Reed–Solomon Codes

Andrew Chang

July 11, 2022

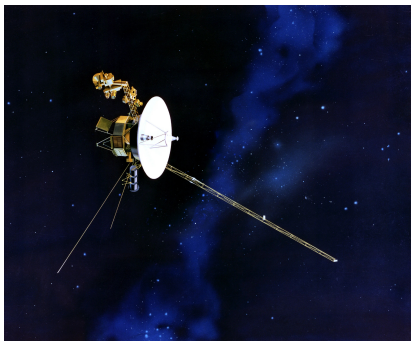
Outline

Development

- Reed–Solomon codes were developed at MIT by Irving S. Reed and Gustave Solomon in 1960
- Originally, they were designed to use a variable generator polynomial to encode a message, then decode potential polynomials by looking at subsets of the encoded message
- By 1963, a faster scheme was developed based on a fixed generator polynomial
  - This variation of RS codes is also technically a type of BCH code: a family of codes invented independently by A. Hocquenghem in 1959 and R.C. Bose & D.K. Ray-Chaudhuri in 1960

Applications over the years

- In 1977, Reed–Solomon codes found a major application in the communications for the Voyager program — transmitting data from interstellar space, tens of billions of km from the Sun
- Later applications include CDs, DVDs, QR codes, digital broadcasts, and storage systems
- Newer, related BCH codes are also popular in flash drives and SSDs
- Advantage when errors occur in bursts, since they can be corrected as a single error



Review: Galois Fields

- Reed–Solomon codes rely on finite field (aka Galois field) polynomials. To construct a finite field of size p^m we find a irreducible polynomial in $\mathbb{Z}_p[x]$ of degree m, and define

$$GF(p^m) = \mathbb{Z}_p[x]/(q(x)).$$

- For Reed–Solomon codes, we use a field $GF(2^m)$, so our polynomials will be of the form $a_0x^0 + a_1x^1 + \cdots + a_ix^i$, where a_i is either 0 or 1
- This means we can express each polynomial's coefficients as a corresponding binary string: $x^3 + x + 1 = 1011$
- Addition is the same as subtraction: $0 + 0 = 1 + 1 = 0, 1 + 0 = 0 + 1 = 1$ (equivalent to the XOR binary operation); multiplication and division is mod $q(x)$
- All nonzero elements can be expressed as powers of a primitive element α — a generator of the cyclic group $(GF(2^m)^*, \cdot)$

Constructing a codeword

- There are various encoding methods for Reed–Solomon codes: the original method was to interpret a binary message as the coefficients of a polynomial (so 1011 would be $x^3 + x + 1$) and evaluate the polynomial at several points — the values would be the codeword to be sent
- Another method uses Lagrange interpolation to calculate a polynomial with specific values at certain points, corresponding to the message
- Possibly the most common method is to interpret the message as a polynomial, but then multiply it by a generator polynomial to obtain the codeword: this is the "BCH view"

Codewords, continued

- Specifically for BCH: for a code with a block size of n, a message size of k, and a symbol size of q bits, we encode the message as a polynomial $p(x)$ and multiply by a generator polynomial

$$g(x) = (x - \alpha)(x - \alpha^2)\dots(x - \alpha^{n-k})$$

where α is a primitive element — in other words, the roots of this polynomial are designed to be consecutive elements of the Galois field

Correcting errors

- At least $t = n - k$ errors can be detected, and $\lfloor t/2 \rfloor$ can be corrected
- We will focus on the BCH-based decoders, since these are the fastest and most popular
- These in general work by evaluating the codeword at the zeros of the generator polynomial, which gives values (the "syndromes") that can be used to compute the number of errors as well as a polynomial that gives the locations of errors
- If values are all 0, there's no errors — otherwise we have to do some math
- Commonly used algorithms for finding error locations are PGZ (Peterson–Gorenstein–Zierler) and Berlekamp–Massey

PGZ algorithm

- Seek to calculate error locator polynomial coefficients: find

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \cdots + \lambda_v x^v$$

- Start by guessing $v = t$, and expect at least $2t$ syndromes $s_c, \ldots, s_{c+2t-1}$
- Build a matrix

$$S_{v \times v} = \begin{bmatrix} s_c & s_{c+1} & \cdots & s_{c+v-1} \\ s_{c+1} & s_{c+2} & \cdots & s_{c+v} \\ \vdots & \vdots & \ddots & \vdots \\ s_{c+v-1} & s_{c+v} & \cdots & s_{c+2v-2} \end{bmatrix}$$

PGZ algorithm, continued

- Also build vectors

$$C_{v\times 1} = \begin{bmatrix} s_{c+v} \\ s_{c+v+1} \\ \vdots \\ s_{c+2v-1} \end{bmatrix}, \Lambda_{v\times 1} = \begin{bmatrix} \lambda_v \\ \lambda_{v-1} \\ \vdots \\ \lambda_1 \end{bmatrix}$$

- Λ gives the coefficients of the unknown polynomial, so we try to solve the equation

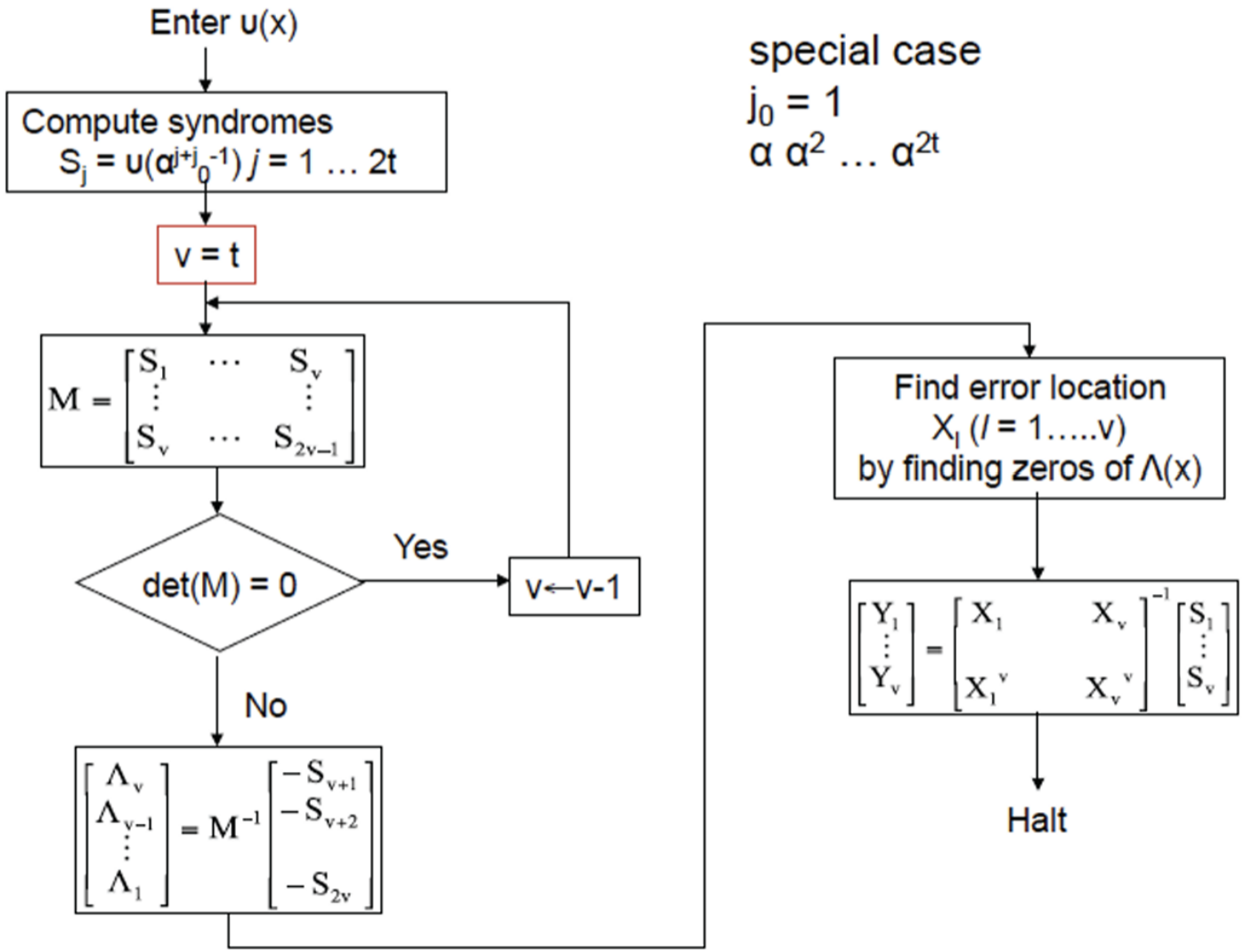
$$S_{v\times v}\Lambda_{v\times 1} = -C_{v\times 1}$$

- If the determinant of $S_{v\times v}$ is 0, keep lowering v until a solvable matrix is obtained — this will give the values of Λ and hence the error locator polynomial

PGZ algorithm, continued

- Finally, factoring the polynomial in the form $\Lambda(x) = \left(\alpha^{i_1}x - 1\right)\left(\alpha^{i_2}x - 1\right)\cdots\left(\alpha^{i_v}x - 1\right)$ (via other computer algorithms) gives the error locations as the powers $\alpha^{i_1}, \ldots, \alpha^{i_v}$
- Generally the last step is to solve another system of equations to get the error values
- For binary, just invert the bits at these locations, and errors are corrected — original message obtained!

PGZ algorithm flowchart

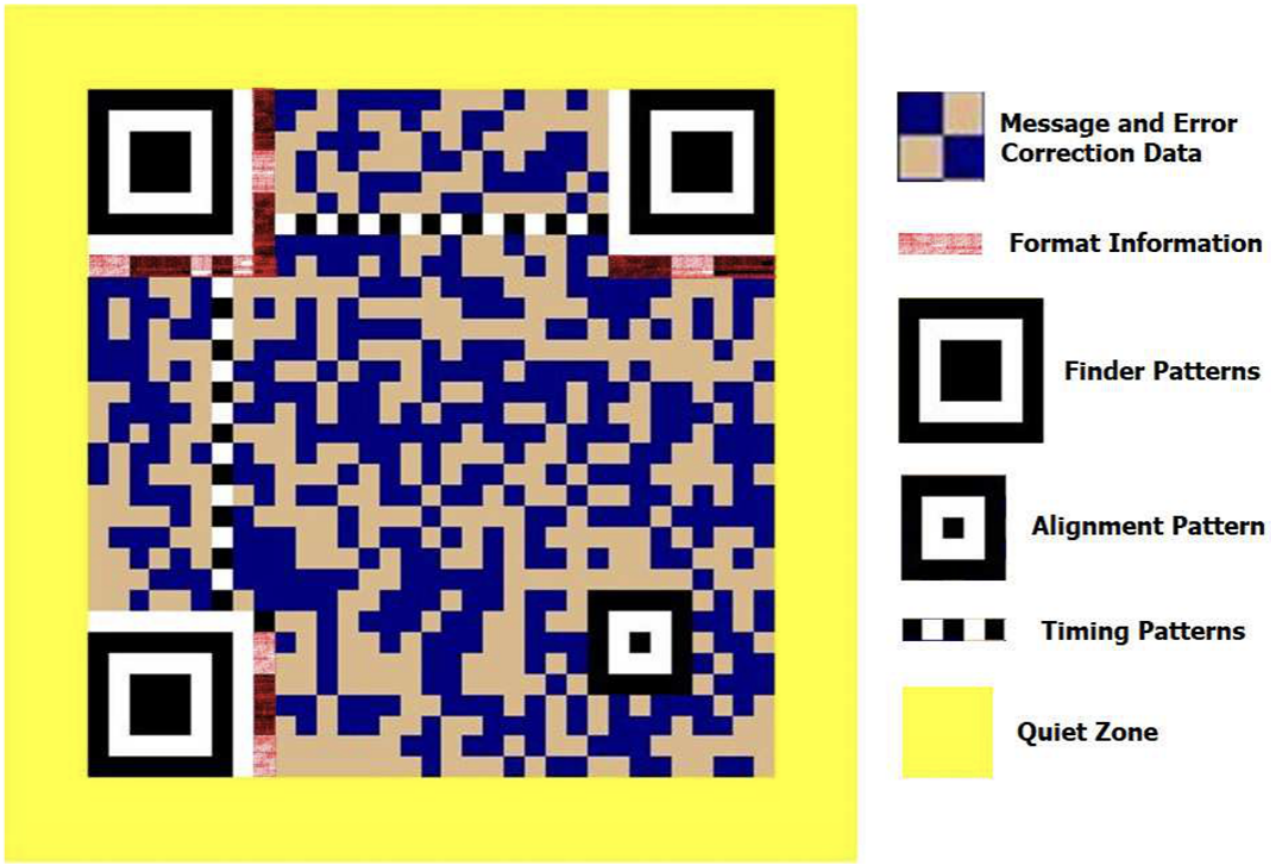


Introduction to QR codes

- QR (quick response) codes were invented in 1994 by the DENSO Corporation, a Toyota subsidiary
- They're present everywhere, for instance allowing people to quickly find a form or scan tickets just by taking a picture
- Reed–Solomon codes allow for flexibility: with a small increase in size, the error correction lets the data be retrieved even if many squares are unreadable
- Sometimes, a whole logo covers the middle of a QR code — with no effect on scanning!

Design of a QR code

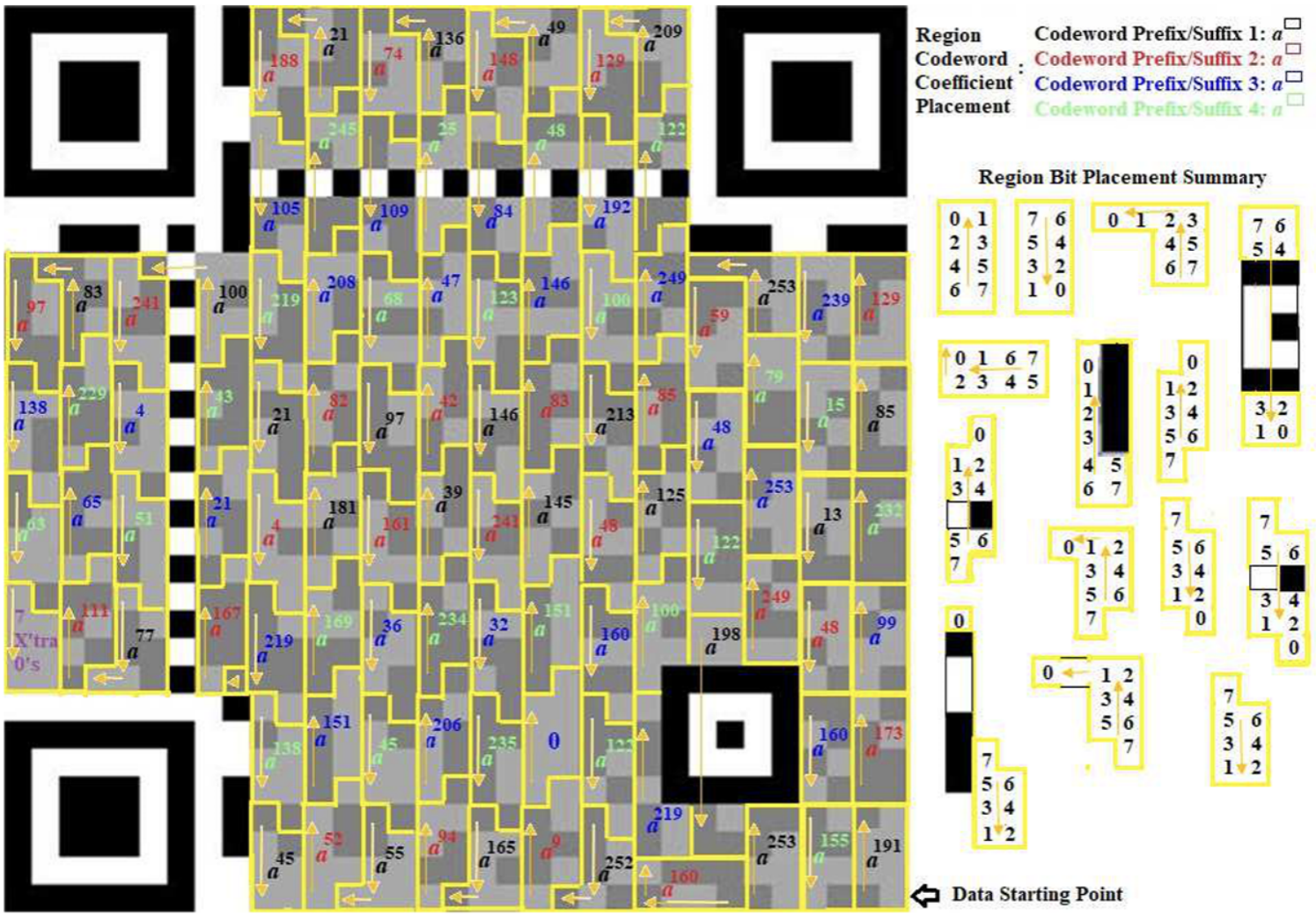
There are various versions/sizes of QR codes — we will look at a fairly simple one, version 4



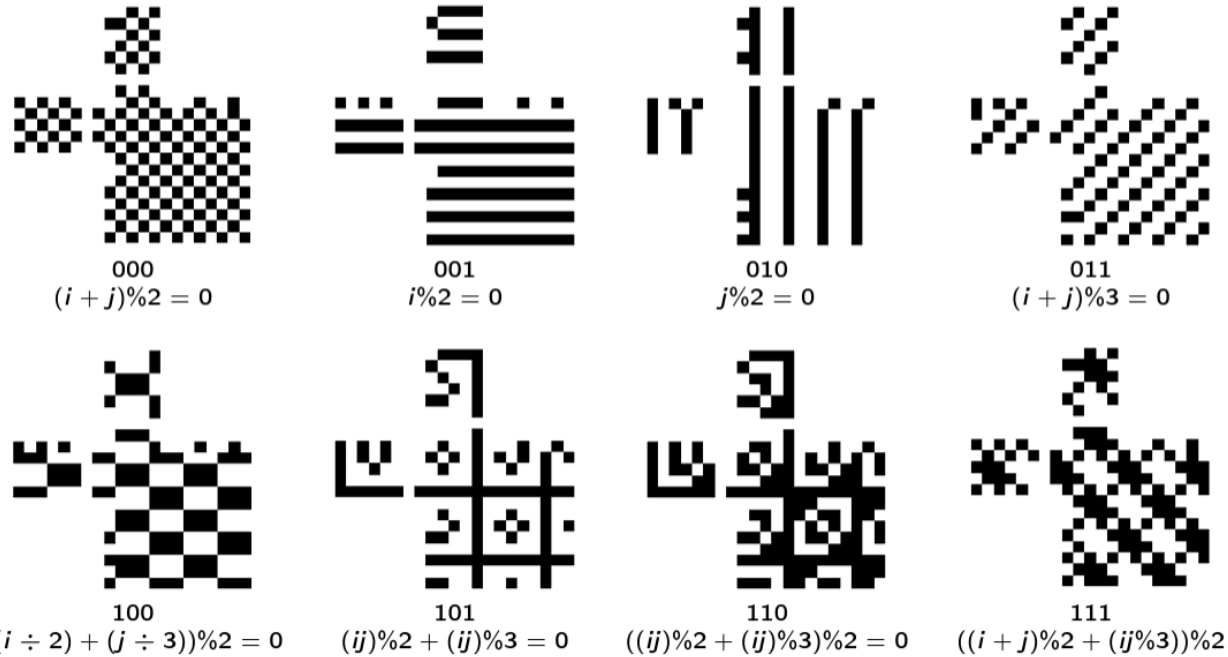
Design explained

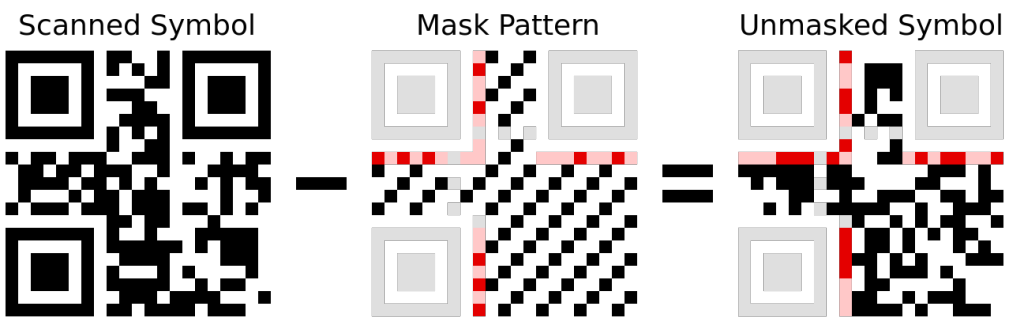
- The data is in 4×2 blocks starting from the bottom right: it goes up and then back down, snaking throughout the matrix
- Bits are in a zigzag order within each block, and encoded with a Reed–Solomon code (of course!)
- Various different finder patterns are used to let scanners detect and align the grid: large finder patterns in 3 corners, an alignment pattern in the bottom right, and a row and column of timing patterns
- Format info is included twice in specific areas, to ensure it can be read — it includes the error correction level and which *masking pattern* is used
- A masking pattern is overlaid onto the data matrix to make the black and white squares more evenly distributed and readable, by inverting certain regions

Data in a QR code



QR code masking patterns





Concluding remarks

Key takeaways:

- Reed–Solomon codes are a crucial application of abstract algebra to a problem encountered everywhere
- Many parts of daily life rely on brilliant algorithms we often don't think about
- Topics like Galois fields are fascinating on their own but can also be utilized in unexpected places

References/further reading

- http://simoneparisotto.com/math/misc/qrcode/qrcode.pdf
- https://atcm.mathandtech.org/EP2021/invited/21891.pdf
- http://user.xmission.com/~rimrock/Documents/Galois%20Fields%20and%20Cyclic%20Codes.pdf
- https://web.ntpu.edu.tw/~yshan/BCH_code.pdf
- https://pages.jh.edu/bcooper8/sigma_files/ERROR_CONTROL_CODING/06dec.pdf

Thank you for listening!