Reed–Solomon error correction

From Wikipedia, the free encyclopedia

In coding theory, **Reed–Solomon (RS) codes** are non-binary[1] cyclic error-correcting codes invented by Irving S. Reed and Gustave Solomon. They described a systematic way of building codes that could detect and correct multiple random symbol errors. By adding t check symbols to the data, an RS code can detect any combination of up to t erroneous symbols, and correct up to $\lfloor t/2 \rfloor$ symbols. As an erasure code, it can correct up to t known erasures, or it can detect and correct combinations of errors and erasures. Furthermore, RS codes are suitable as multiple-burst bit-error correcting codes, since a sequence of $b + 1$ consecutive bit errors can affect at most two symbols of size b.[2] The choice of t is up to the designer of the code, and may be selected within wide limits.

In Reed–Solomon coding, source symbols are viewed as coefficients of a polynomial $p(x)$ over a finite field. The original idea was to create n code symbols from k source symbols by oversampling $p(x)$ at $n > k$ distinct points, transmit the sampled points, and use interpolation techniques at the receiver to recover the original message. That is not how RS codes are used today. Instead, RS codes are viewed as cyclic BCH codes, where encoding symbols are derived from the coefficients of a polynomial constructed by multiplying $p(x)$ with a cyclic generator polynomial. This gives rise to efficient decoding algorithms (described below).

Reed–Solomon codes have since found important applications from deep-space communication to consumer electronics. They are prominently used in consumer electronics such as CDs, DVDs, Blu-ray Discs, in data transmission technologies such as DSL and WiMAX, in broadcast systems such as DVB and ATSC, and in computer applications such as RAID 6 systems.

Reed–Solomon codes

Named after	Irving S. Reed and Gustave Solomon
Classification	
Hierarchy	Linear block code Polynomial code Cyclic code BCH code Reed–Solomon code
Parameters	
Block length	$n = q - 1$
Message length	k
Distance	$n - k + 1$
Alphabet size	$q = p^m$ (p prime)
Notation	$[n, k, n - k + 1]_q$-code
Algorithms	
Decoding	Berlekamp–Massey Euclidean *et al.*
Properties	
Maximum-distance separable code	

Contents

History

Reed–Solomon codes were developed in 1960 by Irving S. Reed and Gustave Solomon, who were then staff members of MIT Lincoln Laboratory. Their seminal article was entitled "Polynomial Codes over Certain Finite Fields." (Reed & Solomon 1960) When the article was

written, an efficient decoding algorithm was not known. A solution for the latter was found in 1969 by Elwyn Berlekamp and James Massey, and is since known as the Berlekamp–Massey decoding algorithm. In 1977, RS codes were notably implemented in the Voyager program in the form of concatenated codes. The first commercial application in mass-produced consumer products appeared in 1982 with the compact disc, where two interleaved RS codes are used. Today, RS codes are widely implemented in digital storage devices and digital communication standards, though they are being slowly replaced by more modern low-density parity-check (LDPC) codes or turbo codes. For example, RS codes are used in the digital video broadcasting (DVB) standard DVB-S, but LDPC codes are used in its successor DVB-S2.

Description

Original view (transmitting points)

The original concept of Reed–Solomon coding (Reed & Solomon 1960) describes encoding of k message symbols by viewing them as coefficients of a polynomial $p(x)$ of maximum degree $k - 1$ over a finite field of order N, and evaluating the polynomial at $n > k$ distinct input points. Sampling a polynomial of degree $k - 1$ at more than k points creates an overdetermined system, and allows recovery of the polynomial at the receiver given any k out of n sample points using (Lagrange) interpolation. The sequence of distinct points is created by a generator of the finite field's multiplicative group, and includes 0, thus permitting any value of n up to N.

Using a mathematical formulation, let $(x_1, x_2, ..., x_n)$ be the input sequence of n distinct values over the finite field F; then the codebook $\mathbf{C}$ created from the tuplets of values obtained by evaluating every polynomial (over F) of degree less than k at each x_i is

$$\mathbf{C} = \{(f(x_1), f(x_2), \ldots, f(x_n)) \,|\, f \in F[x], \deg(f) < k\},$$

where $F[x]$ is the polynomial ring over F, and k and n are chosen such that $1 \le k \le n \le N$.

As described above, an input sequence $(x_1, x_2, ..., x_n)$ of $n = N$ values is created as

$$(0, \alpha^0, \alpha^1, \ldots, \alpha^{N-2}),$$

where α is a primitive root of F. When omitting 0 from the sequence, and since $\alpha^{N-1} = 1$, it follows that for every polynomial $p(x)$ the function $p(\alpha x)$ is also a polynomial of the same degree, and its codeword is a cyclic left-shift of the codeword derived from $p(x)$; thus, a Reed–Solomon code can be viewed as a cyclic code. This is pursued in the classic view of RS codes, described subsequently.

As outlined in the section on a theoretical decoder, the original view does not give rise to an efficient decoding algorithm, even though it shows that such a code can work.

Classic view (Reed–Solomon codes as BCH codes)

In practice, instead of sending sample values of a polynomial, the encoding symbols are viewed as the coefficients of an output polynomial $s(x)$ constructed by multiplying the message polynomial $p(x)$ of maximum degree $k-1$ by a generator polynomial $g(x)$ of degree $t = N - k - 1$. The generator polynomial $g(x)$ is defined by having $\alpha, \alpha^2, ..., \alpha^t$ as its roots, i.e.,

$$g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^t) = g_0 + g_1 x + \cdots + g_{t-1}x^{t-1} + x^t.$$

The transmitter sends the $N-1$ coefficients of $s(x) = p(x)g(x)$, and the receiver can use polynomial division by $g(x)$ of the received polynomial to determine whether the message is in error; a non-zero remainder means that an error was detected.[3] Let $r(x)$ be the non-zero remainder polynomial, then the receiver can evaluate $r(x)$ at the roots of $g(x)$, and build a system of equations that eliminates $s(x)$ and identifies which coefficients of $r(x)$ are in error, and the magnitude of each coefficient's error. (Berlekamp 1984) (Massey 1969) If the system of equations can be solved, then the receiver knows how to modify his $r(x)$ to get the most likely $s(x)$.

Reed–Solomon codes are a special case of a larger class of codes called BCH codes. The Berlekamp–Massey algorithm has been designed for the decoding of such codes, and is thus applicable to Reed–Solomon codes.

To see that Reed–Solomon codes are special BCH codes, it is useful to give the following alternative definition of Reed–Solomon codes.[4]

Given a finite field F of size q, let $n = q - 1$ and let α be a primitive nth root of unity in F. Also let $1 \leq k \leq n$ be given. The *Reed–Solomon code* for these parameters has code word $(f_0, f_1, \ldots, f_{n-1})$ if and only if $\alpha, \alpha^2, \ldots, \alpha^{n-k}$ are roots of the polynomial

$$p(x) = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1},$$

With this definition, it is immediately seen that a Reed–Solomon code is a polynomial code, and in particular a BCH code. The generator polynomial $g(x)$ is the minimal polynomial with roots $\alpha, \alpha^2, \ldots, \alpha^{n-k}$ as defined above, and the code words are exactly the polynomials that are divisible by $g(x)$.

Equivalence of the two formulations

At first sight, the above two definitions of Reed–Solomon codes seem very different. In the first definition, code words are *values* of polynomials, whereas in the second, they are *coefficients*. Moreover, the polynomials in the first definition are required to be of small degree, whereas those in the second definition are required to have specific roots.

The equivalence of the two definitions is proved using the discrete Fourier transform. This transform, which exists in all finite fields as well as

the complex numbers, establishes a duality between the coefficients of polynomials and their values. This duality can be approximately summarized as follows: Let $p(x)$ and $q(x)$ be two polynomials of degree less than n. If the *values* of $p(x)$ are the *coefficients* of $q(x)$, then (up to a scalar factor and reordering), the *values* of $q(x)$ are the *coefficients* of $p(x)$. For this to make sense, the values must be taken at locations $x = \alpha^i$, for $i = 0, \ldots, n-1$, where α is a primitive nth root of unity.

To be more precise, let

$$p(x) = v_0 + v_1 x + v_2 x^2 + \cdots + v_{n-1} x^{n-1},$$
$$q(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{n-1} x^{n-1}$$

and assume $p(x)$ and $q(x)$ are related by the discrete Fourier transform. Then the coefficients and values of $p(x)$ and $q(x)$ are related as follows: for all $i = 0, \ldots, n-1$, $f_i = p(\alpha^i)$ and $v_i = \frac{1}{n} q(\alpha^{n-i})$.

Using these facts, we have: $(f_0, \ldots, f_{n-1})$ is a code word of the Reed–Solomon code according to the first definition

- if and only if $p(x)$ is of degree less than k (because $f_0, \ldots, f_{n-1}$ are the values of $p(x)$),
- if and only if $v_i = 0$ for $i = k, \ldots, n-1$,
- if and only if $q(\alpha^i) = 0$ for $i = 1, \ldots, n-k$ (because $q(\alpha^i) = n v_{n-i}$),
- if and only if $(f_0, \ldots, f_{n-1})$ is a code word of the Reed–Solomon code according to the second definition.

This shows that the two definitions are equivalent.

Remarks

Reed–Solomon codes are usually constructed as systematic codes. Instead of sending $s(x) = p(x)g(x)$, the encoder will construct the transmitted polynomial $s(x)$ such that it is evenly divisible by $g(x)$ and $p(x)$ is apparent in the codeword. Ordinarily, the construction is done by multiplying $p(x)$ by x^t to make room for the t check symbols, dividing that product by $g(x)$ to find the remainder, and then compensating for that remainder. In this case, the t check symbols are created by computing the remainder, $s_r(x)$:

$$s_r(x) = (p(x) \times x^t) \mod g(x)$$

and that remainder is used to make an evenly divisible codeword:

$$s(x) = p(x) \times x^t - s_r(x)$$

with the result

$$s(x) \bmod g(x) = (p(x) \times x^t - s_r(x)) \bmod g(x) = s_r(x) - s_r(x) = 0$$

showing that $s(x)$ is a multiple of the generator polynomial $g(x)$.[5]

Designers are not required to use the "natural" sizes of Reed–Solomon code blocks. A technique known as "shortening" can produce a smaller code of any desired size from a larger code. For example, the widely used (255,223) code can be converted to a (160,128) code by padding the unused portion of the source block with 95 binary zeroes and not transmitting them. At the decoder, the same portion of the block is loaded locally with binary zeroes. The Delsarte-Goethals-Seidel[6] theorem illustrates an example of an application of shortened Reed–Solomon codes. In parallel to shortening, a technique known as puncturing allows omitting some of the encoded parity symbols.

Properties

The Reed–Solomon code is a [n, k, $n - k + 1$] code; in other words, it is a linear block code of length n (over F) with dimension k and minimum Hamming distance $n - k + 1$. The Reed–Solomon code is optimal in the sense that the minimum distance has the maximum value possible for a linear code of size (n, k); this is known as the Singleton bound. Such a code is also called a maximum distance separable (MDS) code.

The error-correcting ability of a Reed–Solomon code is determined by its minimum distance, or equivalently, by $n - k$, the measure of redundancy in the block. If the locations of the error symbols are not known in advance, then a Reed–Solomon code can correct up to $(n - k)/2$ erroneous symbols, i.e., it can correct half as many errors as there are redundant symbols added to the block. Sometimes error locations are known in advance (e.g., "side information" in demodulator signal-to-noise ratios)—these are called erasures. A Reed–Solomon code (like any MDS code) is able to correct twice as many erasures as errors, and any combination of errors and erasures can be corrected as long as the relation $2E + S \leq n - k$ is satisfied, where E is the number of errors and S is the number of erasures in the block.

For practical uses of Reed–Solomon codes, it is common to use a finite field F with 2^m elements. In this case, each symbol can be represented as an m-bit value. The sender sends the data points as encoded blocks, and the number of symbols in the encoded block is $n = 2^m - 1$. Thus a Reed–Solomon code operating on 8-bit symbols has $n = 2^8 - 1 = 255$ symbols per block. (This is a very popular value because of the prevalence of byte-oriented computer systems.) The number k, with $k < n$, of *data* symbols in the block is a design parameter. A commonly used code encodes $k = 223$ eight-bit data symbols plus 32 eight-bit parity symbols in an $n = 255$-symbol block; this is denoted as a $(n, k) = (255, 223)$ code, and is capable of correcting up to 16 symbol errors per block.

The above properties of Reed–Solomon codes make them especially well-suited to applications where errors occur in bursts. This is because it does not matter to the code how many bits in a symbol are in error — if multiple bits in a symbol are corrupted it only counts as a single error.

Conversely, if a data stream is not characterized by error bursts or drop-outs but by random single bit errors, a Reed–Solomon code is usually a poor choice compared to a binary code.

The Reed–Solomon code, like the convolutional code, is a transparent code. This means that if the channel symbols have been inverted somewhere along the line, the decoders will still operate. The result will be the inversion of the original data. However, the Reed–Solomon code loses its transparency when the code is shortened. The "missing" bits in a shortened code need to be filled by either zeros or ones, depending on whether the data is complemented or not. (To put it another way, if the symbols are inverted, then the zero-fill needs to be inverted to a one-fill.) For this reason it is mandatory that the sense of the data (i.e., true or complemented) be resolved before Reed–Solomon decoding.

Error correction algorithms

Theoretical decoder

Reed & Solomon (1960) described a theoretical decoder that corrected errors by finding the most popular message polynomial. The decoder for a RS (n, k) code would look at all possible subsets of k symbols from the set of n symbols that were received. For the code to be correctable in general, at least k symbols had to be received correctly, and k symbols are needed to interpolate the message polynomial. The decoder would interpolate a message polynomial for each subset, and it would keep track of the resulting polynomial candidates. The most popular message is the corrected result. Unfortunately, there are a lot of subsets, so the algorithm is impractical. The number of subsets is the binomial coefficient, $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, and the number of subsets is infeasible for even modest codes. For a $(255, 249)$ code that can correct 3 errors, the naive theoretical decoder would examine 359 billion subsets. The RS code needed a practical decoder.

Peterson decoder

Main article: Peterson–Gorenstein–Zierler algorithm

Peterson (1960) developed a practical decoder based on syndrome decoding. (Welch 1997, p. 10) Berlekamp (below) would improve on that decoder.

Syndrome decoding

The transmitted message is viewed as the coefficients of a polynomial *s*(*x*) that is divisible by a generator polynomial *g*(*x*). Welch (1997, p. 5)

$$s(x) = \sum_{i=0}^{n-1} c_i x^i$$
$$g(x) = \prod_{j=1}^{n-k} (x - \alpha^j),$$

where α is a primitive root.

Since $s(x)$ is divisible by generator $g(x)$, it follows that

$$s(\alpha^i) = 0,\ i = 1, 2, \ldots, n-k$$

The transmitted polynomial is corrupted in transit by an error polynomial $e(x)$ to produce the received polynomial $r(x)$.

$$r(x) = s(x) + e(x)$$
$$e(x) = \sum_{i=0}^{n-1} e_i x^i$$

where e_i is the coefficient for the i-th power of x. Coefficient e_i will be zero if there is no error at that power of x and nonzero if there is an error. If there are v errors at distinct powers i_k of x, then

$$e(x) = \sum_{k=1}^{\nu} e_{i_k} x^{i_k}$$

The goal of the decoder is to find the number of errors (v), the positions of the errors (i_k), and the error values at those positions (e_{i_k}).

The syndromes S_j are defined as

$$S_j = r(\alpha^j) = s(\alpha^j) + e(\alpha^j) = 0 + e(\alpha^j) = e(\alpha^j),\ j = 1, 2, \ldots, n-k$$
$$= \sum_{k=1}^{\nu} e_{i_k} \left(\alpha^j\right)^{i_k}$$

The advantage of looking at the syndromes is that the message polynomial drops out.

Error locators and error values

For convenience, define the **error locators** X_k and **error values** Y_k as:

$$X_k = \alpha^{i_k},\ Y_k = e_{i_k}$$

Then the syndromes can be written in terms of the error locators and error values as

$$S_j = \sum_{k=1}^{\nu} Y_k X_k^j$$

The syndromes give a system of $n - k \geq 2v$ equations in $2v$ unknowns, but that system of equations is nonlinear in the X_k and does not have an obvious solution. However, if the X_k were known (see below), then the syndrome equations provide a linear system of equations that can easily be solved for the Y_k error values.

$$\begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_\nu^1 \\ X_1^2 & X_2^2 & \cdots & X_\nu^2 \\ \vdots & \vdots & & \vdots \\ X_1^{n-k} & X_2^{n-k} & \cdots & X_\nu^{n-k} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_\nu \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{n-k} \end{bmatrix}$$

Consequently, the problem is finding the X_k.

Error locator polynomial

Peterson found a linear recurrence relation that gave rise to a system of linear equations. (Welch 1997, p. 10) Solving those equations identifies the error locations.

Define the **error locator polynomial** $\Lambda(x)$ as

$$\Lambda(x) = \prod_{k=1}^{\nu} (1 - xX_k) = 1 + \Lambda_1 x^1 + \Lambda_2 x^2 + \cdots + \Lambda_\nu x^\nu$$

The zeros of $\Lambda(x)$ are the reciprocals X_k^{-1}:

$$\Lambda(X_k^{-1}) = 0$$

$$\Lambda(X_k^{-1}) = 1 + \Lambda_1 X_k^{-1} + \Lambda_2 X_k^{-2} + \cdots + \Lambda_\nu X_k^{-\nu} = 0$$

Multiply both sides by $Y_k X_k^{j+\nu}$ and it will still be zero.

$$Y_k X_k^{j+\nu} \Lambda(X_k^{-1}) = 0.$$

$$\text{Hence } Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu} X_k^{-1} + \Lambda_2 Y_k X_k^{j+\nu} X_k^{-2} + \cdots + \Lambda_\nu Y_k X_k^{j+\nu} X_k^{-\nu} = 0,$$

$$\text{and so } Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu-1} + \Lambda_2 Y_k X_k^{j+\nu-2} + \cdots + \Lambda_\nu Y_k X_k^{j} = 0$$

Sum for $k = 1$ to ν

$$\sum_{k=1}^{\nu} (Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu-1} + \Lambda_2 Y_k X_k^{j+\nu-2} + \cdots + \Lambda_\nu Y_k X_k^{j}) = 0$$

$$\sum_{k=1}^{\nu} (Y_k X_k^{j+\nu}) + \Lambda_1 \sum_{k=1}^{\nu} (Y_k X_k^{j+\nu-1}) + \Lambda_2 \sum_{k=1}^{\nu} (Y_k X_k^{j+\nu-2}) + \cdots + \Lambda_\nu \sum_{k=1}^{\nu} (Y_k X_k^{j}) = 0$$

This reduces to

$$S_{j+\nu} + \Lambda_1 S_{j+\nu-1} + \cdots + \Lambda_{\nu-1} S_{j+1} + \Lambda_\nu S_j = 0$$

$$S_j \Lambda_\nu + S_{j+1} \Lambda_{\nu-1} + \cdots + S_{j+\nu-1} \Lambda_1 = -S_{j+\nu}$$

This yields a system of linear equations that can be solved for the coefficients Λ_i of the error location polynomial:

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_\nu \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & \vdots & & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{\nu+\nu} \end{bmatrix}$$

Obtain the error locations from the error locator polynomial

Use the coefficients Λ_i found in the last step to build the error location polynomial. The roots of the error location polynomial can be found by exhaustive search. The error locators (and hence the error locations) can be found from those roots. Chien search is an efficient implementation of this step.

Calculate the error values

Once the error locations are known, the error values can be determined and corrected. This can be done by direct solution for Y_k in the error equations given above, or using the Forney algorithm.

Berlekamp–Massey decoder

The Berlekamp–Massey algorithm is an alternate iterative procedure for finding the error locator polynomial. During each iteration, it calculates a discrepancy based on a current instance of Λ(x) with an assumed number of errors *e*:

$$\Delta = S_i + \Lambda_1\ S_{i-1} + \cdots + \Lambda_e\ S_{i-e}$$

and then adjusts Λ(x) and *e* so that a recalculated Δ would be zero. Berlekamp–Massey algorithm has a detailed description of the procedure. In the following example, C(x) is used to represent Λ(x).

Example

Consider the Reed–Solomon code defined in $GF(929)$ with $\alpha = 3$ and $t = 4$ (this is used in PDF417 barcodes). The generator polynomial is

$$g(x) = (x-3)(x-3^2)(x-3^3)(x-3^4) = x^4 + 809x^3 + 723x^2 + 568x + 522$$

If the message polynomial is $p(x) = 3\ x^2 + 2\ x + 1$, then the codeword is calculated as follows.

$$s_r(x) = p(x)\,x^t \mod g(x) = 547x^3 + 738x^2 + 442x + 455$$
$$s(x) = p(x)\,x^t - s_r(x) = 3x^6 + 2x^5 + 1x^4 + 382x^3 + 191x^2 + 487x + 474$$

Errors in transmission might cause this to be received instead.

$$r(x) = s(x) + e(x) = 3x^6 + 2x^5 + 123x^4 + 456x^3 + 191x^2 + 487x + 474$$

The syndromes are calculated by evaluating r at powers of α.

$$S_1 = r(3^1) = 3 \cdot 3^6 + 2 \cdot 3^5 + 123 \cdot 3^4 + 456 \cdot 3^3 + 191 \cdot 3^2 + 487 \cdot 3 + 474 = 732$$
$$S_2 = r(3^2) = 637,\ S_3 = r(3^3) = 762,\ S_4 = r(3^4) = 925$$

To correct the errors, first use the Berlekamp–Massey algorithm to calculate the error locator polynomial.

n	S_{n+1}	d	C	B	b	m
0	732	732	$197\,x + 1$	1	732	1
1	637	846	$173\,x + 1$	1	732	2
2	762	412	$634\,x^2 + 173\,x + 1$	$173\,x + 1$	412	1
3	925	576	$329\,x^2 + 821\,x + 1$	$173\,x + 1$	412	2

The final value of C is the error locator polynomial, $\Lambda(x)$. The zeros can be found by trial substitution. They are $x_1 = 757 = 3^{-3}$ and $x_2 = 562 = 3^{-4}$, corresponding to the error locations. To calculate the error values, apply the Forney algorithm.

$$\Omega(x) = S(x)\Lambda(x) \mod x^4 = 546x + 732$$
$$\Lambda'(x) = 658x + 821$$
$$e_1 = -\Omega(x_1)/\Lambda'(x_1) = -649/54 = 280 \times 843 = 74$$
$$e_2 = -\Omega(x_2)/\Lambda'(x_2) = 122$$

Subtracting e_1x^3 and e_2x^4 from the received polynomial r reproduces the original codeword s.

Euclidean decoder

Another iterative method for calculating the error locator polynomial is based on the Euclidean algorithm

t = number of parities
$R_0 = x^t$
S_0 = syndrome polynomial
$A_0 = 1$

$B_0 = 0$
$i = 0$
while degree of $S_i \geq (t/2)$

$Q = R_i / S_i$
$S_{i+1} = R_i - \mathrm{Q}\ S_i = R_i$ modulo S_i
$A_{i+1} = \mathrm{Q}\ A_i + B_i$
$R_{i+1} = S_i$
$B_{i+1} = A_i$
$i = i + 1$

$\Lambda(x) = A_i / A_i(0)$
$\Omega(x) = (-1)^{\deg A_i} S_i / A_i(0)$

$A_i(0)$ is the constant (least significant) term of A_i.

Here is an example of the Euclidean method, using the same data as the Berlekamp Massey example above. In the table below, R and S are forward, A and B are reversed.

i	$\mathbf{R}_i$	$\mathbf{A}_i$	$\mathbf{S}_i$	$\mathbf{B}_i$
0	$001\ x^4 + 000\ x^3 + 000\ x^2 + 000\ x + 000$	001	$925\ x^3 + 762\ x^2 + 637\ x + 732$	000
1	$925\ x^3 + 762\ x^2 + 637\ x + 732$	$533 + 232\ x$	$683x^2 + 676\ x + 024$	001
2	$683\ x^2 + 676\ x + 024$	$544 + 704\ x + 608\ x^2$	$673\ x + 596$	$533 + 232\ x$

$\Lambda(x) = A_2 / 544 = 001 + 821\ \mathrm{x} + 329\ \mathrm{x}^2$
$\Omega(x) = (-1)^2\ S_2 / 544 = 546\ \mathrm{x} + 732$

Decoding in frequency domain (sketch)

The above algorithms are presented in the time domain. Decoding in the frequency domain, using Fourier transform techniques, can offer computational and implementation advantages. (Hong & Vetterli 1995)

The following is a sketch of the main idea behind this error correction technique.

By definition, a code word of a Reed–Solomon code is given by the sequence of values of a low-degree polynomial over a finite field. A key fact for the error correction algorithm is that the *values* and the *coefficients* of a polynomial are related by the discrete Fourier transform.

The purpose of a Fourier transform is to convert a signal from a time domain to a frequency domain or vice versa. In case of the Fourier transform over a finite field, the frequency domain signal corresponds to the coefficients of a polynomial, and the time domain signal correspond to the values of the same polynomial.

As shown in Figures 1 and 2, an isolated value in the frequency domain corresponds to a smooth wave in the time domain. The wavelength depends on the location of the isolated value.

Conversely, as shown in Figures 3 and 4, an isolated value in the time domain corresponds to a smooth wave in the frequency domain.

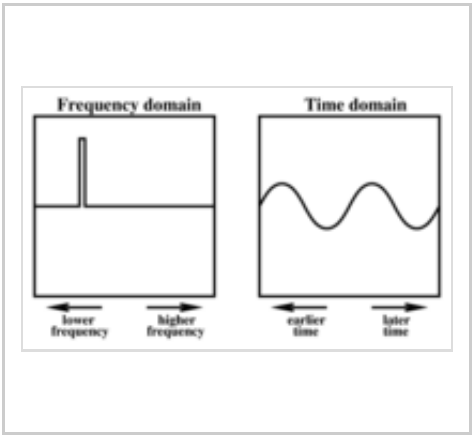

Figure 1

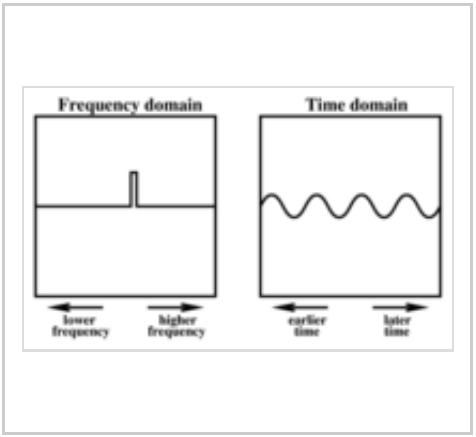

Figure 2

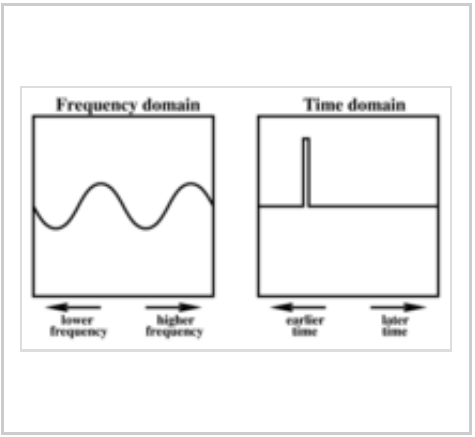

Figure 3

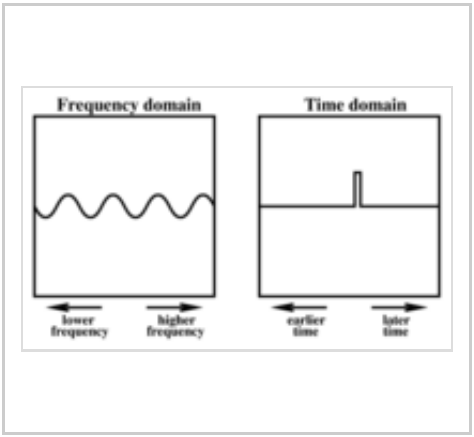

Figure 4

In a Reed–Solomon code, the frequency domain is divided into two regions as shown in Figure 5: a left (low-frequency) region of length k, and a right (high-frequency) region of length $n - k$. A data word is then embedded into the left region (corresponding to the k coefficients of a polynomial of degree at most $k - 1$), while the right region is filled with zeros. The result is Fourier transformed into the time domain, yielding a code word that is composed only of low frequencies. In the absence of errors, a code word can be decoded by reverse Fourier transforming it back into the frequency domain.

Now consider a code word containing a single error, as shown in red in Figure 6. The effect of this error in the frequency domain is a smooth, single-frequency wave in the right region, called the *syndrome* of the error. The error location can be determined by determining the frequency of the syndrome signal.

Similarly, if two or more errors are introduced in the code word, the syndrome will be a signal composed of two or more frequencies, as shown in Figure 7. As long as it is possible to determine the frequencies of which the syndrome is composed, it is possible to determine the error locations. Notice that the error *locations* depend only on the *frequencies* of these waves, whereas the error *magnitudes* depend on their amplitudes and phase.

The problem of determining the error locations has therefore been reduced to the problem of finding, given a sequence of $n - k$ values, the smallest set of elementary waves into which these values can be decomposed. It is known from digital signal processing that this problem is equivalent to finding the roots of the minimal polynomial of the sequence, or equivalently, of finding the shortest linear feedback shift register (LFSR) for the sequence. The latter problem can either be solved inefficiently by solving a system of linear equations, or more efficiently by the Berlekamp–Massey algorithm.

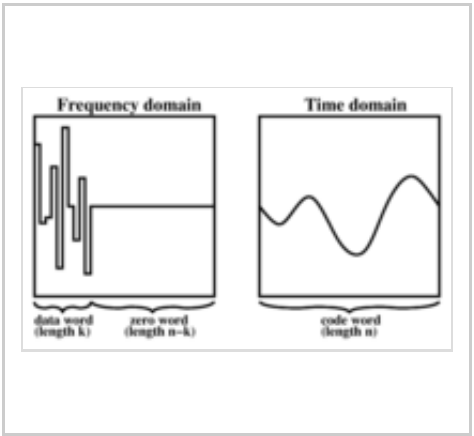

Figure 5

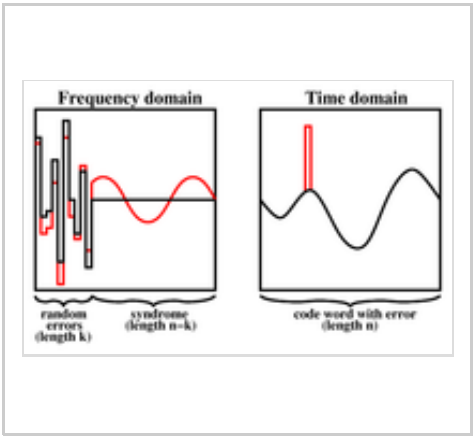

Figure 6

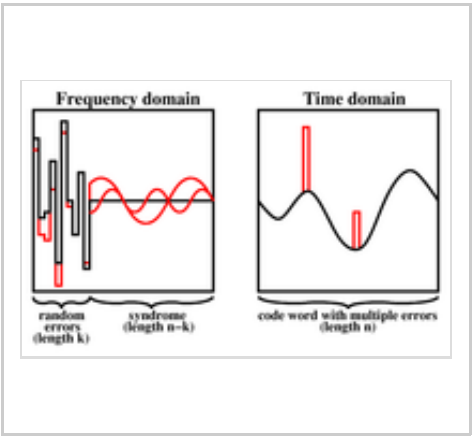

Figure 7

Decoding beyond the error-correction bound

The Singleton bound states that the minimum distance d of a linear block code of size (n,k) is upper-bounded by $n - k + 1$. The distance d was usually understood to limit the error-correction capability to $\lfloor d/2 \rfloor$. The Reed–Solomon code achieves this bound with equality, and can thus correct up to $\lfloor (n - k + 1)/2 \rfloor$ errors. However, this error-correction bound is not exact.

In 1999, Madhu Sudan and Venkatesan Guruswami at MIT published "Improved Decoding of Reed–Solomon and Algebraic-Geometry Codes" introducing an algorithm that allowed for the correction of errors beyond half the minimum distance of the code. It applies to Reed–Solomon codes and more generally to algebraic geometric codes. This algorithm produces a list of codewords (it is a list-decoding algorithm) and is based on interpolation and factorization of polynomials over $GF(2^m)$ and its extensions.

Soft-decoding

The algebraic decoding methods described above are hard-decision methods, which means that for every symbol a hard decision is made about its value.[7] The advent of LDPC and turbo codes, which employ iterated soft-decision belief propagation decoding methods to achieve error-correction performance close to the theoretical limit, has spurred interest in applying soft-decision decoding to conventional algebraic codes. In 2003, Ralf Koetter and Alexander Vardy presented a polynomial-time soft-decision algebraic list-decoding algorithm for RS codes, which was based upon the work by Sudan and Guruswami.[8]

Applications

Data storage

Reed–Solomon coding is very widely used in mass storage systems to correct the burst errors associated with media defects.

Reed–Solomon coding is a key component of the compact disc. It was the first use of strong error correction coding in a mass-produced consumer product, and DAT and DVD use similar schemes. In the CD, two layers of Reed–Solomon coding separated by a 28-way convolutional interleaver yields a scheme called Cross-Interleaved Reed Solomon Coding (CIRC). The first element of a CIRC decoder is a relatively weak inner (32,28) Reed–Solomon code, shortened from a (255,251) code with 8-bit symbols. This code can correct up to 2 byte errors per 32-byte block. More importantly, it flags as erasures any uncorrectable blocks, i.e., blocks with more than 2 byte errors. The decoded 28-byte blocks, with erasure indications, are then spread by the deinterleaver to different blocks of the (28,24) outer code. Thanks to the deinterleaving, an erased 28-byte block from the inner code becomes a single erased byte in each of 28 outer code blocks. The outer code easily corrects this, since it can handle up to 4 such erasures per block.

The result is a CIRC that can completely correct error bursts up to 4000 bits, or about 2.5 mm on the disc surface. This code is so strong that

most CD playback errors are almost certainly caused by tracking errors that cause the laser to jump track, not by uncorrectable error bursts.[9]

DVDs uses a similar scheme, but with much larger blocks, a (208,192) inner code, and a (182,172) outer code.

Reed–Solomon error correction is also used in parchive files which are commonly posted accompanying multimedia files on USENET. The Distributed online storage service Wuala also makes use of Reed–Solomon when breaking up files.

Bar code

Almost all two-dimensional bar codes such as PDF-417, MaxiCode, Datamatrix, QR Code, and Aztec Code use Reed–Solomon error correction to allow correct reading even if a portion of the bar code is damaged. When the bar code scanner cannot recognize a bar code symbol, it will treat it as an erasure.

Reed-Solomon coding is less common in one-dimensional bar codes, but is used by the PostBar symbology.

Data transmission

Specialized forms of Reed–Solomon codes, specifically Cauchy-RS and Vandermonde-RS, can be used to overcome the unreliable nature of data transmission over erasure channels. The encoding process assumes a code of RS(N, K) which results in N codewords of length N symbols each storing K symbols of data, being generated, that are then sent over an erasure channel.

Any combination of K codewords received at the other end is enough to reconstruct all of the N codewords. The code rate is generally set to 1/2 unless the channel's erasure likelihood can be adequately modelled and is seen to be less. In conclusion, N is usually $2K$, meaning that at least half of all the codewords sent must be received in order to reconstruct all of the codewords sent.

Reed–Solomon codes are also used in xDSL systems and CCSDS's Space Communications Protocol Specifications as a form of forward error correction.

Satellite transmission

One significant application of Reed–Solomon coding was to encode the digital pictures sent back by the Voyager space probe.

Voyager introduced Reed–Solomon coding concatenated with convolutional codes, a practice that has since become very widespread in deep space and satellite (e.g., direct digital broadcasting) communications.

Viterbi decoders tend to produce errors in short bursts. Correcting these burst errors is a job best done by short or simplified Reed–Solomon

codes.

Modern versions of concatenated Reed–Solomon/Viterbi-decoded convolutional coding were and are used on the Mars Pathfinder, Galileo, Mars Exploration Rover and Cassini missions, where they perform within about 1–1.5 dB of the ultimate limit imposed by the Shannon capacity.

These concatenated codes are now being replaced by more powerful turbo codes where the transmitted data does not need to be decoded immediately.

See also

- BCH code
- Cyclic code
- Chien search
- Berlekamp–Massey algorithm
- Forward error correction
- Berlekamp–Welch algorithm
- Folded Reed–Solomon code

Notes

1. ^ Codes for which each input symbol is from a set of size greater than 2.
2. ^ A popular construction is a concatenation of an outer RS code with an inner convolutional code, since the latter delivers errors primarily in bursts.
3. ^ There is a slight but usually negligible chance, depending on channel properties, that channel errors turn the message into another valid polynomial.
4. ^ Lidl, Rudolf; Pilz, Günter (1999). *Applied Abstract Algebra* (2nd ed.). Wiley. p. 226.
5. ^ See Lin & Costello (1983, p. 171), for example.
6. ^ "Kissing Numbers, Sphere Packings, and Some Unexpected Proofs", Notices of the American Mathematical Society, Volume 51, Issue 8, 2004/09. Explains the Delsarte-Goethals-Seidel (http://www.ams.org/notices/200408/fea-pfender.pdf) theorem as used in the context of the error correcting code for compact disc.
7. ^ For example, a decoder could associate with each symbol an additional value corresponding to the channel demodulator's confidence in the correctness of the symbol.
8. ^ Koetter, Ralf; Vardy, Alexander (2003). "Algebraic soft-decision decoding of Reed–Solomon codes". *IEEE Transactions on Information Theory* **49** (11): 2809–2825. DOI:10.1109/TIT.2003.819332 (http://dx.doi.org/10.1109%2FTIT.2003.819332) .
9. ^ K.A.S. Immink, *Reed–Solomon Codes and the Compact Disc* in S.B. Wicker and V.K. Bhargava, Edrs, *Reed–Solomon Codes and Their Applications*, IEEE Press, 1994.

References

- Cipra, Barry A. (1993), "The Ubiquitous Reed–Solomon Codes" (http://www.eccpage.com/reed_solomon_codes.html) , *SIAM News* **26** (1), http://www.eccpage.com/reed_solomon_codes.html
- Berlekamp, Elwyn R. (1967), *Nonbinary BCH decoding*, International Symposium on Information Theory, San Remo, Italy
- Berlekamp, Elwyn R. (1984) [1968], *Algebraic Coding Theory*, Laguna Hills, CA: Aegean Park Press, ISBN 0-89412-063-8
- Forney, Jr., G. (October 1965), "On Decoding BCH Codes", *IEEE Transactions on Information Theory* **11** (4): 549–557, DOI:10.1109/TIT.1965.1053825 (http://dx.doi.org/10.1109%2FTIT.1965.1053825)
- Gill, John (unknown), *EE387 Notes #7, Handout #28* (http://www.stanford.edu/class/ee387/handouts/notes7.pdf) , Stanford University, http://www.stanford.edu/class/ee387/handouts/notes7.pdf, retrieved April 21, 2010
- Hong, Jonathan; Vetterli, Martin (August 1995), "Simple Algorithms for BCH Decoding", *IEEE Transactions on Communications* **43** (8): 2324–2333
- Koetter, Ralf (2005), (http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-451Spring-2005/LectureNotes/detail/embed10.htm) dead link] *Reed–Solomon Codes*, MIT Lecture Notes 6.451 (Video), http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-451Spring-2005/LectureNotes/detail/embed10.htm
- Lin, Shu; Costello, Jr., Daniel J. (1983), *Error Control Coding: Fundamentals and Applications*, New Jersey, NJ: Prentice-Hall, ISBN 0-13-283796-X
- MacWilliams, F. J.; Sloane, N. J. A. (1977), *The Theory of Error-Correcting Codes*, New York, NY: North-Holland Publishing Company
- Massey, J. L. (1969), "Shift-register synthesis and BCH decoding" (http://crypto.stanford.edu/~mironov/cs359/massey.pdf) , *IEEE Transactions on Information Theory* **IT-15** (1): 122–127, http://crypto.stanford.edu/~mironov/cs359/massey.pdf
- Peterson, Wesley W. (1960), "Encoding and Error Correction Procedures for the Bose-Chaudhuri Codes", *IRE Transactions on Information Theory* (Institute of Radio Engineers) **IT-6**: 459–470
- Reed, Irving S.; Chen, Xuemin (1999), *Error-Control Coding for Data Networks*, Boston, MA: Kluwer Academic Publishers
- Reed, Irving S.; Solomon, Gustave (1960), "Polynomial Codes over Certain Finite Fields", *Journal of the Society for Industrial and Applied Mathematics (SIAM)* **8** (2): 300–304, DOI:10.1137/0108018 (http://dx.doi.org/10.1137%2F0108018)
- Welch, L. R. (1997), *The Original View of Reed–Solomon Codes* (http://csi.usc.edu/PDF/RSoriginal.pdf) , Lecture Notes, http://csi.usc.edu/PDF/RSoriginal.pdf

External links

- Schifra Open Source C++ Reed–Solomon Codec (http://www.schifra.com)
- Henry Minsky's RSCode library, Reed–Solomon encoder/decoder (http://rscode.sourceforge.net/)

- Open source Verilog Reed-Solomon IP (http://opencores.org/project,reed_solomon_codec_generator)
- A Tutorial on Reed–Solomon Coding for Fault-Tolerance in RAID-like Systems (http://www.cs.utk.edu/%7Eplank/plank/papers/SPE-9-97.html)
- Algebraic soft-decoding of Reed–Solomon codes (http://sidewords.files.wordpress.com/2007/12/thesis.pdf)
- Matlab implementation of errors and-erasures Reed–Solomon decoding (http://dept.ee.wits.ac.za/~versfeld/research_resources/sourcecode/Errors_And_Erasures_Test.zip)
- BBC R&D White Paper WHP031 (http://www.bbc.co.uk/rd/pubs/whp/whp031.shtml)
- Geisel, William A. (August 1990), *Tutorial on Reed–Solomon Error Correction Coding* (http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900019023_1990019023.pdf) , Technical Memorandum, NASA, TM-102162, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900019023_1990019023.pdf

Retrieved from "http://en.wikipedia.org/w/index.php?title=Reed–Solomon_error_correction&oldid=501730948"
Categories: Error detection and correction | Coding theory
