15-853: Algorithms in the Real World

Error Correcting Codes II
- Cyclic Codes
- Reed-Solomon Codes

Viewing Messages as Polynomials

A (n, k, n-k+1) code:
Consider the polynomial of degree k-1
\[ p(x) = a_{k-1} x^{k-1} + \ldots + a_2 x^2 + a_0 \]

Message: \((a_{k-1}, \ldots, a_1, a_0)\)

Codeword: \((p(1), p(2), \ldots, p(n))\)

To keep the \(p(i)\) fixed size, we use \(a_i \in GF(p^r)\)
To make the \(i\) distinct, \(n < p^r\)

Unisolvence Theorem: Any subset of size \(k\) of \((p(1), p(2), \ldots, p(n))\) is enough to (uniquely) reconstruct \(p(x)\) using polynomial interpolation, e.g., LaGrange's Formula.

Polynomial-Based Code

A \((n, k, 2s +1)\) code:

Can **detect** \(2s\) errors
Can **correct** \(s\) errors
Generally can correct \(\alpha\) erasures and \(\beta\) errors if \(\alpha + 2\beta \leq 2s\)

Correcting Errors

Correcting \(s\) errors:
1. Find \(k + s\) symbols that agree on a polynomial \(p(x)\). These must exist since originally \(k + 2s\) symbols agreed and only \(s\) are in error
2. There are no \(k + s\) symbols that agree on the wrong polynomial \(p'(x)\)
   - Any subset of \(k\) symbols will define \(p'(x)\)
   - Since at most \(s\) out of the \(k+s\) symbols are in error, \(p'(x) = p(x)\)
A Systematic Code

Systematic polynomial-based code
\[ p(x) = a_{k-1}x^{k-1} + \ldots + a_1x + a_0 \]

**Message:** \((a_{k-1}, \ldots, a_1, a_0)\)

**Codeword:** \((a_{k-1}, \ldots, a_1, a_0, p(1), p(2), \ldots, p(2s))\)

This has the advantage that if we know there are no errors, it is trivial to decode.

The version of RS used in practice uses something slightly different than \(p(1), p(2), \ldots\)

This will allow us to use the "Parity Check" ideas from linear codes (i.e., \(Hc^T = 0?\)) to quickly test for errors.

Reed-Solomon Codes in the Real World

(204, 188, 17)\(_{256}\) : ITU J.83(A)^2
(128, 122, 7)\(_{256}\) : ITU J.83(B)
(255, 223, 33)\(_{256}\) : Common in Practice
  - Note that they are all byte based (i.e., symbols are from \(GF(2^8)\)).

Decoding rate on 1.8GHz Pentium 4:
  - (255,251) = 89Mbps
  - (255,223) = 18Mbps

Dozens of companies sell hardware cores that operate 10x faster (or more)
  - (204,188) = 320Mbps (Altera decoder)

Applications of Reed-Solomon Codes

- **Storage:** CDs, DVDs, "hard drives",
- **Wireless:** Cell phones, wireless links
- **Sateline and Space:** TV, Mars rover, ...
- **Digital Television:** DVD, MPEG2 layover
- **High Speed Modems:** ADSL, DSL, ..

Good at handling burst errors.
Other codes are better for random errors.
  - e.g., Gallager codes, Turbo codes

RS and “burst” errors

Let’s compare to Hamming Codes (which are “optimal”).

<table>
<thead>
<tr>
<th>Code</th>
<th>Code Bits</th>
<th>Check Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS (255, 253, 3)(_{256})</td>
<td>2040</td>
<td>16</td>
</tr>
<tr>
<td>Hamming (2^{11}-1, 2^{11}-11-1, 3)(_2)</td>
<td>2047</td>
<td>11</td>
</tr>
</tbody>
</table>

They can both correct 1 error, but not 2 random errors.
  - The Hamming code does this with fewer check bits
  - However, RS can fix 8 contiguous bit errors in one byte
  - Much better than lower bound for 8 arbitrary errors

\[
\log \left(1 + \frac{n}{1} + \ldots + \frac{n}{8}\right) > 8\log(n - 7) \approx 88 \text{ check bits}
\]
**Galois Field**

$GF(2^3)$ with irreducible polynomial: $x^3 + x + 1$

$\alpha = x$ is a generator

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$x$</th>
<th>010</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha^2$</td>
<td>$x^2$</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>$\alpha^3$</td>
<td>$x + 1$</td>
<td>011</td>
<td>4</td>
</tr>
<tr>
<td>$\alpha^4$</td>
<td>$x^2 + x$</td>
<td>110</td>
<td>5</td>
</tr>
<tr>
<td>$\alpha^5$</td>
<td>$x^2 + x + 1$</td>
<td>111</td>
<td>6</td>
</tr>
<tr>
<td>$\alpha^6$</td>
<td>$x^2 + 1$</td>
<td>101</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha^7$</td>
<td>1</td>
<td>001</td>
<td>1</td>
</tr>
</tbody>
</table>

Will use this as an example.

**Discrete Fourier Transform (DFT)**

Another View of polynomial-based codes

$\alpha$ is a primitive $n^{th}$ root of unity ($\alpha^n = 1$) – a generator

$$T = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-1} & \alpha^{2(n-1)} & \cdots & \alpha^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{k-1} \end{pmatrix} = T \cdot \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{k-1} \end{pmatrix}$$

Evaluate polynomial $m_k \cdot x^{k-1} + \cdots + m_1 \cdot x + m_0$ at $n$ distinct roots of unity, $1, \alpha, \alpha^2, \alpha^3, \cdots, \alpha^{n-1}$

Inverse DFT: $m = T^{-1} c$

**DFT Example**

$\alpha = x$ is 7th root of unity in $GF(2^3)/x^3 + x + 1$

(i.e., multiplicative group, which excludes additive inverse)

Recall $\alpha = "2", \alpha^2 = "3", \cdots, \alpha^7 = 1 = "1"$

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ \vdots & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \alpha^{10} & \alpha^{12} \\ 1 & \alpha^4 & \alpha^8 & \alpha^{12} & \alpha^{16} & \alpha^{20} & \alpha^{24} \\ 1 & \alpha^6 & \alpha^{12} & \alpha^{18} & \alpha^{24} & \alpha^{30} & \alpha^{36} \\ \vdots & \alpha^5 & \alpha^{10} & \alpha^{15} & \alpha^{20} & \alpha^{25} & \alpha^{30} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^{15} & \alpha^{18} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \alpha^{10} & \alpha^{12} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 \\ 1 & 3 & 3^2 & 3^3 & 3^4 & 3^5 & 3^6 \\ 1 & 4 & 4^2 & 4^3 & 4^4 & 4^5 & 4^6 \\ 1 & 5 & 5^2 & 5^3 & 5^4 & 5^5 & 5^6 \\ 1 & 6 & 6^2 & 6^3 & 6^4 & 6^5 & 6^6 \\ 1 & 7 & 7^2 & 7^3 & 7^4 & 7^5 & 7^6 \end{pmatrix}$$

Should be clear that $c = T \cdot (m_0, m_1, \ldots, m_{k-1}, 0, \ldots)^T$ is the same as evaluating $p(x) = m_0 + m_1 \cdot x + \cdots + m_{k-1} \cdot x^{k-1}$ at $n$ points.

**Decoding**

Why is it hard?

Brute Force: try $k+2s$ choose $k+s$ possibilities and solve for each.
**Cyclic Codes**

A linear code is cyclic if:
\[(c_0, c_1, ..., c_{n-1}) \in C \Rightarrow (c_{n-1}, c_0, ..., c_{n-2}) \in C\]

Both Hamming and Reed-Solomon codes are cyclic.

**Motivation:** They are more efficient to decode than general codes.

---

**Generator and Parity Check Matrices**

**Generator Matrix:**
A \(k \times n\) matrix \(G\) such that:
\[C = \{m \cdot G \mid m \in \Sigma^k\}\]
Made from stacking the basis vectors

**Parity Check Matrix:**
A \((n-k) \times n\) matrix \(H\) such that:
\[C = \{v \in \Sigma^n \mid H \cdot v^T = 0\}\]
Codewords are the nullspace of \(H\)

These always exist for linear codes
\[H \cdot G^T = 0\]

---

**Generator and Parity Check Polynomials**

**Generator Polynomial:**
A degree \((n-k)\) polynomial \(g\) such that:
\[C = \{m \cdot g \mid m = m_0 + m_1x + ... + m_{k-1}x^{k-1}\}\]

such that \(g \mid x^n - 1\)

**Parity Check Polynomial:**
A degree \(k\) polynomial \(h\) such that:
\[C = \{v \in \Sigma^n [x] \mid h \cdot v = 0 \, (\text{mod } x^n - 1)\}\]

such that \(h \mid x^n - 1\)

These always exist for linear cyclic codes
\[h \cdot g = x^n - 1\]

---

**Viewing \(g\) as a matrix**

If \(g(x) = g_0 + g_1x + ... + g_{n-k-1}x^{n-k-1}\)
We can put this generator in matrix form:
\[
G = \begin{pmatrix}
g_0 & g_1 & \cdots & g_{n-k-1} & 0 & \cdots & 0 \\
0 & g_0 & \cdots & g_{n-k-2} & g_{n-k-1} & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k-1}
\end{pmatrix}
\]

Write \(m = m_0 + m_1x + ... + m_{k-1}x^{k-1}\) as \((m_0, m_1, ..., m_{k-1})\)
Then \(c = mG\)
\[ g \text{ generates cyclic codes} \]
\[
G = \begin{pmatrix}
g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\
0 & g_0 & & g_{n-k-2} & g_{n-k-1} & \cdots & 0 \\
& \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k-1} \\
\end{pmatrix} = \begin{pmatrix} g \\ xg \\ \vdots \\ x^{k-1} \end{pmatrix}
\]

Codes are linear combinations of the rows.
All but last row is clearly cyclic (based on next row)
Shift of last row is \( x^k g \mod (x^n - 1) = g_{n-k-1} 0 \ldots g_1 \ldots g_{n-k-2} \)
Consider \( h = h_0 + h_1 x + \ldots + h_{k-1} x^{k-1} \) \( (gh = x^n - 1) \)
\[
h_0 g + (h_1 x) g + \ldots + (h_{k-1} x^{k-2}) g + (h_{k-1} x^{k-1}) g = x^n - 1 \\
x^k g = -h_{k-1} (h_0 g + h_1 (xg) + \ldots + h_{k-1} (x^{k-1} g)) \mod (x^n - 1)
\]
This is a linear combination of the rows.

**Hamming Codes Revisited**

The Hamming \((7,4,3)\)_2 code.
\[
g = 1 + x + x^3 \quad \quad \quad \quad h = x^4 + x^2 + x + 1
\]
\[
G = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
\end{pmatrix} \quad \quad \quad H = \begin{pmatrix}
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 \\
\end{pmatrix}
\]
\[
gh = x^7 - 1, \quad GH^T = 0
\]
The columns are not identical to the previous example Hamming code.

**Viewing h as a matrix**

If \( h = h_0 + h_1 x + \ldots + h_{k-1} x^{k-1} \)
we can put this parity check poly. in matrix form:
\[
H = \begin{pmatrix}
0 & \cdots & 0 & h_{k-1} & \cdots & h_1 & h_0 \\
0 & \cdots & h_{k-1} & h_{k-2} & \cdots & h_0 & 0 \\
& \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
& & \ddots & h_{k-1} & h_1 & h_0 & 0 & \cdots & 0 \\
\end{pmatrix}
\]
\[
Hc^T = 0
\]

**Factors of \( x^n - 1 \)**

Intentionally left blank
Another way to write $g$

Let $\alpha$ be a **generator** of $GF(p^r)$.
Let $n = p^r - 1$ (the size of the multiplicative group) Then we can write a generator polynomial as

$$g(x) = (x-\alpha)(x-\alpha^2) \ldots (x-\alpha^{n-k})$$
$$h = (x-\alpha^{n-k+1}) \ldots (x-\alpha^n)$$

**Lemma:** $g \mid x^n - 1$, $h \mid x^n - 1$, $gh \mid x^n - 1$

(a | b means a divides b)

**Proof:**
- $\alpha^n = 1$ (because of the size of the group)
  - $\Rightarrow \alpha^{n-1} = 0$
  - $\Rightarrow \alpha$ root of $x^n - 1$
  - $\Rightarrow (x - \alpha) \mid x^n - 1$
- similarly for $\alpha^2, \alpha^3, \ldots, \alpha^n$
- therefore $x^n - 1$ is divisible by $(x - \alpha)(x - \alpha^2) \ldots$

---

Back to Reed-Solomon

Consider a generator polynomial $g \in GF(p^r)[x]$, s.t. $g \mid (x^n - 1)$
Recall that $n - k = 2s$ (the degree of $g$ is $n-k-1$, $n-k$ coefficients)

**Encode:**
- $m' = m x^{2s}$ (basically shift by $2s$)
- $b = m'$ (mod $g$)
- $c = m' - b = (m_{n-1}, ... , m_0, -b_{2s-1}, ... , -b_0)$
- Note that $c$ is a **cyclic code** based on $g$
  - $m' = qg + b$
  - $c = m' - b = qg$

**Parity check:**
- $h c = 0$ ?

---

**Example RS (7,3,5)$_8$**

$n = 7$, $k = 3$, $n-k = 2s = 4$, $d = 2s+1 = 5$

$$g = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)$$
$$= x^4 + \alpha x^3 + \alpha x^2 + \alpha^3 x + \alpha$$

$$h = (x - \alpha^5)(x - \alpha^6)(x - \alpha^7)$$
$$= x^3 + \alpha^3 x^2 + \alpha^2 x + \alpha$$

$$gh = x^7 - 1$$

Consider the message: 110 000 110

$m = (\alpha^4, 0, \alpha^2) = \alpha^4 x^2 + \alpha^4$

$m' = x^4 m = \alpha^4 x^6 + \alpha^4 x^4$

$$= (\alpha^4 x^2 + x + \alpha^3) g + (\alpha^3 x^3 + \alpha^6 x + \alpha^6)$$

$c = (\alpha^4, 0, \alpha^2, 0, \alpha^6, \alpha^6)$

$= 110 000 110 011 000 101 101$

$ch = 0$ (mod $x^7 -1$)
A useful theorem

**Theorem:** For any $\beta$, if $g(\beta) = 0$ then $\beta^2 m(\beta) = b(\beta)$

**Proof:**

$x^2 m(x) = m'(x) = g(x)q(x) + b(x)$

$\beta^2 m(\beta) = g(\beta)q(\beta) + b(\beta) = b(\beta)$

**Corollary:** $\beta^2 m(\beta) = b(\beta)$ for $\beta \in \{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2s+n-k}\}$

**Proof:**

$\{\alpha, \alpha^2, \ldots, \alpha^{2s}\}$ are the roots of $g$ by definition.

Fixing errors

**Theorem:** Any $k$ symbols from $c$ can reconstruct $c$ and hence $m$

**Proof:**

We can write $2s$ equations involving $m (c_{n-1}, \ldots, c_{2s})$ and $b (c_{2s-1}, \ldots, c_0)$. These are

$\alpha^2 s m(\alpha) = b(\alpha)$

$\alpha^{4s} m(\alpha^2) = b(\alpha^2)$

$\ldots$

$\alpha^{2s(2s)} m(\alpha^{2s}) = b(\alpha^{2s})$

We have at most $2s$ unknowns, so we can solve for them. (I'm skipping showing that the equations are linearly independent).

Efficient Decoding

I don't plan to go into the Reed-Solomon decoding algorithm, other than to mention the steps.

Syndrome Calculator → Error Polynomial ← Error Locations

Berlekamp Massy → Chien Search

Error Magnitudes Forney Algorithm → Error Corrector

This is the hard part. CD players use this algorithm. (Can also use Euclid's algorithm.)