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

$p(x) = a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$

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

Codeword: (p(1), p(2), …, p(n))

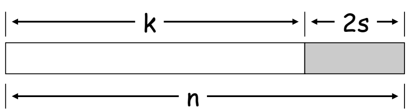
To keep the p(i) fixed size, we use $a_i \in GF(p^r)$

To make the i distinct, $n < p^r$

Unisolvence Theorem: Any subset of size k of (p(1), p(2), …, 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 α erasures and β 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

$p(x) = a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$

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), …

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").

	code bits	check bits
RS $(255, 253, 3)_{256}$	2040	16
Hamming $(2^{11}-1, 2^{11}-11-1, 3)_2$	2047	11

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

$$\log\left(1 + \binom{n}{1} + \cdots + \binom{n}{8}\right) > 8\log(n-7) \approx 88 \text{ check bits}$$

Galois Field

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

$\alpha = x$ is a generator

α	x	010	**2**
α^2	x^2	100	**3**
α^3	$x + 1$	011	**4**
α^4	$x^2 + x$	110	**5**
α^5	$x^2 + x + 1$	111	**6**
α^6	$x^2 + 1$	101	**7**
α^7	1	001	**1**

Will use this as an example.

Discrete Fourier Transform (DFT)

Another View of polynomial-based codes

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

$$T = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-1} & \alpha^{2(n-1)} & \cdots & \alpha^{(n-1)(n-1)} \end{pmatrix} \qquad \begin{pmatrix} c_0 \\ \vdots \\ c_{k-1} \\ c_k \\ \vdots \\ c_{n-1} \end{pmatrix} = T \cdot \begin{pmatrix} m_0 \\ \vdots \\ m_{k-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Evaluate polynomial $m_{k-1}x^{k-1} + \cdots + m_1x + m_0$ 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 7^{th} root of unity in $GF(2^3)/x^3 + x + 1$

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

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

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & & & \\ 1 & \alpha^3 & \alpha^6 & & & & \\ 1 & \alpha^4 & & \ddots & & & \\ 1 & \alpha^5 & & & & & \\ 1 & \alpha^6 & & & & & \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 \\ 1 & 3 & 3^2 & 3^3 & & & \\ 1 & 4 & 4^2 & & & & \\ 1 & 5 & & \ddots & & & \\ 1 & 6 & & & & & \\ 1 & 7 & & & & & 7^6 \end{pmatrix}$$

Should clear that $c = T \bullet (m_0, m_1, \ldots, m_{k-1}, 0, \ldots)^T$ is the same as evaluating $p(x) = m_0 + m_1x + \ldots + m_{k-1}x^{k-1}$ at n points.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$

$$\begin{aligned} X_k &= \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k} \\ &= \sum_{m=0}^{M-1} x_{2m} e^{-\frac{2\pi i}{M} mk} + e^{-\frac{2\pi i}{N}k} \sum_{m=0}^{M-1} x_{2m+1} e^{-\frac{2\pi i}{M} mk} \\ &= \begin{cases} E_k + e^{-\frac{2\pi i}{N}k} O_k & \text{if } k < M \\ E_{k-M} - e^{-\frac{2\pi i}{N}(k-M)} O_{k-M} & \text{if } k \geq M. \end{cases} \end{aligned}$$

```
function fft(a,w,add,mult) =
if #a == 1 then return a
Else
  w' = [w_0,w_2,...,w_{n-1}]
  e = fft([a_0,a_2,...,a_{n-2}],w')
  o = fft([a_1,a_3,...,a_{n-1}],w')
  return [e_0+o_0w_0, e_1+o_1w_1,...,e_{n/2-1}+o_{n/2-1}w_{n/2-1},
          e_0+o_0w_{n/2}, e_1+o_1w_{n/2+1},..., e_{n/2-1}+o_{n/2-1}w_{n-1}]
```

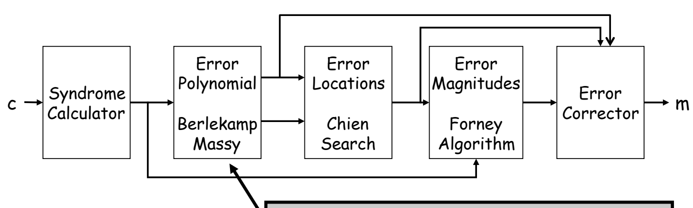

Decoding

Why is it hard?

Brute Force: try k+2s choose k + s possibilities and solve for each.

Efficient Decoding

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



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