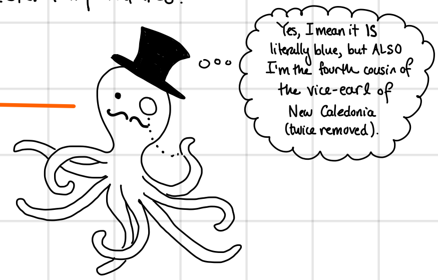
CS250/EE387 - LECTURE 5 - ALGORITHMS for REED-SOLOMON CODES!

AGENDA

(0) Recall RS codes
(1) Berlekamp-Welch
BONUS:
(2) Berlekamp-Massey [sketch]

TODAY'S OCTOPUS FACT

Octopusses have blue blood. That's because they have a copper-based system (rather than an iron-based system like we do), which is more efficient in cold temperatures.



(0) Recall the definition of RS codes:

DEF. (REED-SOLOMON CODES)

Let $n \geq k$, $q \geq n$. The REED-SOLOMON CODE of dimension k over $\mathbb{F}_q$, with evaluation points $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$, is

$$RS_q(\vec{\alpha}, n, k) = \{ (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) : f \in \mathbb{F}_q[X],\ \deg(f) \leq k-1 \}$$

Sometimes I'll just write $RS(n,k)$.

Last time, we saw that they meet the singleton bound.

HISTORIC ASIDE. RS codes were invented by Reed + Solomon in 1960.
At the time, they didn't have any fast decoding algs, so they were sort of neat but not that useful. But in the late 1960's, Peterson, Berlekamp-Massey developed an $O(n^2)$-time alg, which can be made to run in time $O(n\log(n))$ with FFT tricks.
Then RS codes started to be used all over the place! CDs, satellites, QR codes,...
In 1986, Welch + Berlekamp came up w/ another decoding alg — it's a bit slower but it is really pretty, so we'll start with that.

① Welch–Berlekamp Algorithm

Problem (Decoding $RS_q(\vec{\alpha}, n, k)$ from $e \leq \lfloor \frac{n-k}{2} \rfloor$ errors)

Given $w = (w_1, \ldots, w_n) \in \mathbb{F}_q^n$, find a polynomial $f \in \mathbb{F}_q[X]$ so that:

- $\deg(f) < k$
- $f(\alpha_i) \neq w_i$ for at most $e \leq \lfloor \frac{n-k}{2} \rfloor$ values of i,

or else return $\perp$ if no such polynomial exists.

Idea: Consider the polynomial $E(X) = \prod_{i : w_i \neq f(\alpha_i)} (X - \alpha_i)$.

This is called the "error locator polynomial." (Notice that we don't know what it is...)

Then $\forall i$, $w_i \cdot E(\alpha_i) = \underbrace{f(\alpha_i) \cdot E(\alpha_i)}_{\text{Call this } Q(\alpha_i)}$

Algorithm (Berlekamp-Welch)

① Find:

- a monic degree e polynomial $E(X)$
- a deg $\leq e+k-1$ polynomial $Q(X)$

so that: $w_i \cdot E(\alpha_i) = Q(\alpha_i) \ \forall i$ $\quad (*)$

If it doesn't exist, RETURN $\perp$.

② Let $\tilde{f}(X) = Q(X)/E(X)$

If $\Delta(\tilde{f}, w) > e$:

    RETURN $\perp$

RETURN $\tilde{f}$

TWO QUESTIONS:

1. How do we find such polys?
2. Once we do, why is it correct to return Q/E? What if we didn't find the "correct" Q and E?

Let's answer QUESTION 2 first.

CLAIM. If there is a degree $\le k-1$ poly f s.t. $\Delta(f,w) \le e$, then there exists E and Q satisfying $(*)$, and with $f(X) = Q(X)/E(X)$.

proof. Let $E(X) = \left[\prod_{i:\, w_i \ne f(\alpha_i)} (X-\alpha_i)\right] \cdot X^{e-\Delta(f,w)}$

Let $Q(X) = E(X) \cdot f(X)$.

CLAIM. Suppose that (E_1, Q_1), (E_2, Q_2) BOTH satisfy the requirements in STEP 1. Then:

$$\frac{Q_1(X)}{E_1(X)} = \frac{Q_2(X)}{E_2(X)}$$

proof. Consider $R(X) = \underbrace{Q_1(X)}_{\deg \le e+k-1}\underbrace{E_2(X)}_{\deg e} - Q_2(X)E_1(X)$

$\deg(R) \le 2e+k-1$, and $\forall i \in \{1,\dots,n\}$,

$$R(\alpha_i) = [w_i \cdot E_1(\alpha_i)] \cdot E_2(\alpha_i) - [w_i \cdot E_2(\alpha_i)] \cdot E_1(\alpha_i) = 0$$

This is where we need $e \le \lfloor \frac{n-k}{2} \rfloor$

Hence R has at least n roots. Since $e < \frac{n-k+1}{2}$, $2e+k-1 < n$.

So $R \equiv 0$ is the all-zero polynomial. (Low degree polynomials don't have too many roots!) ∎

Together, these CLAIMS answer QUESTION 2.

Moving on to QUESTION 1. How do we find E, Q? POLYNOMIAL INTERPOLATION!

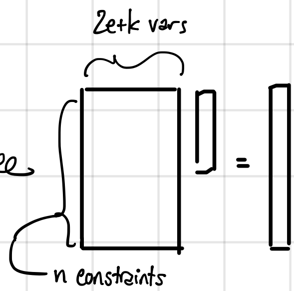
More precisely, we want:

$$\underbrace{w_i \cdot E(\alpha_i) = Q(\alpha_i) \text{ for } i = 1, \dots, n}_{n \text{ linear constraints.}}, \qquad \underbrace{\begin{array}{l} \deg(E) = e, \quad E \text{ monic} \\ \deg(Q) \leq e + k - 1. \end{array}}_{e + (e+k) = 2e+k \text{ variables,}}$$

which are the coefficients on these two polynomials.

We already know (from CLAIM 1) that a solution exists (assuming f does). So solve this system of eqs. to find it!

[Notice that $2e + k < (n-k+1) + k \leq n$, so the system looks like



But we don't actually care if the system is over or under-determined, now that we know that a solution exists and that any solution will do.]

RUNNING TIME of BERLEKAMP-WELCH:

- Step 1 takes time $O(n^3)$ for polynomial division
- Step 2 takes time $O(n^3)$ for Gaussian Elimination

$\Rightarrow O(n^3)$ total.

NOTE: The videos only cover up to this point. We may talk about the stuff below in class.

② BERLEKAMP-MASSEY (sketch)

PROBLEM (DECODING $RS_q(\vec{\alpha}, n, k)$ from $e \le \lfloor \frac{n-k}{2} \rfloor$ ERRORS)

Given $w = (w_1, \ldots, w_n) \in \mathbb{F}_q^n$, find a polynomial $f \in \mathbb{F}_q[X]$ so that:

- $\deg(f) < k$
- $f(\alpha_i) \neq w_i$ for at most $e \le \lfloor \frac{n-k}{2} \rfloor$ values of i,

or else return $\perp$ if no such polynomial exists.

Again we solve this PROBLEM

The Berlekamp-Massey algorithm is more efficient than the Berlekamp-Welch alg, especially when the #errors is small. Also, it turns out to be really nice to implement in hardware, although we won't go into that.

While the BM algorithm can be made to work on any RS code, we'll focus on the case where $n = q-1$ and the evaluation pts are $\gamma, \gamma^2, \ldots, \gamma^{q-1}$ for a primitive element $\gamma \in \mathbb{F}_q$.

Recall that in this case, the parity-check matrix for the RS code is:

$$H = \underbrace{\begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 & \cdots & \gamma^n \\ 1 & \gamma^2 & \gamma^4 & & & \gamma^{2n} \\ \vdots & \vdots & \vdots & & & \vdots \\ 1 & \gamma^{d-1} & \gamma^{2(d-1)} & & & \gamma^{n(d-1)} \end{bmatrix}}_{n} \left.\vphantom{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}\right\} d-1$$

where $d = n-k+1$ is the distance of the code.

Our goal is to decode from $e \le \lfloor \frac{d-1}{2} \rfloor$ errors.

Suppose we receive the vector $v = c + p \in \mathbb{F}_q^n$, where

$$c \in RS(\mathbb{F}_q^*, n, k) \quad \text{and} \quad wt(p) = e \le \left\lfloor \frac{d-1}{2} \right\rfloor.$$

Let $E = supp(p) \subseteq \{1, \ldots, n\}$ be the locations of the errors.

The Berlekamp-Massey algorithm is a SYNDROME DECODING algorithm. That is, we begin by computing the syndrome

$$Hv = Hc + Hp = Hp.$$

Now, because of the structure of H, we observe that

$$Hp = \begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 & \cdots & \gamma^n \\ 1 & \gamma^2 & \gamma^4 & & & \gamma^{2n} \\ \vdots & \vdots & \vdots & & & \vdots \\ 1 & \gamma^{d-1} & \gamma^{2(d-1)} & & & \gamma^{n(d-1)} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} = \begin{pmatrix} p(\gamma) \\ p(\gamma^2) \\ \vdots \\ p(\gamma^{d-1}) \end{pmatrix}$$

where $p(X) = \sum_i p_i X^i$.

$$= \sum_{i \in E} p_i X^i$$

Now let's define a polynomial $S(Z) := \sum_{\ell=1}^{d-1} p(\gamma^\ell) \cdot Z^\ell$.

Notice that we can compute $S(Z)$ given the syndrome. We have:

$$S(Z) = \sum_{\ell=1}^{d-1} p(\gamma^\ell) Z^\ell$$

$\longleftarrow$ We will also write $S(Z) = \sum_{\ell=1}^{d-1} S_\ell Z^\ell$, so $S_\ell := p(\gamma^\ell)$ is the ℓ^{th} syndrome symbol.

$$= \sum_{\ell=1}^{d-1} \sum_{i \in E} p_i \gamma^{\ell i} Z^\ell$$

$$= \sum_{i \in E} p_i \sum_{\ell=1}^{d-1} (\gamma^i Z)^\ell$$

$$= \sum_{i \in E} p_i \left(\frac{\gamma^i Z - (\gamma^i Z)^d}{1 - \gamma^i Z} \right)$$

$\swarrow$ using the fact that $\sum_{\ell=1}^{d-1} y^\ell = \frac{y - y^d}{1 - y}$ for any y.

This form of $S(Z)$ is a bit mucky... let's clear the denominators!

Let $\sigma(Z) = \prod_{i\in E} (1-\gamma^i Z)$.

Notice that $\sigma(Z)$ is an error-locator polynomial, in the sense that

$$i \in E \iff \sigma(\gamma^{-i}) = 0.$$

Part of our goal is to recover $\sigma(Z)$, which will tell us the locations of the errors.

Now, consider

$$\sigma(Z)\cdot S(Z) = \prod_{i\in E}(1-\gamma^i Z)\cdot \sum_{i\in E} p_i \left(\frac{\gamma^i Z - (\gamma^i Z)^d}{1-\gamma^i Z}\right)$$

$$= \sum_{i\in E} p_i \left(\gamma^i Z - (\gamma^i Z)^d\right) \prod_{j\in E\setminus\{i\}} (1-\gamma^j Z)$$

Foreshadowing: we will call this deg $\leq e$ poly $\omega(Z)$. It will come in handy soon.

$$= \left(\text{some degree} \leq e \text{ polynomial}\right) + Z^d \cdot \left(\text{some other polynomial}\right)$$

Write $S(Z)\cdot\sigma(Z) = \sum_{i=0}^{d+e} a_i Z^i$ for some coefficients a_i.
We don't know what those coefficients are, but we know that they look like this:

$$\underbrace{a_0 \quad a_1 \quad a_2 \quad \cdots \quad a_e}_{\text{these are for the degree} \leq e \text{ polynomial}} \quad \underbrace{a_{e+1} \quad a_{e+2} \quad \cdots \quad a_{d-1}}_{\text{THESE ARE ALL ZERO!!}} \quad \underbrace{a_d \quad a_{d+1} \quad \cdots}_{\text{these are for } Z^d\cdot(\text{some other polynomial})}$$

In particular, the coeffients on $Z^{e+1}, \ldots, Z^{d-1}$ in $S(Z)\sigma(Z)$ are all zero!

This gives us a linear system that we can set up to solve for (hopefully) σ:

Let $e+1 \leq r \leq d-1$. So

$$0 = \begin{pmatrix} \text{coefficient on} \\ Z^r \text{ in } S(Z)\sigma(Z) \end{pmatrix} = \sum_{i=0}^{e} \sigma_i S_{r-i}$$

where $\sigma(Z) = \sum_{i=0}^{e} \sigma_i Z^i$, and $S(Z) = \sum_{i=1}^{d-1} S_i Z^i$

Since this is true for all such r, we can set up a system of eqns that the coefficients of $\sigma(Z)$ must satisfy:

$$d-e-1\left\{\underbrace{\begin{bmatrix} S_{e+1} & S_e & S_{e-1} & \cdots & & S_1 \\ S_{e+2} & S_{e+1} & S_e & S_{e-1} & & \\ S_{e+3} & S_{e+2} & S_{e+1} & S_e & S_{e-1} & \\ & S_{e+3} & S_{e+2} & \ddots & \ddots & \ddots \\ \vdots & & S_{e+3} & \ddots & & \\ & & & \ddots & & \\ S_{d-1} & & & & & \end{bmatrix}}_{e+1}\right. \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \\ \sigma_e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \\ \\ 0 \end{bmatrix}$$

The row of this matrix that starts with S_r is the constraint corresponding to the coefficient on Z^r.

So the next step of this algorithm is to solve this system of equations to recover $\tilde{\sigma}_0, \tilde{\sigma}_1, \ldots, \tilde{\sigma}_e$, and a corresponding polynomial $\tilde{\sigma}(Z)$.

We know that a solution exists since σ itself is a solution. But why should $\tilde{\sigma} = \sigma$? Actually, it may not... but we will still be able to use it, as we will see below.

Let's take a brief detour to define another useful polynomial. Let

$$\omega(Z) = \sum_{i \in E} p_i \gamma^i Z \prod_{j \in E \setminus \{i\}} (1 - \gamma^j Z)$$

Notice that $\omega(Z) = \begin{pmatrix} \text{that polynomial of degree} \le e \\ \text{from earlier} \end{pmatrix}$.

That is, $\omega(Z) \equiv \sigma(Z) S(Z) \bmod Z^d$. Further, for any $\ell \in E$,

$$\omega(\gamma^{-\ell}) = \sum_{i \in E} p_i \gamma^{i-\ell} \underbrace{\prod_{j \in E \setminus \{i\}} (1 - \gamma^{j-\ell})}_{\substack{\text{this will vanish unless } \ell = i \\ (\text{assuming } \ell \in E)}}$$

$$= p_\ell$$

So $\sigma(Z)$ tells us WHERE the errors occur, and $\omega(Z)$ tells us what the values of those errors are.

Moreover, this also tells us that $\sigma(Z)$ and $\omega(Z)$ are relatively prime – that is, they have no common factors. Indeed, since $\sigma(Z) = \prod_{i \in E} (1 - \gamma^i Z)$ is a product of degree-1 factors, if $\sigma(Z)$ and $\omega(Z)$ had any factors in common then $\omega(\gamma^{-i}) = 0$ for some $i \in E$. But we just saw that $\omega(\gamma^{-i}) = p_i \neq 0$ for any $i \in E$.

Now, back to our solution $\tilde{\sigma}(Z)$ to those linear equations.

Let $\tilde{\omega}(Z) = \tilde{\sigma}(Z) \cdot S(Z) \mod Z^d$.

That is, multiply out $\tilde{\sigma}(Z)S(Z)$ and throw away all terms of degree $\geq d$.

We have

$$S(Z)\sigma(Z)\tilde{\sigma}(Z) = (S(Z)\sigma(Z)) \cdot \tilde{\sigma}(Z)$$
$$\equiv \omega(Z) \cdot \tilde{\sigma}(Z) \mod Z^d$$

and similarly,

$$S(Z)\sigma(Z)\tilde{\sigma}(Z) = (S(Z)\tilde{\sigma}(Z)) \cdot \sigma(Z)$$
$$\equiv \tilde{\omega}(Z) \cdot \sigma(Z) \mod Z^d.$$

Moreover, $\deg(\omega), \deg(\tilde{\omega}), \deg(\sigma), \deg(\tilde{\sigma}) \leq e$, and so

$$\deg(\tilde{\omega}(Z) \cdot \sigma(Z)), \deg(\omega(Z) \cdot \tilde{\sigma}(Z)) \leq 2e < d$$

Using our assm that $e \leq \lfloor \frac{d-1}{2} \rfloor$.

Thus, not only are $\tilde{\omega}(Z)\sigma(Z)$ and $\omega(Z)\tilde{\sigma}(Z)$ equal mod Z^d, but they are actually equal!

We conclude that

$$\frac{\tilde{\omega}(Z)}{\tilde{\sigma}(Z)} = \frac{\omega(Z)}{\sigma(Z)}.$$

We have solved for $\tilde{\omega}, \tilde{\sigma}$, and we know that $\gcd(\omega, \sigma) = 1$, so we can find ω and σ from $\tilde{\omega}$ and $\tilde{\sigma}$ by computing $\gcd(\tilde{\omega}, \tilde{\sigma})$ and dividing it out.

So, our algorithm is the following:

ALGORITHM Slow_Berlekamp_Massey(v):

- Compute the syndrome $S = H \cdot v = \begin{pmatrix} S_1 \\ \vdots \\ S_{d-1} \end{pmatrix}$
- Solve the system of linear equations

$$\begin{bmatrix} S_{e+1} & S_e & S_{e-1} & \cdots & S_1 \\ S_{e+2} & S_{e+1} & S_e & S_{e-1} & \\ S_{e+3} & S_{e+2} & S_{e+1} & S_e & S_{e-1} \\ & S_{e+3} & S_{e+2} & \ddots & \ddots \\ \vdots & & S_{e+3} & \ddots & \\ S_{d-1} & & & & \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

to find $\tilde{\sigma}(Z)$

- Let $S(Z) = \sum_{\ell=1}^{d-1} S_\ell Z^\ell$

  and compute $\tilde{\omega}(Z) = S(Z)\tilde{\sigma}(Z) \bmod Z^d$
- Find $g(Z) = \gcd(\tilde{\sigma}(Z), \tilde{\omega}(Z))$
- Let $\sigma(Z) = \tilde{\sigma}(Z)/g(Z)$, $\omega(Z) = \tilde{\omega}(Z)/g(Z)$
- Factor $\sigma(Z)$ to find roots $\{\gamma^{-i} : i \in E\}$ (defining E)
- Define $p_i = \omega(\gamma^i)$ for $i \in E$ (and $p_i = 0$ if $i \notin E$)
- Let $p = (p_1, p_2, \ldots, p_n)$ and return $c = v + p$.

You'll notice that this alg is called "Slow_Berlekamp_Massey."

As written, we need to:

- solve a $\approx d \times d$ linear system ← If $d \ll n$, this is much smaller than the Berlekamp-Welch system of eqs.
- Do some polynomial multiplication/division of degree $O(d)$ polynomials
- Find the gcd of some polynomials of degree $O(d)$
- Factor a polynomial of degree $O(d)$.

If d is small, this is actually pretty fast, $\mathrm{poly}(d \log n)$

The $\log(n)$ is b/c $q \approx n$ and we need to deal with elements of $\mathbb{F}_q$

In particular, except for computing the syndrome Hv, this can be sublinear in n!

It turns out that we can do much better, because the linear system is so structured. Given the syndrome Hv, we can find v in time

$$O\left(d \log^2(d) \cdot \log\log(d) + d^2 \log(n) \right) \text{ operations in } \mathbb{F}_q.$$

See [Dodis, Ostrovsky, Reyzin, Smith 2006] for more details.

They also discuss a "dual view" of the alg. which is a more traditional way to present it.

QUESTIONS TO PONDER

1. Find a more efficient way to implement the Berlekamp-Massey alg.
2. Can you think of any other algs for RS codes?
3. How would you adapt RS codes / these algorithms to come up with BINARY codes?