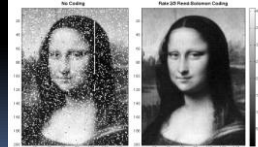
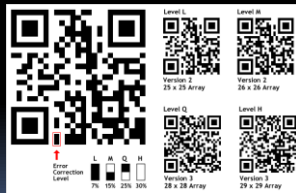


Error Correction



Polynomial Interpolation

Theorem:

Let arbitrary pairs $(a_1, b_1), (a_2, b_2), \dots, (a_{d+1}, b_{d+1})$ from a field F be given (with all a_i 's distinct). Then there always **exists** a polynomial $P(x)$ of **degree $\leq d$** with $P(a_i) = b_i$ for all i .

Lagrange Interpolation

a_1	b_1
a_2	b_2
a_3	b_3
...	...
a_d	b_d
a_{d+1}	b_{d+1}

Want $P(x)$
(with degree $\leq d$)
such that $P(a_i) = b_i \forall i$.

Lagrange Interpolation

a_1	1
a_2	0
a_3	0
...	...
a_d	0
a_{d+1}	0



Can we do this special case?

Lagrange Interpolation

Numerator is a deg. d polynomial	a_1	1	Denominator is a nonzero field element
	a_2	0	
	a_3	0	
	
	a_d	0	
	a_{d+1}	0	

Idea #2:

$$S_1(x) = \frac{(x - a_2)(x - a_3) \cdots (x - a_{d+1})}{(a_1 - a_2)(a_1 - a_3) \cdots (a_1 - a_{d+1})}$$

Call this the **selector polynomial** for a_1 .

a_1	0
a_2	1
a_3	0
...	...
a_d	0
a_{d+1}	0

What about above data?

$$S_2(x) = \frac{(x - a_1)(x - a_3) \cdots (x - a_{d+1})}{(a_2 - a_1)(a_2 - a_3) \cdots (a_2 - a_{d+1})}$$

a_1	0
a_2	0
a_3	0
...	...
a_d	0
a_{d+1}	1

And for this data,

$$S_{d+1}(x) = \frac{(x - a_1)(x - a_2) \cdots (x - a_d)}{(a_{d+1} - a_1)(a_{d+1} - a_2) \cdots (a_{d+1} - a_d)}$$

Polynomial Interpolation

a_1	b_1
a_2	b_2
a_3	b_3
...	...
a_d	b_d
a_{d+1}	b_{d+1}

$$P(x) = b_1 \cdot S_1(x) + b_2 \cdot S_2(x) + \cdots + b_{d+1} \cdot S_{d+1}(x)$$

The Chinese Remainder Theorem had a very similar proof



Not a coincidence:
algebraically, integers & polynomials share many common properties

Lagrange interpolation is the *exact analog* of Chinese Remainder Theorem for polynomials.

Chinese Remainder Theorem: Suppose n_1, n_2, \dots, n_k are pairwise coprime. Then, for all integers a_1, a_2, \dots, a_k , there exists an integer x solving the below system of simultaneous congruences

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

Further, all solutions x are congruent modulo $N = \prod_{i=1}^k n_i$.

Let $m_i = N/n_i$

i 'th "selector" number: $T_i = (m_i^{-1} \pmod{n_i}) m_i$

$$x = a_1 T_1 + a_2 T_2 + \dots + a_k T_k$$

Recall: Interpolation

Let pairs $(a_1, b_1), (a_2, b_2), \dots, (a_{d+1}, b_{d+1})$ from a field F be given (with all a_i 's distinct).

Theorem:

There is a unique degree d polynomial $P(x)$ satisfying $P(a_i) = b_i$ for all $i = 1 \dots d+1$.

A linear algebra view

Let $p(x) = p_0 + p_1 x + p_2 x^2 + \dots + p_d x^d$
Need to find the coefficient vector (p_0, p_1, \dots, p_d)

$$\begin{aligned} p(a) &= p_0 + p_1 a + \dots + p_d a^d \\ &= 1 \cdot p_0 + a \cdot p_1 + a^2 \cdot p_2 + \dots + a^d \cdot p_d \end{aligned}$$

Thus we need to solve:

$$\begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^d \\ 1 & a_2 & a_2^2 & \cdots & a_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_{d+1} & a_{d+1}^2 & \cdots & a_{d+1}^d \end{pmatrix} \cdot \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{d+1} \end{pmatrix}$$

Lagrange interpolation

The $(d + 1) \times (d + 1)$ Vandermonde matrix

$$M = \begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^d \\ 1 & a_2 & a_2^2 & \cdots & a_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_{d+1} & a_{d+1}^2 & \cdots & a_{d+1}^d \end{pmatrix}$$

is invertible.

- The determinant of M is nonzero when a_i 's are distinct.

Thus can recover coefficient vector as $\vec{p} = M^{-1}\vec{b}$

The columns of M^{-1} are given by the coefficients of the various "selector" polynomials we constructed in Lagrange interpolation.

Representing Polynomials

Let $P(x) \in \mathbb{F}[x]$ be a degree- d polynomial.

Representing $P(x)$ using $d+1$ field elements:

- List the $d+1$ coefficients.
- Give P 's value at $d+1$ different elements.

Rep 1 to Rep 2: Evaluate at $d+1$ elements

Rep 2 to Rep 1: Lagrange Interpolation

Application of Fields/Polynomials (and linear algebra): Error-correcting codes

Sending messages on a noisy channel

Alice

Bob

" bit.ly/vrxUBN "



The channel may corrupt up to k symbols.

How can Alice still get the message across?

Sending messages on a noisy channel

Let's say messages are sequences from \mathbb{F}_{257}

vrxUBN ↔ 118 114 120 85 66 78

noisy channel ↓

vrxUBN ↔ 118 114 104 85 35 78

The channel may corrupt up to k symbols.

How can Alice still get the message across?

Sending messages on a noisy channel

Let's say messages are sequences from \mathbb{F}_{257}

vrxUBN ↔ 118 114 120 85 66 78

noisy channel ↓

vrxUBN ↔ 118 114 104 85 35 78

How to correct the errors?

How to even detect that there are errors?

Simpler case: "Erasures"

118 114 120 85 66 78
 erasure channel ↓
 118 114 ?? 85 ?? 78

What can you do to handle up to k erasures?

Repetition code

Have Alice **repeat** each symbol $k+1$ times.

118 114 120 85 66 78
 becomes
 118 118 118 114 114 114 120 120 120 85 85 85 66 66 66 78 78 78
 erasure channel ↓
 118 118 118 ?? ?? 114 120 120 120 85 85 85 66 66 66 78 78 78

If at most k erasures, Bob can figure out each symbol.

Repetition code – noisy channel

Have Alice **repeat** each symbol $2k+1$ times.

118 114 120 85 66 78
 becomes
 118 118 118 114 114 114 120 120 120 85 85 85 66 66 66 78 78 78
 noisy channel ↓
 118 118 118 114 223 114 120 120 120 85 85 85 66 66 66 78 78 78

At most k corruptions: Bob can take **majority** of each block.

This is pretty wasteful!

To send **message** of $d+1$ symbols and guard against k erasures, we had to send $(d+1)(k+1)$ **total** symbols.

Can we do better?

This is pretty wasteful!

To send **message** of $d+1$ symbols and guard against k erasures, we had to send $(d+1)(k+1)$ **total** symbols.

To send even **1 message symbol** with k erasures, **need** to send $k+1$ **total** symbols.

Maybe for $d+1$ **message symbols** with k erasures, $d+k+1$ **total** symbols can suffice??

Enter polynomials

Say Alice's message is $d+1$ elements from \mathbb{F}_{257}

118 114 120 85 66 78

Alice thinks of it as the **coefficients** of a degree- d polynomial $P(x) \in \mathbb{F}_{257}[x]$

$$P(x) = 118x^5 + 114x^4 + 120x^3 + 85x^2 + 66x + 78$$

Now trying to send the degree- d polynomial $P(x)$.

Send it in the Values Representation!

$$P(x) = 118x^5 + 114x^4 + 120x^3 + 85x^2 + 66x + 78$$

Alice sends $P(x)$'s values on $d+k+1$ inputs:

$$P(1), P(2), P(3), \dots, P(d+k+1)$$

This is called the **Reed-Solomon encoding**.



Send it in the Values Representation!

$$P(x) = 118x^5 + 114x^4 + 120x^3 + 85x^2 + 66x + 78$$

Alice sends $P(x)$'s values on $d+k+1$ inputs:

$$P(1), P(2), P(3), \dots, P(d+k+1)$$

If there are at most k erasures, then Bob still knows P 's value on $d+1$ points.

Bob recovers $P(x)$ using **Lagrange Interpolation!**

Example

Reed-Solomon codes are used a lot in practice!

Another everyday use: CD/DVDs, hard discs, satellite communication, ...



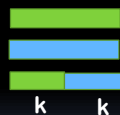
Maxicodes
= "UPS codes"
= another 2-d
Reed-Solomon codes

PDF417 codes
= 2-d Reed-Solomon
codes

What about corruptions/errors

To send message of $d+1$ symbols and enable correction from up to k errors, repetition code has to send $(d+1)(2k+1)$ total symbols.

To even communicate 1 symbol while enabling recovery from k errors, need to send $2k+1$ total symbols.



Maybe for $d+1$ message symbols with k errors, $d+2k+1$ total symbols can suffice??

Want to send a polynomial of degree- d subject to at most k corruptions.

First simpler problem: Error detection

Suppose we try the same idea

- Evaluate $P(x)$ at $d+1+k$ points
- Send $P(0), P(1), P(2), \dots, P(d+k)$

At least $d+1$ of these values will be unchanged

Example

$P(X) = 2X^2 + 1$, and $k = 1$.

So I sent $P(0)=1, P(1)=3, P(2)=9, P(3)=19$

Corrupted email says (1, 4, 9, 19)

Choosing (1, 4, 9) will give us $Q(X) = X^2 + 2X + 1$

We can now detect (up to k) errors

Evaluate $P(X)$ at $d+1+k$ points

Send $P(0), P(1), P(2), \dots, P(d+k)$

At least $d+1$ of these values will be correct

Say $P(0), P'(1), P(2), P(3), P'(4), \dots, P(d+k)$

Using these $d+1$ correct values will give $P(X)$

Using any of the incorrect values will give something else

Quick way of detecting errors

Interpolate first $d+1$ points to get $Q(X)$

Check that all other received values are consistent with this polynomial $Q(X)$

If all values consistent, no errors.

In that case, we know $Q(X) = P(X)$
else there were errors...

How good is our encoding?

Naïve Repetition:

To send $d+1$ numbers with error detection,
sent $(d+1)(k+1)$ numbers

Polynomial Coding:

To send $d+1$ numbers with error detection,
sent $(d+k+1)$ numbers

How about error correction?

requires more work

To send $d+1$ numbers in such a way
that we can correct up to k errors,
need to send $d+1+2k$ numbers.

Similar encoding scheme

Evaluate degree- d $P(x)$ at $d+1+2k$ points

Send $P(0), P(1), P(2), \dots, P(d+2k)$

At least $d+1+k$ of these values will be correct

Say $P(0), P(1), P(2), P(3), P(4), \dots, P(d+2k)$

Trouble: How do we know which are correct?

Theorem: $P(X)$ is the unique degree- d polynomial that agrees with the received data on at least $d+1+k$ points

Clearly, the original polynomial $P(X)$ agrees with data on $d+1+k$ points (since at most k errors, out of total $d+1+2k$ points)

And if a different degree- d polynomial $R(X)$ did so, $R(X)$ and $P(X)$ would have to agree with each other on $d+1$ points, and hence be the same.

So any such $R(X) = P(X)$

Theorem: $P(X)$ is the unique degree- d polynomial that agrees with the received data on at least $d+1+k$ points

Brute-force Algorithm to find $P(X)$:

Interpolate each subset of $(d+1)$ points

Check if the resulting polynomial agrees with received data on $d+1+k$ pts

Takes too much time...

A fast (cubic runtime) algorithm to decode was given by [Peterson, 1960]

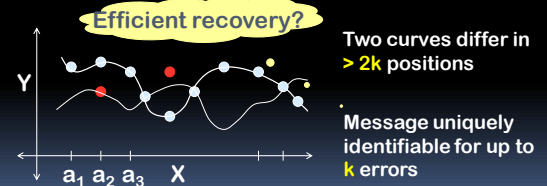
Later improvements by Berlekamp and Massey gave practical algorithms

We will now describe the Welch-Berlekamp algorithm to recover the original polynomial when there are k errors

Aside: Recent research (incl. some of my own) has given algorithms to correct even more than k errors (in a meaningful model)

Reed-Solomon codes

- Message = $(m_0, m_1, \dots, m_d) \in F^{d+1}$
- $(F = \mathbb{Z}_p)$ $(P(a_1), P(a_2) \dots P(a_n))$
- Polynomial curve $Y = P(X) = m_0 + m_1X + \dots + m_dX^d$
- Encoding = eval. at $n = d+2k+1$ distinct $a_i \in F$



Error-correction approach

- Given $n = d+2k+1$ points (a_i, y_i) where received value $y_i \neq P(a_i)$ for at most k points.
- If we locate positions of errors, problem then easy by interpolation on correct data



Locating the errors

Noisy points mess up curve-fitting/interpolation

Interpolate more general curve through all n points

Here's such a curve.
 $(Y - P(X))(X - a_2)(X - a_5) = 0$

Here $E(X) := (X - a_2)(X - a_5)$ is the error locator polynomial, with roots at all error locations

Of course, we don't know $E(X)$ (that's the whole problem!)

Developing the algorithm

Let Err be subset of k erroneous locations, and define error locator polynomial $E(X) = \prod_{i \in \text{Err}} (X - a_i)$

- $\text{degree}(E) = k$

We have $E(a_i) y_i = E(a_i) P(a_i)$ for $i=1,2,\dots,n$

Let $N(X) = E(X) P(X)$; $\text{degree}(N) = d + k$.

So $E(a_i) y_i - N(a_i) = 0$ for all points (a_i, y_i)

Can we use above to find polynomials $E(X)$ and $N(X)$ (and hence also $P(X) = N(X)/E(X)$)?

Finding N and E

$E(a_i) y_i - N(a_i) = 0$ for all points (a_i, y_i)

- $E(X) = X^k + b_{k-1} X^{k-1} + \dots + b_0 =$
- $N(X) = c_{d+k} X^{d+k} + \dots + c_1 X + c_0 =$

Finding $E(X)$ and $N(X)$ is same as finding the unknowns $b_0, b_1, \dots, b_{k-1}, c_0, \dots, c_{d+k}$

- There are $k + (d+k+1) = d+2k+1 = n$ unknowns
- Also n **linear** equations $E(a_i) y_i - N(a_i) = 0$ in these n unknowns (why are they linear?)

So we can find $E(X)$ and $N(X)$ by solving this linear system and then output $N(X)/E(X)$

Spurious solutions?

We know coefficients of $E(X)$ and $N(X)=E(X)P(X)$ are a solution, but what if there are other solutions?

Lemma: If $E_1(X)$ and $N_1(X)$ are a different solution, to $E_1(a_i) y_i - N_1(a_i) = 0$ with $\text{deg}(E_1) \leq k$, $\text{deg}(N_1) \leq d+k$, then $N_1(X)/E_1(X) = P(X)$

Proof: Define $R(X) = E_1(X)P(X) - N_1(X)$

When $P(a_i) = y_i$, $\text{degree}(R) \leq d+k$

$$R(a_i) = E_1(a_i)P(a_i) - N_1(a_i) = E_1(a_i)y_i - N_1(a_i) = 0$$

So $R(X)$ has at least $d+k+1$ roots. $\Rightarrow R(X) = 0$ \square

Thus every solution (E_1, N_1) to the linear system yields the same $P(X)$ as the ratio!

Sending messages on a noisy channel

Alice

Um, what if $d+2k+1 > 257$?

Message: $d+1$ symbols from \mathbb{F}_{257}

Reed-Solomon:

To guard against k corruptions, treat message as coeffs of poly P , send $P(1), P(2), \dots, P(d+2k+1)$

Sending messages on a noisy channel

Alice

Um, what if $d+2k+1 > 257$?

What if the noisy channel corrupts **bits**, not bytes?

(Can we have fewer redundant **bits**?)

\mathbb{F}_{257}

Against k corruptions, message as coeffs of poly P , send $P(1), P(2), \dots, P(d+2k+1)$

Sending messages on a noisy channel

Alice wants to send an n -bit message to Bob.

The channel may flip up to k bits.

How can Alice get the message across?

Sending messages on a noisy channel

Alice wants to send an $(n-1)$ -bit message to Bob.

The channel may flip up to 1 bit.

How can Alice get the message across?

Q1: How can Bob **detect** if there's been a bit-flip?

Parity-check solution

Alice tacks on a bit, equal to the parity of the message's $n-1$ bits.

Alice's n -bit 'encoding' always has an **even number of 1's**.

Bob can **detect** if the channel flips a bit: if he receives a string with an odd # of 1's.

1-bit error-detection for 2^{n-1} messages by sending n bits: **optimal!** (simple exercise)

Linear Algebra perspective

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

G: an $n \times (n-1)$ 'generator' matrix

Alice's message $x \in \mathbb{F}_2^{n-1}$

Bob receives

Linear Algebra perspective

Let **C** be the set of strings Alice may transmit.

C is the span of the columns of **G**.

C is a subgroup of \mathbb{F}_2^n
 [In linear algebra terms, an $(n-1)$ -dimensional subspace of the vector space \mathbb{F}_2^n]

Linear Algebra perspective

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-1} \\ z_n \end{bmatrix} \stackrel{?}{=} 0$$

H: a $1 \times n$ 'parity check' matrix

Bob receives

Bob checks this to detect if no errors

Solves 1-bit error **detection**, but not **correction**

If Bob sees $z = (1, 0, 0, 0, 0, 0, 0)$,

did Alice send $y = (0, 0, 0, 0, 0, 0, 0)$,
 or $y = (1, 1, 0, 0, 0, 0, 0)$,
 or $y = (1, 0, 1, 0, 0, 0, 0)$,
 or... ?

The Hamming(7,4) Code



Alice communicates 4-bit messages (16 possible messages) by transmitting 7 bits.

$$G' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Alice encodes $x \in \mathbb{F}_2^4$ by $G'x$, which looks like x followed by 3 extra bits.

The Hamming(7,4) Code



Alice sends 4-bit messages using 7 bits.

Any 'codeword' $y = Gx$ satisfies some 'parity checks':

$$y_1 = y_3 + y_5 + y_7$$

$$y_2 = y_3 + y_6 + y_7$$

$$y_4 = y_5 + y_6 + y_7$$

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

i.e., $Hy = 0$

Let's permute the output 7 bits (rows of G')

$G =$

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The Hamming(7,4) Code



Alice communicates 4-bit messages using 7 bits.

Columns are 1...7 in binary!

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$Hy = 0$, because $HG = 0$.

$G =$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

The Hamming(7,4) Code

On receiving $z \in \mathbb{F}_2^7$, Bob computes $H z$.

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

If no errors, $z = Gx$, so $H z = H G x = 0$.

If j th coordinate corrupted, $z = Gx + e_j$.

Then $H z = H(Gx + e_j) = H G x + H e_j = H e_j = (j\text{'th column of } H) = \text{binary rep. of } j$

Bob knows where the error is, can recover msg!

Sending longer messages: General Hamming Code

By sending $n = 7$ bits, Alice can communicate one of 16 messages, guarding against 1 error.

This scheme generalizes: Let $n = 2^r - 1$, take H to be the $r \times (2^r - 1)$ matrix whose columns are the numbers $1 \dots 2^r - 1$ in binary.

There are $2^{n-r} = 2^n / (n+1)$ solutions $z \in \{0,1\}^n$ to the check equations $H z = 0$.

- These are *codewords* of the Hamming code of length n

Summary: Hamming code

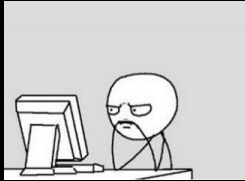
To **detect** 1 bit error in n transmitted bits:

- one parity check bit suffices,
- so can communicate 2^{n-1} messages by sending n bits.

To **correct** 1 bit error in n transmitted bits:

- for $n = 2^r - 1$, r check bits suffice
- so can communicate $2^n / (n+1)$ messages by sending n bits

Fact (left as exercise): This is optimal!



Study Guide

Polynomials:

Lagrange Interpolation
Parallel with Chinese Remainderin

Reed-Solomon codes:

Erasur correction via interpolation
Error correction

Hamming codes:

Correcting 1 bit error