminate errors in transmitted messages.
Coding theory involves the study of both encoding and decoding.

- The encoding algorithm incorporates redundancy into a message.
Coding theory involves the study of both encoding and decoding.

- The encoding algorithm incorporates redundancy into a message.

- The message is transmitted.
Coding theory involves the study of both encoding and decoding.

- The encoding algorithm incorporates redundancy into a message.
- The message is transmitted.
- The decoding algorithm analyzes the received word and uses the redundancy to find the possibilities for the original message.
Block Codes

Uncoded messages are divided into words with fixed length $k$. 
Block Codes

- Uncoded messages are divided into words with fixed length $k$.
- The words are made from an alphabet of $q$ symbols.
Block Codes

- Uncoded messages are divided into words with fixed length $k$.
- The words are made from an alphabet of $q$ symbols.
- Each alphabet of $q$ symbols corresponds to a finite field $\mathbb{F}_q$. 
Block Codes

- Uncoded messages are divided into words with fixed length $k$.
- The words are made from an alphabet of $q$ symbols.
- Each alphabet of $q$ symbols corresponds to a finite field $\mathbb{F}_q$.
- The possible words in a message can be thought of as $k$-tuples of elements of $\mathbb{F}_q$. The collection of words is identified as $\mathbb{F}_q^k$. 
Block Codes

- Uncoded messages are divided into words with fixed length $k$.
- The words are made from an alphabet of $q$ symbols.
- Each alphabet of $q$ symbols corresponds to a finite field $\mathbb{F}_q$.
- The possible words in a message can be thought of as $k$-tuples of elements of $\mathbb{F}_q$. The collection of words is identified as $\mathbb{F}_q^k$.
- Pick an integer $n > k$. 

Introduction
Unique Decoding Algorithms for Reed-Solomon Codes
List Decoding Algorithms for Reed-Solomon Codes
The Lee O'Sullivan Algorithm and the FGLM Algorithm
Summary

Coding Theory
Reed-Solomon Codes
Encoding

Uncoded messages are divided into words with fixed length $k$.
The words are made from an alphabet of $q$ symbols.
Each alphabet of $q$ symbols corresponds to a finite field $\mathbb{F}_q$.
The possible words in a message can be thought of as $k$-tuples of elements of $\mathbb{F}_q$. The collection of words is identified as $\mathbb{F}_q^k$.
Pick an integer $n > k$. 

Annie Cervin
Decoding Algorithms for Reed-Solomon Codes
Block Codes

- Uncoded messages are divided into words with fixed length $k$.
- The words are made from an alphabet of $q$ symbols.
- Each alphabet of $q$ symbols corresponds to a finite field $\mathbb{F}_q$.
- The possible words in a message can be thought of as $k$-tuples of elements of $\mathbb{F}_q$. The collection of words is identified as $\mathbb{F}_q^k$.
- Pick an integer $n > k$.
- Each message will consist of blocks of $n$-tuples.
Encoding and Decoding Operations

- The encoding operation can be described as $E : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$. 
Encoding and Decoding Operations

- The encoding operation can be described as $E : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$.
- The decoding operation is $D : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^k \cup \{\text{fail}\}$.
Encoding and Decoding Operations

- The encoding operation can be described as $E : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$.
- The decoding operation is $D : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^k \cup \{\text{fail}\}$.
- The set of all codewords is $C = \text{Im}(E)$. 

Annie Cervin
Decoding Algorithms for Reed-Solomon Codes
Encoding and Decoding Operations

- The encoding operation can be described as $E : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$.
- The decoding operation is $D : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^k \cup \{\text{fail}\}$.
- The set of all codewords is $C = \text{Im}(E)$.
- When a codeword $x$ is transmitted with an error, $x$ is replaced by $v = x + e$. 

Encoding and Decoding Operations

- The encoding operation can be described as $E : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$.
- The decoding operation is $D : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^k \cup \{\text{fail}\}$.
- The set of all codewords is $C = \text{Im}(E)$.
- When a codeword $x$ is transmitted with an error, $x$ is replaced by $v = x + e$.
- The error vector is $e \in \mathbb{F}_q^n$ and $v$ is the received word.
Hamming Distance

**Definition**

*The Hamming weight of a word $u$, written as $wt(u)$, is the number of non-zero entries in $u$. The Hamming distance between two words $u$ and $v$, written as $d(u, v)$, is the number of entries in which they differ.*

- Working over the finite field $\mathbb{F}_7$, let $u=(3, 1, 3, 5, 0, 6, 5)$ and $v = (3, 1, 2, 4, 0, 2, 0)$. 
Hamming Distance

Definition

The Hamming weight of a word $u$, written as $\text{wt}(u)$, is the number of non-zero entries in $u$. The Hamming distance between two words $u$ and $v$, written as $d(u, v)$, is the number of entries in which they differ.

- Working over the finite field $\mathbb{F}_7$, let $u = (3, 1, 3, 5, 0, 6, 5)$ and $v = (3, 1, 2, 4, 0, 2, 0)$.
- $u - v = (0, 0, 1, 1, 0, 4, 5)$. 
Hamming Distance

Definition

The Hamming weight of a word $u$, written as $\text{wt}(u)$, is the number of non-zero entries in $u$. The Hamming distance between two words $u$ and $v$, written as $d(u, v)$, is the number of entries in which they differ.

- Working over the finite field $\mathbb{F}_7$, let $u=(3, 1, 3, 5, 0, 6, 5)$ and $v = (3, 1, 2, 4, 0, 2, 0)$.
- $u - v = (0, 0, 1, 1, 0, 4, 5)$.
- Then $\text{wt}(u - v)$ is 4 and the Hamming distance between $u$ and $v$ is 4.
Definition

The minimum distance $d$ of a code $C$ is the smallest Hamming distance between distinct codewords of $C$.

Let $C$ have the following codewords:

$(0, 0, 0), (1, 1, 0), (0, 1, 1), (1, 0, 1)$. 
**Definition**

*The minimum distance* $d$ *of a code* $C$ *is the smallest Hamming distance between distinct codewords of* $C$.

- Let $C$ have the following codewords:
  - $(0, 0, 0)$, $(1, 1, 0)$, $(0, 1, 1)$, $(1, 0, 1)$.
- Each differs from the other in at least two places.
**Definition**

*The minimum distance* $d$ *of a code* $C$ *is the smallest Hamming distance between distinct codewords of* $C$.

- Let $C$ have the following codewords: $(0, 0, 0), (1, 1, 0), (0, 1, 1), (1, 0, 1)$.
- Each differs from the other in at least two places.
- Thus, the minimum distance of $C$ is 2.
Detection and Correction of Errors

**Theorem**

Let $C$ be a code. Then errors of weight $\leq \delta$ in the received words can be detected if and only if the minimum distance $d \geq \delta + 1$.

**Theorem**

Errors of weight $\leq \delta$ can be corrected by nearest neighbor decoding if $d \geq 2\delta + 1$. 
Introduction

- Presented in a paper by Irving Reed and Gustave Solomon in 1960.
Introduction

- Presented in a paper by Irving Reed and Gustave Solomon in 1960.
- They are used in Compact Disc players and space communication.
Introduction

- Presented in a paper by Irving Reed and Gustave Solomon in 1960.
- They are used in Compact Disc players and space communication.
- Based on finite fields or Galois Fields.
Introduction

- Presented in a paper by Irving Reed and Gustave Solomon in 1960.
- They are used in Compact Disc players and space communication.
- Based on finite fields or Galois Fields.
- Reed-Solomon codes are linear codes.

Definition

A linear code of length $n$ over the field $\mathbb{F}_q$ is a vector subspace of $\mathbb{F}_q^n$. 

Annie Cervin

Decoding Algorithms for Reed-Solomon Codes
The codes achieve the Singleton bound over a fixed finite field.

Theorem

*The Singleton bound requires that for any code* \( C \subset \mathbb{F}_q^n \) *with* \( q^k \) *codewords and minimum distance* \( d \),

\[
k \leq n - d + 1.
\]
Creating Reed-Solomon Codes

- We fix \( n = q - 1 \), an integer \( k \leq q \), and all polynomials with degree \( \leq k - 1 \) over \( \mathbb{F}_q \).
Creating Reed-Solomon Codes

- We fix $n = q - 1$, an integer $k \leq q$, and all polynomials with degree $\leq k - 1$ over $\mathbb{F}_q$.
- Each codeword is made by evaluating one of these polynomials with coefficients in $\mathbb{F}_q$ at the nonzero elements of $\mathbb{F}_q$. 
Creating Reed-Solomon Codes

- We fix $n = q - 1$, an integer $k \leq q$, and all polynomials with degree $\leq k - 1$ over $\mathbb{F}_q$.
- Each codeword is made by evaluating one of these polynomials with coefficients in $\mathbb{F}_q$ at the nonzero elements of $\mathbb{F}_q$.
- The nonzero elements can be written in terms of a primitive element for $\mathbb{F}_q$, $\alpha$, and are $1, \alpha, \ldots, \alpha^{q-2}$.
Creating Reed-Solomon Codes

- We fix \( n = q - 1 \), an integer \( k \leq q \), and all polynomials with degree \( \leq k - 1 \) over \( \mathbb{F}_q \).
- Each codeword is made by evaluating one of these polynomials with coefficients in \( \mathbb{F}_q \) at the nonzero elements of \( \mathbb{F}_q \).
- The nonzero elements can be written in terms of a primitive element for \( \mathbb{F}_q \), \( \alpha \), and are \( 1, \alpha, \ldots, \alpha^{q-2} \).
- Let \( L_k \) be the \( \mathbb{F}_q \) vector space of polynomials of degree \( < k \) with coefficients in \( \mathbb{F}_q \).
Creating Reed-Solomon Codes

- We fix \( n = q - 1 \), an integer \( k \leq q \), and all polynomials with degree \( \leq k - 1 \) over \( \mathbb{F}_q \).
- Each codeword is made by evaluating one of these polynomials with coefficients in \( \mathbb{F}_q \) at the nonzero elements of \( \mathbb{F}_q \).
- The nonzero elements can be written in terms of a primitive element for \( \mathbb{F}_q \), \( \alpha \), and are \( 1, \alpha, \ldots, \alpha^{q-2} \).
- Let \( L_k \) be the \( \mathbb{F}_q \) vector space of polynomials of degree \( < k \) with coefficients in \( \mathbb{F}_q \).
- The linear evaluation mapping can be written as:

\[
\omega : L_k \rightarrow \mathbb{F}_q^{q-1}
\]
\[
f \mapsto (f(1), f(\alpha), \ldots, f(\alpha^{q-2})).
\]
Creating Reed-Solomon Codes

- We fix $n = q - 1$, an integer $k \leq q$, and all polynomials with degree $\leq k - 1$ over $\mathbb{F}_q$.
- Each codeword is made by evaluating one of these polynomials with coefficients in $\mathbb{F}_q$ at the nonzero elements of $\mathbb{F}_q$.
- The nonzero elements can be written in terms of a primitive element for $\mathbb{F}_q$, $\alpha$, and are $1, \alpha, \ldots, \alpha^{q-2}$.
- Let $L_k$ be the $\mathbb{F}_q$ vector space of polynomials of degree $< k$ with coefficients in $\mathbb{F}_q$.
- The linear evaluation mapping can be written as:
  \[
  \omega : L_k \rightarrow \mathbb{F}_q^{q-1}
  \]
  \[
  f \mapsto (f(1), f(\alpha), \ldots, f(\alpha^{q-2})).
  \]
- $\text{Im}(L_k)$ is denoted $RS(k, q)$. 
Minimum Distance

- The minimum distance of $RS(k, q)$ is $d = q - k$. 
Minimum Distance

- The minimum distance of $RS(k, q)$ is $d = q - k$.
- The Singleton bound is achieved for $RS(k, q)$ because
  \[ k = n - d + 1 = (q - 1) - (q - k) + 1 = k. \]
Minimum Distance

- The minimum distance of $RS(k, q)$ is $d = q - k$.
- The Singleton bound is achieved for $RS(k, q)$ because $k = n - d + 1 = (q - 1) - (q - k) + 1 = k$.
- Every $RS(k, q)$ achieves the largest possible code minimum distance for this specific block length $n$. 

Annie Cervin
Minimum Distance

- The minimum distance of $RS(k, q)$ is $d = q - k$.
- The Singleton bound is achieved for $RS(k, q)$ because
  \[ k = n - d + 1 = (q - 1) - (q - k) + 1 = k. \]
- Every $RS(k, q)$ achieves the largest possible code minimum distance for this specific block length $n$.
- $RS(k, q)$ can correct codes up to $\tau$ where $\tau = \left\lfloor (n - k)/2 \right\rfloor$. 
The codewords themselves can then be used to produce polynomials such as
\[(c_1, c_2, \ldots, c_n) \rightarrow c_1 + c_2 t + c_3 t^2 + \ldots + c_n t^{n-1}.

**Theorem**

*The Reed-Solomon code* $RS(k, q)$ *is a cyclic code over* $\mathbb{F}_q$. *It is generated by* $g(t) = (t - \alpha)(t - \alpha^2) \ldots (t - \alpha^{2\tau})$. *Its minimum distance is* $d = q - k = 2\tau + 1$. 
Division Algorithm
The most common method uses division to achieve encoding.

- Take $c = (c_1, \ldots, c_k)$ and create
  \[ m(t) = c_k t^{q-2} + \ldots + c_1 t^{q-k-1}. \]
Division Algorithm

The most common method uses division to achieve encoding.

- Take \( c = (c_1, \ldots, c_k) \) and create
  \[
  m(t) = c_k t^{q-2} + \ldots + c_1 t^{q-k-1}.
  \]

- Divide \( g(t) = (t - \alpha)(t - \alpha^2) \ldots (t - \alpha^{q-k-1}) \) into \( m(t) \)
  using the division algorithm. Thus, \( m(t) = q(t) \cdot g(t) + r(t) \).
Division Algorithm

The most common method uses division to achieve encoding.

- Take \( c = (c_1, \ldots, c_k) \) and create
  \[
  m(t) = c_k t^{q-2} + \ldots + c_1 t^{q-k-1}.
  \]

- Divide \( g(t) = (t - \alpha)(t - \alpha^2)\ldots (t - \alpha^{q-k-1}) \) into \( m(t) \) using the division algorithm. Thus, \( m(t) = q(t) \cdot g(t) + r(t) \).

- Form \( f(t) = q(t) \cdot g(t) = m(t) - r(t) \).
Division Algorithm

The most common method uses division to achieve encoding.

- Take \( c = (c_1, \ldots, c_k) \) and create
  \[
  m(t) = c_k t^{q-2} + \ldots + c_1 t^{q-k-1}.
  \]
- Divide \( g(t) = (t - \alpha)(t - \alpha^2) \ldots (t - \alpha^{q-k-1}) \) into \( m(t) \) using the division algorithm. Thus, \( m(t) = q(t) \cdot g(t) + r(t) \).
- Form \( f(t) = q(t) \cdot g(t) = m(t) - r(t) \).
- \( f(t) \) is a codeword because it is a multiple of the generator polynomial \( g(t) \). Transmit \( f(t) \).
Introduction

Unique decoding algorithms are constructed to return only one codeword from the received word.
Introduction

- Unique decoding algorithms are constructed to return only one codeword from the received word.

- Let a code $C$ have minimum distance $d \geq 2\delta + 1$ and the weight of the error introduced by the channel be $\text{wt}(e) \leq \delta$. 
Introduction

- Unique decoding algorithms are constructed to return only one codeword from the received word.
- Let a code $C$ have minimum distance $d \geq 2\delta + 1$ and the weight of the error introduced by the channel be $\text{wt}(e) \leq \delta$.
- If an error occurs, the nearest neighbor will be the original word.
Introduction

- Unique decoding algorithms are constructed to return only one codeword from the received word.
- Let a code $C$ have minimum distance $d \geq 2\delta + 1$ and the weight of the error introduced by the channel be $\text{wt}(e) \leq \delta$.
- If an error occurs, the nearest neighbor will be the original word.
- If, however, the error has $\text{wt}(e) > \delta$, a fail message will be returned.
Introduction

- Unique decoding algorithms are constructed to return only one codeword from the received word.
- Let a code $C$ have minimum distance $d \geq 2\delta + 1$ and the weight of the error introduced by the channel be $wt(e) \leq \delta$.
- If an error occurs, the nearest neighbor will be the original word.
- If, however, the error has $wt(e) > \delta$, a fail message will be returned.
- There exists a unique decoding algorithm based on the Extended Euclidean Algorithm for the greatest common divisor and the combination of polynomials that gives you the greatest common divisor.
The Basics

- Introduced by Peter Elias in the 1950s.
The Basics

Introduced by Peter Elias in the 1950s.

Accept lists of size \( \leq L \) with a decoding radius \( T \), the decoder will return at most \( L \) codewords which are at most a distance \( T \) from the received word.
The Basics

- Introduced by Peter Elias in the 1950s.
- Accept lists of size \( \leq L \) with a decoding radius \( T \), the decoder will return at most \( L \) codewords which are at most a distance \( T \) from the received word.
- We focus on Sudan-Guruswami’s work from the late 1990s.
Polynomials in Two Variables

- The ring of polynomials in $x, y$ with coefficients in a field $K$ is denoted as $K[x, y]$. 
Polynomials in Two Variables

- The ring of polynomials in $x, y$ with coefficients in a field $K$ is denoted as $K[x, y]$.
- $I = \langle x, y \rangle$ is a nonprincipal ideal in $K[x, y]$. 
Polynomials in Two Variables

- The ring of polynomials in $x, y$ with coefficients in a field $K$ is denoted as $K[x, y]$.
- $I = \langle x, y \rangle$ is a nonprincipal ideal in $K[x, y]$.
- A monomial order $>$ in $K[x, y]$ is a relation on the set of monomials $\{x^a y^b | a, b \geq 0\} = \{x^\alpha | \alpha = (a, b)\}$. 
Polynomials in Two Variables

- The ring of polynomials in $x, y$ with coefficients in a field $K$ is denoted as $K[x, y]$.
- $I = \langle x, y \rangle$ is a nonprincipal ideal in $K[x, y]$.
- A monomial order $\succ$ in $K[x, y]$ is a relation on the set of monomials $\{x^a y^b | a, b \geq 0\} = \{x^\alpha | \alpha = (a, b)\}$.
- A specific type of monomial ordering is the lexicographic order. For $x \succ y$, $x^a y^b \succ_{\text{lex}} x^c y^d$ if $a > c$, or $a = c$ and $b > d$. 
Polynomials in Two Variables

- The ring of polynomials in $x, y$ with coefficients in a field $K$ is denoted as $K[x, y]$.
- $I = \langle x, y \rangle$ is a nonprincipal ideal in $K[x, y]$.
- A monomial order $>$ in $K[x, y]$ is a relation on the set of monomials \( \{x^ay^b | a, b \geq 0\} = \{x^\alpha | \alpha = (a, b)\} \).
- A specific type of monomial ordering is the lexicographic order. For $x > y$, $x^ay^b >_{\text{lex}} x^cy^d$ if $a > c$, or $a = c$ and $b > d$.
- For example, in the weight order $>(1,3), \text{lex}$, if $a + 3b > c + 3d$ or $a + 3b = c + 3d$ then $x^ay^b >_{\text{lex}} x^cy^d$. 

\( \text{Summary} \)
Polynomials in Two Variables

- The ring of polynomials in $x, y$ with coefficients in a field $K$ is denoted as $K[x, y]$.
- $I = \langle x, y \rangle$ is a nonprincipal ideal in $K[x, y]$.
- A monomial order $>$ in $K[x, y]$ is a relation on the set of monomials $\{x^a y^b | a, b \geq 0\} = \{x^\alpha | \alpha = (a, b)\}$.
- A specific type of monomial ordering is the lexicographic order. For $x > y$, $x^a y^b >_{\text{lex}} x^c y^d$ if $a > c$, or $a = c$ and $b > d$.
- For example, in the weight order $>(1,3), \text{lex}$, if $a + 3b > c + 3d$ or $a + 3b = c + 3d$ then $x^a y^b >_{\text{lex}} x^c y^d$.
- The following gives the monomials listed in increasing $(1, 3), \text{lex}$ order:

$1 < x < x^2 < y < x^3 < xy < x^4 < x^2y < x^5 < y^2 < x^3y < x^6 < \ldots$
Leading Term

**Definition**

*The leading term of a polynomial $f$ with respect to a monomial order is the term of highest weighted degree in $f$. It is denoted as $\text{LT}_>(f)$.***
Weighted Degree

Given $v \geq 1$, the $(1, v)$-degree of $x^a y^b$ is $a \cdot 1 + b \cdot v = a + bv$. 
Weighted Degree

- Given $v \geq 1$, the $(1, v)$-degree of $x^a y^b$ is $a \cdot 1 + b \cdot v = a + bv$.
- $C(v, l)$ is the number of monomials $x^a y^b$ with $(1, v)$-degree $\leq l$.

**Proposition**

$$C(v, l) = \left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right) \left(l + 1 - \left\lfloor \frac{l}{v} \right\rfloor \cdot \frac{v}{2}\right)$$
Example

- Let’s look at the $x^a y^b$ that have (1, 4)-degree $\leq 6$. 
Example

- Let’s look at the $x^a y^b$ that have $(1, 4)$-degree $\leq 6$.

\[
C(v, l) = C(4, 6) = \left(\left\lfloor \frac{6}{4} \right\rfloor + 1\right) \left(6 + 1 - \left\lfloor \frac{6}{4} \right\rfloor \cdot \frac{4}{2}\right)
\]
\[
= (1 + 1)(6 + 1 - 1 \cdot 2) = 10.
\]
Example

- Let’s look at the $x^a y^b$ that have $(1, 4)$-degree $\leq 6$.

$$C(v, l) = C(4, 6) = \left(\left\lceil \frac{6}{4} \right\rceil + 1 \right) \left( 6 + 1 - \left\lceil \frac{6}{4} \right\rceil \cdot \frac{4}{2} \right)$$

$$= (1 + 1)(6 + 1 - 1 \cdot 2) = 10.$$ 

Thus there are 10 monomials $x^a y^b$ with $(1, 4)$ degree $\leq 6$. 

Example

- Let’s look at the $x^a y^b$ that have $(1, 4)$-degree $\leq 6$.

\[
C(v, l) = C(4, 6) = \left( \left\lfloor \frac{6}{4} \right\rfloor + 1 \right) \left( 6 + 1 - \left\lfloor \frac{6}{4} \right\rfloor \cdot \frac{4}{2} \right) \\
= (1 + 1)(6 + 1 - 1 \cdot 2) = 10.
\]

- Thus there are 10 monomials $x^a y^b$ with $(1, 4)$ degree $\leq 6$.
- These monomials are $1, x, x^2, x^3, x^4, x^5, x^6, y, xy$, and $x^2 y$. 
Division for Polynomials in Two Variables

The division algorithm for polynomials in two variables works according to a monomial order. Given polynomials $f, f_1, \ldots, f_s \in K[x, y]$, using the division algorithm we can find

$$f = a_1 f_1 + \ldots + a_s f_s + r$$

where $\text{LT}(a_i f_i) \leq \text{LT}(f)$ for all $i$ and $a_i, r \in K[x, y]$. Either the polynomial $r = 0$ or no term in $r$ is divisible by any $\text{LT}(f_i)$. 

Annie Cervin
Gröbner Basis

**Definition**

If $I$ is an ideal in $K[x, y]$ and $>$ is a monomial order then a subset $G \subseteq I$ is a Gröbner basis for $I$ with respect to $>$ if

$$\langle LT_>(g) \mid g \in G \rangle = \langle LT_>(f) \mid f \in I \rangle.$$  

**Theorem**

Given an ideal $I$ and a monomial order $>,$ there is a unique reduced Gröbner basis for $I$ with respect to $>.$
List decoding algorithms have two steps: interpolation and factorization.
List decoding algorithms have two steps: interpolation and factorization.

Interpolation finds a minimal polynomial

\[ Q(x, y) = a_L(x)y^L + a_{L-1}(x)y^{L-1} + \ldots + a_0(x) \]

such that

\[ Q(\alpha^i, y_i) = 0 \text{ for all } i = 0, \ldots, q - 2. \]
List decoding algorithms have two steps: interpolation and factorization.

Interpolation finds a minimal polynomial
\[ Q(x, y) = a_L(x)y^L + a_{L-1}(x)y^{L-1} + \ldots + a_0(x) \]
such that
\[ Q(\alpha^i, y_i) = 0 \]
for all \( i = 0, \ldots, q - 2 \).

For every Reed-Solomon codeword within distance \( T \) of \( y \), factorization gives some \( y - f_i(x) \) with \( \deg(f_i) \leq k - 1 \) that divides \( Q(x, y) \). In other words, factoring gives
\[ Q(x, y) = (y - f_1(x))(y - f_2(x)) \cdots (y - f_L(x)). \]
List decoding algorithms have two steps: interpolation and factorization.

Interpolation finds a minimal polynomial
\[ Q(x, y) = a_L(x)y^L + a_{L-1}(x)y^{L-1} + \ldots + a_0(x) \]
such that
\[ Q(\alpha^i, y_i) = 0 \text{ for all } i = 0, \ldots, q - 2. \]

For every Reed-Solomon codeword within distance \( T \) of \( y \), factorization gives some \( y - f_i(x) \) with \( \deg(f_i) \leq k - 1 \) that divides \( Q(x, y) \). In other words, factoring gives
\[ Q(x, y) = (y - f_1(x))(y - f_2(x)) \cdots (y - f_L(x)). \]

The decoder returns \( ev(f_1), ev(f_2), \ldots, ev(f_L) \).
Definition

\( Q(x, y) \) has a zero of multiplicity at least \( m \) at \((\alpha^i, y_i)\) if

\[
Q(x, y) = \sum_{k, l \geq 0} c_{k, l} (x - \alpha^i)^k (y - y_i)^l
\]

and

\[
c_{0, 0} = c_{1, 0} = c_{0, 1} = \ldots = c_{k, l} = 0
\]

for all \( k, l \leq m - 1 \). \( Q(x, y) \) has a zero of multiplicity exactly \( m \) if \( c_{k, l} \neq 0 \) for some \( k, l \) with \( k + l = m \).
Theorem

Let $\phi_j(x, y)$ denote monomials of the form $x^a y^b$ listed in increasing order according to an arbitrary monomial order and

$$Q(x, y) = \sum_{j=0}^{C} a_j \phi_j(x, y).$$

Then a nonzero $Q(x, y)$ polynomial exists that interpolates the points $(\alpha^i, y_i)$ for $i = 1, 2, \ldots, n$ with multiplicity $m$ at each point if

$$C = n\binom{m + 1}{2}.$$
Theorem

Let $K_m = \min\{K : C(k - 1, mK - 1)\} > \binom{m+1}{2} n$. Then if the following are satisfied:

\[
\begin{cases}
C(k - 1, l) > \binom{m+1}{2} n \\
mK_m > l \\
p(x) \text{ has degree } \leq k - 1 \\
y_i = p(\alpha^i) \text{ for at least } K_m \text{ different } i,
\end{cases}
\]

$Q(x, y)$ is divisible by $y - p(x)$. 

Annie Cervin
Decoding Algorithms for Reed-Solomon Codes
Introduction

- Kwankyu Lee and Michael O’Sullivan introduced a new way to solve the interpolation step.
Introduction

- Kwankyu Lee and Michael O’Sullivan introduced a new way to solve the interpolation step.
- It finds the minimal polynomial of an ideal using Gröbner bases of modules.
Introduction

- Kwankyu Lee and Michael O’Sullivan introduced a new way to solve the interpolation step.
- It finds the minimal polynomial of an ideal using Gröbner bases of modules.
- It starts with a set of generators of the module induced from the ideal for the points \( \{ P_1, P_2, \ldots, P_n \} \) where \( P_i = (\alpha^i, y_i) \).
Introduction

- Kwankyu Lee and Michael O’Sullivan introduced a new way to solve the interpolation step.
- It finds the minimal polynomial of an ideal using Gröbner bases of modules.
- It starts with a set of generators of the module induced from the ideal for the points \( \{ P_1, P_2, \ldots, P_n \} \) where \( P_i = (\alpha^i, y_i) \).
- It then translates the generators to a Gröbner basis of the module.
Definition

$l_{v,m} \text{ is an ideal of all polynomials } p(x, y) \text{ in } \mathbb{F}_q[x, y] \text{ such that } p(x, y) \text{ vanishes to multiplicity } m \text{ at all } (\alpha_i, v_i)$.

Definition

$\mathbb{F}_q[x, y]_l \text{ is a free module over } \mathbb{F}_q[x] \text{ with basis } \{1, y, y^2, \ldots, y^l\}$. It can be written as

$$\mathbb{F}_q[x, y]_l = \{p(x, y) \mid \deg_y(p(x, y)) \leq l\}.$$  

Monomials in this module are $x^i y^j$ with $i \geq 0$ and $0 \leq j \leq l$.

Definition

$l_{v, m, l} = l_{v, m} \cap \mathbb{F}_q[x, y]_l$. 
The Algorithm

- We will use a set of generators of $I_v, m, l$. 


The Algorithm

- We will use a set of generators of \( l_v, m, l \).
- It has input \( m, l \), and \( v = (v_1, v_2, \ldots, v_n) \) and monomial order \( >_{k-1} \).
The Algorithm

- We will use a set of generators of $I_v, m, l$.
- It has input $m, l$, and $v = (v_1, v_2, \ldots, v_n)$ and monomial order $>_k-1$.
- We will let $g_i = \sum_{j=0}^l a_{ij} y^j$ for $0 \leq i \leq l$. 

Annie Cervin  
Decoding Algorithms for Reed-Solomon Codes
This algorithm is creating a Gröbner basis \( \{g_0, g_1, \ldots, g_l\} \) of \( S \) such that \( \deg_y(LT(g_i)) = i \) for \( 0 \leq i \leq l \). This algorithm begins with:

\[
g_0 = a_{00} \\
g_1 = a_{10} + a_{11}y \\
g_2 = a_{20} + a_{21}y + a_{22}y^2 \\
\vdots \\
g_l = a_{l0} + a_{l1}y + a_{l2}y^2 + \ldots + a_{11}y^l.
\]

The algorithm goes through the steps such that each time \( g_s \) and \( g_r \) are updated, \( \{g_0, g_1, \ldots, g_l\} \) still generates \( S \). It terminates when we have \( \deg_y(LT(g_i)) = i \) for \( 0 \leq i \leq l \).
The FGLM Algorithm

- It takes an input of a Gröbner basis for a zero-dimensional ideal \( I \) and outputs another Gröbner basis for \( I \) for some other monomial order.
The FGLM Algorithm

- It takes an input of a Gröbner basis for a zero-dimensional ideal \( I \) and outputs another Gröbner basis for \( I \) for some other monomial order.

- We use the lex order as the new monomial order.
The FGLM Algorithm

- It takes an input of a Gröbner basis for a zero-dimensional ideal \( I \) and outputs another Gröbner basis for \( I \) for some other monomial order.
- We use the lex order as the new monomial order.
- \( \mathbb{F} \) is a field and \( R = \mathbb{F}[x_1, \ldots, x_n] \) is the ring of polynomials with \( n \) variables and coefficients in \( \mathbb{F} \).
The FGLM Algorithm

- It takes an input of a Gröbner basis for a zero-dimensional ideal $I$ and outputs another Gröbner basis for $I$ for some other monomial order.
- We use the lex order as the new monomial order.
- $\mathbb{F}$ is a field and $R = \mathbb{F}[x_1, \ldots, x_n]$ is the ring of polynomials with $n$ variables and coefficients in $\mathbb{F}$.
- A zero-dimensional ideal $I$ is one such that

$$\dim_{\mathbb{F}} \mathbb{F}[x_1, \ldots, x_n]/I < \infty.$$
Remainder Arithmetic

- Dividing \( f \in R \) by \( G \) results in:

\[
    f = h_1 g_1 + \ldots + h_t g_t + \bar{f}^G.
\]
Remainder Arithmetic

- Dividing \( f \in R \) by \( G \) results in:

\[
f = h_1 g_1 + \ldots + h_t g_t + \tilde{f}^G.
\]

- \( \tilde{f}^G \) is a linear combination of the monomials \( x^\gamma \notin \langle LT(I) \rangle \) which is a basis for \( \mathbb{F}[x_1, \ldots, x_n]/I \).
Remainder Arithmetic

- Dividing \( f \in R \) by \( G \) results in:

\[
f = h_1 g_1 + \ldots + h_t g_t + \bar{f}^G.
\]

- \( \bar{f}^G \) is a linear combination of the monomials \( x^\gamma \notin \langle LT(I) \rangle \) which is a basis for \( \mathbb{F}[x_1, \ldots, x_n]/I \).

- Since \( G \) is a Gröbner basis, \( f \in I \) if and only if \( \bar{f}^G = 0 \).
The Code

- **Input:** The lex order and $G_1$, the Gröbner basis of the original monomial ordering.
The Code

- Input: The lex order and $G_1$, the Gröbner basis of the original monomial ordering.
- Algorithm updates a list $G_2 = \{g_1, \ldots, g_k\}$ where each $g_i$ is an element of the ideal $I$. 
The Code

- Input: The lex order and $G_1$, the Gröbner basis of the original monomial ordering.
- Algorithm updates a list $G_2 = \{g_1, \ldots, g_k\}$ where each $g_i$ is an element of the ideal $I$.
- Algorithm updates $B$ which is a list of monomials that is initially empty.
The Code

- **Input:** The lex order and $G_1$, the Gröbner basis of the original monomial ordering.
- **Algorithm updates a list** $G_2 = \{g_1, \ldots, g_k\}$ where each $g_i$ is an element of the ideal $I$.
- **Algorithm updates** $B$ which is a list of monomials that is initially empty.
- **Algorithm moves through a list of monomials of the form** $x^\gamma$ that increase by lex order to create the new Gröbner basis.
The Code

1. Compute $\overline{x^\gamma}^G$.

2. If $\overline{x^\gamma}^G$ is linearly dependent of the monomials in $B$ then add $g$ to the list of $G_2$ as the last element.

3. If $\overline{x^\gamma}^G$ is linearly independent of the monomials in $B$ then add $x^\gamma$ to $B$ as the last element.

4. End if the leading term of the last added polynomial $g$ is a power of $x_1$ where $x_1$ is the greatest variable in our lex order.

5. Replace $x^\gamma$ by the next monomial in lex order which is not divisible by any of the monomials $LT(g_i)$ for $g_i \in G_2$ and go back to Step 1.
We are applying a module version of the above algorithm to the Gröbner basis \( \{g_0, \ldots, g_l\} \) for \( I_v, m, l \) with Position over Term order and converting it to a \( > (l, k-1) \) order Gröbner basis for \( I_v, m, l \).
Theoretical Comparison

- We can compare both algorithms by calculating the upper bound of how many multiplication operations are needed.
Theoretical Comparison

- We can compare both algorithms by calculating the upper bound of how many multiplication operations are needed.
- If two polynomials of degree $a$ and $b$ are multiplied it requires $(a + 1)(b + 1)$ operations over $\mathbb{F}$. 
Theoretical Comparison

- We can compare both algorithms by calculating the upper bound of how many multiplication operations are needed.
- If two polynomials of degree $a$ and $b$ are multiplied it requires $(a + 1)(b + 1)$ operations over $\mathbb{F}$.
- Big-O notation describes the behavior of a function when the variable tends to infinity.
Theoretical Comparison

- We can compare both algorithms by calculating the upper bound of how many multiplication operations are needed.
- If two polynomials of degree $a$ and $b$ are multiplied it requires $(a + 1)(b + 1)$ operations over $\mathbb{F}$.
- Big-O notation describes the behavior of a function when the variable tends to infinity.
- For example, if a function $f(n) = O(n^2)$, then $f(n) \leq cn^2$ for some constant $c$ and all values of $n > n_0$. 
Results

- The Lee-O’Sullivan algorithm requires

\[ O(n^4 m^5) \]

multiplication operations.
Results

- The Lee-O’Sullivan algorithm requires
  \[ O(n^4 m^5) \]
  multiplication operations.

- The FGLM algorithm has at most
  \[ O(n^3 m^6) \]
  multiplication operations.
Results

- The Lee-O’Sullivan algorithm requires
  \[ \mathcal{O}(n^4 m^5) \]
  multiplication operations.

- The FGLM algorithm has at most
  \[ \mathcal{O}(n^3 m^6) \]
  multiplication operations.

- Then for those codes that have big \( n \) but the same \( m \), we expect that the FGLM algorithm is better. This corresponds to large fields with small \( m \).
Experimental Comparison

- We used the original procedure for the Lee-O'Sullivan algorithm and strived to optimize the FGLM algorithm.
Experimental Comparison

- We used the original procedure for the Lee-O'Sullivan algorithm and strived to optimize the FGLM algorithm.
- An error vector was randomly created and added to a randomly chosen codeword to create the received word.
Experimental Comparison

- We used the original procedure for the Lee-O'Sullivan algorithm and strived to optimize the FGLM algorithm.
- An error vector was randomly created and added to a randomly chosen codeword to create the received word.
- Since Maple times varied, we calculated the average of 10 run times of each algorithm.
Experimental Comparison

- We used the original procedure for the Lee-O'Sullivan algorithm and strived to optimize the FGLM algorithm.
- An error vector was randomly created and added to a randomly chosen codeword to create the received word.
- Since Maple times varied, we calculated the average of 10 run times of each algorithm.
- For fields smaller than $\mathbb{F}_{11}$, the Lee-O’Sullivan algorithm won every time.
### Field of Size 11

<table>
<thead>
<tr>
<th>Run</th>
<th>Weight of error</th>
<th>Codewords</th>
<th>AVG FGLM</th>
<th>AVG L.O.</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3.083</td>
<td>2.370</td>
<td>FGLM</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3.540</td>
<td>3.341</td>
<td>FGLM</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4.170</td>
<td>4.440</td>
<td>L.O.</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3.063</td>
<td>2.415</td>
<td>FGLM</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4.052</td>
<td>3.445</td>
<td>FGLM</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1.931</td>
<td>2.528</td>
<td>L.O.</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>3.479</td>
<td>3.221</td>
<td>FGLM</td>
</tr>
</tbody>
</table>

**Table:** Comparison of Lee-O’Sullivan and FGLM algorithm for field of size $q=11$, multiplicity $m=4$, and lists of size $l=9$. 
Field of Size 17

<table>
<thead>
<tr>
<th>Run</th>
<th>Weight of error</th>
<th>Codewords</th>
<th>AVG FGLM</th>
<th>AVG L.O.</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>6.358</td>
<td>6.524</td>
<td>L.O.</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1</td>
<td>6.515</td>
<td>6.409</td>
<td>FGLM</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>1</td>
<td>6.532</td>
<td>6.122</td>
<td>FGLM</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>2</td>
<td>6.357</td>
<td>5.802</td>
<td>FGLM</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>3</td>
<td>6.552</td>
<td>6.133</td>
<td>FGLM</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>2</td>
<td>6.714</td>
<td>6.534</td>
<td>FGLM</td>
</tr>
</tbody>
</table>

**Table**: Comparison of Lee-O’Sullivan and FGLM algorithm for field of size $q=17$, multiplicity $m=3$, and lists of size $l=9$. 
Summary

- Decoding algorithms are useful to correct errors.
Summary

- Decoding algorithms are useful to correct errors.
- When the size of the field is greater than $\mathbb{F}_{11}$, we expect that the FGLM algorithm will consistently be faster than the Lee-O’Sullivan algorithm.
Summary

- Decoding algorithms are useful to correct errors.
- When the size of the field is greater than $F_{11}$, we expect that the FGLM algorithm will consistently be faster than the Lee-O’Sullivan algorithm.
- If you are trying to decode received words from a smaller field, the Lee-O’Sullivan algorithm gives superior performance.
For Further Reading


