Decoding Reed–Solomon Codes Using Euclid's Algorithm

Priti Shankar

Reed–Solomon codes are indeed an elegant illustration of a very practical application of abstract algebra. An interesting discovery was the fact that Euclid's algorithm for finding greatest common divisors of polynomials, could be used for decoding these codes. In this article we explain this connection.

Priti Shankar is with the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore. Her interests are in theoretical computer science and error correcting codes.

1. Introduction

Many of us who have seen the pictures sent back by the spacecraft Voyager 2, have been struck by the fine quality of the images received. In 1989, when Voyager was about 3 billion miles away from the Earth, it was able to transmit high resolution images of Triton, Neptune's largest moon (*Figure* 1). This great feat was in no small measure due to the fact that the sophisticated communication system on Voyager had an elaborate error correcting scheme built into it. A *Reed-Solomon* code was used to enhance the reliability of the transmission. Each codeword contained 223 bytes of data (a byte consisting of 8 bits) and 32 bytes of redundancy. Without the error correction scheme, Voyager would not have been able to send the volume of data that it did, using a transmitting power of only 20 watts. The spacecrafts Galileo (1991), Mars Global Surveyor (1997), Mars Pathfinder (1997) and Mars Exploration Rover (2004) have all used Reed–Solomon codes in their data transmission schemes. It would not be out of place to say that Reed-Solomon codes have gone to the furthest reaches of the solar system and beyond!

Irving Reed and Gus Solomon

[1] See *Resonance*, Vol.7, No.2, 2002.

Keywords
Error correcting codes, Reed–Solomon codes, decoding algorithms.

We owe to the genius of Claude Shannon[1], one of the

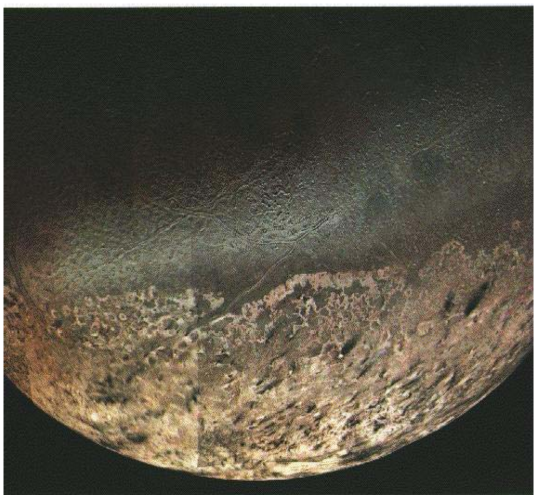
Figure 1. Image of Triton transmitted by Voyager 2. (*Courtesy*: NASA/JPL-Caltech)

finest scientific minds of the last century, the remarkable discovery that reliable communication is possible, even over an unreliable medium. In 1948, Shannon, then a young mathematician, showed that arbitrarily reliable communication is possible at any rate below something called the *channel capacity.* Roughly speaking, the channel capacity is its ultimate capability to transmit information, and one can push the channel to this limit and yet correct all the errors it makes using error correction. The illustration in *Figure* 2 shows Shannon's scheme for communicating reliably over an unreliable channel. The message is sent over the channel, but before it is sent it is processed by an *encoder.* The encoder combines the message with some *redundancy,* in order to create a *codeword* . This makes it possible to detect or correct errors introduced by the channel. At the receiver's end, the codeword, which may be corrupted by errors is *decoded* to recover the message.

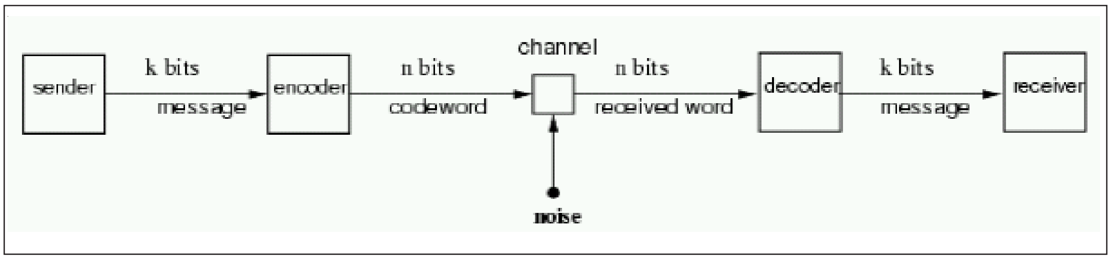

Figure 2. Shannon's schematic of a communication system.

In mathematical terms the important components of a coding scheme are as follows.

- The **input alphabet** is the set from which symbols to be transmitted are drawn. We assume that the input alphabet size is q and that the symbols are from $\mathbb{F}_q$ the field of q elements. The output alphabet is the set of symbols from which the channel produces outputs. Here we assume that the **output alphabet** set is also $\mathbb{F}_q$.
- The **message** $\boldsymbol{u}$ is a block of symbols that the sender wishes to transmit. For our purposes, $\boldsymbol{u}$ is an arbitrary word of k symbols from the finite field $\mathbb{F}_q$, that is $\boldsymbol{u} \in \mathbb{F}_q^k$.
- The **encoder** is an algorithm that the sender uses to pad the message $\boldsymbol{u}$ with redundant information. It can be looked upon as a function $enc : \mathbb{F}_q^k \to \mathbb{F}_q^n$ that transforms a message of k symbols to a **codeword** of n symbols.
- The **code** $\mathcal{C}$ is the set of all possible codewords. That is, $\mathcal{C} = \{enc(\boldsymbol{u}) : \boldsymbol{u} \in \mathbb{F}_q^k\}$.
- Three important parameters of a code are
- The **block length** of the code is the length n of the codewords output by the encoder.
- The **dimension** k of the code is the length of a message word. This term is typically used for *linear codes* when the code forms a vector space.
- The **rate** R of a code is the ratio of its dimension to its block length, or $R = k/n$.
- The **received word** $\boldsymbol{y} \in \mathbb{F}_q^n$ is the output of the channel which is a possibly corrupted version of the codeword $\boldsymbol{c}$.

The **encoder** is an algorithm that the sender uses to pad the message $\boldsymbol{u}$ with redundant information.

The **received word** $\boldsymbol{y} \in \mathbb{F}_q^n$ is the output of the channel which is a possibly corrupted version of the codeword $\boldsymbol{c}$.

In 1975 Sugiyama *et al.* discovered the beautiful fact that one can use Euclid's algorithm to decode a class of codes called Goppa codes and hence also Reed–Solomon codes.

- The **decoder** is an algorithm that the receiver uses to recover the original message $\boldsymbol{u}$ from the received word $\boldsymbol{y}$. It can be looked upon as a function $dec : \mathbb{F}_q^n \to \mathbb{F}_q^k$ that estimates a message word of k symbols from the received word $\boldsymbol{y}$ of n symbols.

The primary objective in coding theory is to design coding schemes with the best possible parameters and efficient encoding/decoding algorithms. Linear block codes defined below, are the best known error correcting codes and are the codes of choice in most practical applications.

A code $\mathcal{C}$ with block length n over an alphabet $\mathbb{F}_q$ is a **linear block code** if it is a linear subspace of $\mathbb{F}_q^n$, the vector space of n-tuples over $\mathbb{F}_q$. If $\mathcal{C}$ has dimension k, we will call the code an $(n,k)_q$ linear block code or an (n,k) code if the underlying field is assumed to be known.

Reed–Solomon codes are a class of linear block codes discovered by Irving Reed and Gus Solomon in 1960. While the codes were notable because of their elegant mathematical structure, their practical versatility became apparent only after the discovery in 1968, of an efficient decoding algorithm by Elwyn Berlekamp [1]. In 1975 Sugiyama *et al.* [2] discovered the beautiful fact that one can use Euclid's algorithm to decode a class of codes called Goppa codes and hence also Reed-Solomon codes. We will describe this algorithm presently, but we first present some background material.

2. Background

The *Hamming distance* between two n symbol words is the number of corresponding positions in which they differ. Thus if d symbol errors occur in a transmitted codeword, the Hamming distance between the codeword and the received word is d. The *minimum distance* of a code is the minimum of the Hamming distances between all pairs of distinct codewords. For linear codes, this is also the minimum *Hamming weight* of a codeword (that is, the number of non-zero components of the word). In general, if up to t symbol errors in an arbitrary codeword have to be corrected, the minimum distance d_{min} must satisfy

$$d_{min} \geq 2t + 1.$$

2.1. *Reed–Solomon Codes*

From now on we will concentrate on fields with q^m elements. The construction of these large fields is illustrated in *Box* 1.

We will use the fact that the non-zero elements of every such field can be generated by powers of a single element called a *primitive* element of the field.

Box 1. Construction of Finite Fields

The basic building blocks for these large fields or extension fields, as they are called, are the prime fields. $\mathbb{F}_p$ is the field whose elements are the set $\{0, 1, 2, ...p - 1\}$ where p is prime and all arithmetic is performed modulo p. We illustrate with the construction of a field with $p = 2$ and with $2^3 = 8$ elements. Let $V_3(\mathbb{F}_2)$ be the set of 3-tuples $\mathbf{a} = (a_2 a_1 a_0)$ over F_2, with addition being defined component wise. Thus $V_3(\mathbb{F}_2)$ is the set $\{000, 001, 010, 011, 100, 101, 110, 111\}$, and, for example, $(101) + (110) = (011)$. We can make $V_3(\mathbb{F}_2)$ into a field as follows. Assume $f(x)$ is of degree 3 over $\mathbb{F}_2$ and has no roots in $\mathbb{F}_2$. For example $f(x) = x^3 + x + 1$. (No roots in $\mathbb{F}_2$ would mean $f(1)$ and $f(0)$ are both non-zero modulo 2.) In such a case, $f(x)$ is said to be *irreducible* over $\mathbb{F}_2$. Associate with each tuple $\mathbf{a} = (a_2, a_1, a_0)$ in $V_3(\mathbb{F}_2)$ the polynomial $a_2x^2 + a_1x + a_0$, and define the product of tuples $\mathbf{a}$ and $\mathbf{b}$ to be the tuple $\mathbf{c}$ defined uniquely by the equation $a(x)b(x) = c(x)\ mod(f(x))$. ($c(x)$ is the remainder when the product $a(x)b(x)$ is divided by $f(x)$.) For example $(010)(101) = (001)$ as $(x)(x^2 + 1) \equiv 1\ mod(f(x))$. The field constructed above, has 2^3 elements. Of course, since there may be several irreducible polynomials of degree 3, there will be different ways to define the multiplication above. But all these fields are isomorphic and we can talk of *the* field with 2^3 elements called $GF(2^3)$. (We will refer to this field as $\mathbb{F}_8$.) Every field has what is called a *primitive* element, powers of which generate all the non-zero elements of the field. For example, the element α corresponding to the residue class $\{x\}$ or the tuple (010), is a primitive element of this field, consecutive powers of α generating the sequence of elements $(010), (100), (011), (110), (111), (101), (001)$. We denote the multiplicative inverse of a field element α by α^{-1}.

A (255,223) Reed–Solomon code has a redundancy of 32 symbols per codeword, a minimum distance 33, and can correct any combination of 16 symbol errors, each symbol being made up of 8 bits.

Reed–Solomon codes work with elements in a field F_{q^m}. We consider the case with $q = 2$ here. Reed-Solomon codes have parameters (n, k) where n is at most $2^m - 1$, m is the number of bits per symbol, and $k = n - 2t$ where t is the symbol error correction capability of the code. Thus, only $2t$ redundant symbols are required, to be able to correct t errors. For example, a (255,223) Reed–Solomon code has a redundancy of 32 symbols per codeword, a minimum distance 33, and can correct any combination of 16 symbol errors, each symbol being made up of 8 bits. We define a Reed-Solomon code as follows. Let a codeword $\boldsymbol{c} = (c_0, c_1, \ldots c_{n-1})$ be represented by a polynomial which we henceforth refer to as a code polynomial $c(x)$ given by

$$c(x) = c_0 + c_1 x + \ldots c_{n-1} x^{n-1}. \tag{1}$$

The polynomial $c(x)$ represents a codeword of the RS code if and only if $\alpha, \alpha^2, \ldots\ldots \alpha^{2t}$ are its roots where α is primitive in the multiplicative group of $\mathbb{F}_{2^m}$.

Thus every code polynomial has the above $2t$ consecutive powers of α as roots. It is not difficult to show (see the article on Reed–Solomon codes in [3]) that the minimum distance of such a code is at least $2t + 1$. We illustrate with an example borrowed from an article by the author on Reed–Solomon codes [3].

Suppose one wished to transmit messages in English uppercase. Then we would need at least 5 bits (which give a total of $2^5 = 32$ combinations) to encode all 26 letters, with 6 combinations left over to use for blanks and other special symbols like punctuation marks, etc. We could use a $(2^5 - 1, 15)$ Reed–Solomon code for our purpose. This code has symbols in $\mathbb{F}_{32}$ each requiring 5 bits for encoding. The code has a length of 31 symbols, with 15 message symbols and 16 check symbols, a minimum distance of 17, and hence an error correction capability of 8 symbols. Suppose we wanted to transmit the fifteen characters, (including the blank).

READ RESONANCE!

The encoder would add 16 more parity check symbols of its own to form a codeword. Assume for the sake of argument that the resultant codeword is

READ RESONANCE!QTBPJ!TL.,ZBFALK

Now even if the message is changed to

ROAD REPAIRSCE!STOPJ!TL.,ZBFALK

where errors occur in both the message, as well as the check symbols, the decoder would be able to correct all of these (as there are not more than 8 of them) and recover the original message!

Thus if the transmitted codeword is an n-tuple of symbols from the field $\mathbb{F}_q$, the received n-tuple is also made up of elements from the same field. If transmission errors have occurred, some of the received symbols may differ from those that were transmitted.

2.2 *Decoding an Error-Correcting Code*

Recall the assumption that the channel input and output alphabets are the same. Thus if the transmitted codeword is an n-tuple of symbols from the field $\mathbb{F}_q$, the received n-tuple is also made up of elements from the same field. If transmission errors have occurred, some of the received symbols may differ from those that were transmitted. If $\boldsymbol{x}$ is transmitted and $\boldsymbol{y}$ received, the difference $\boldsymbol{z} = \boldsymbol{y} - \boldsymbol{x}$ is called the *error pattern.* If the i^{th} component z_i of the vector $\boldsymbol{z}$ is not equal to 0, an error is said to have occurred in the i^{th} position. The decoding problem is to identify the error pattern given the received vector.

Let $\hat{\boldsymbol{x}}$ be the estimate of the transmitted codeword $\boldsymbol{x}$ output by the decoder. Thus, a decoding error occurs if and only if $\hat{\boldsymbol{x}} \neq \boldsymbol{x}$.

It is reasonable to assume that given a received vector, an error pattern with the smallest Hamming weight is

The problem to be addressed is: Given the received vector, which may be seen as the corruption of any one of the codewords with the appropriate error pattern, find the most likely transmitted codeword.

the most likely error pattern. The problem to be addressed is: Given the received vector, which may be seen as the corruption of any one of the codewords with the appropriate error pattern, find the most likely transmitted codeword. (Once we have this estimate we can easily recover the message). If the Hammming weight of the error pattern is at most t, and the minimum distance of the code is at least $2t + 1$ it is easy to see that there is exactly one codeword that is closest in terms of the Hamming distance to the received vector and this is the one the decoder should output. This kind of decoding is known as *bounded distance decoding.* A brute force approach would involve checking the Hamming distances of all codewords from the received vector, which is clearly infeasible. We therefore search for more efficient solutions to this problem.

3. The Key Equation

As we shall presently see, the bounded distance decoding problem can be formulated so that it reduces to the solution of an equation that is popularly known as the *Key Equation.* We first derive this equation and then show how Euclid's algorithm for finding the greatest common divisor (gcd), can be used to get the solution. Let a codeword $c = c_0, c_1, \dots c_{n-1}$ be represented by the codeword polynomial $c(x)$ defined earlier.

Assume we receive the polynomial $r(x)$ when $c(x)$ is transmitted. We define the error polynomial $e(x)$ as $e(x) = r(x) - c(x)$

This polynomial will have non-zero coefficients in positions where transmission errors have occurred.

For example if $e(x) = 1 + x^4$ then errors have occurred in the 0th and 4th positions.

The problem is to find the most likely error polynomial $e(x)$ given $r(x)$. Recall that the 'most likely' error pattern means the error pattern of minimum Hamming weight. Define

$$S_i = e(\alpha^i), i = 1, 2, \ldots n-1. \quad (2)$$

As every codeword has $\alpha^i, i = 1, 2, \ldots 2t$ as roots, and as $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ for some codeword $\boldsymbol{c}$ we have

$$S_i = r(\alpha^i), i = 1, 2, \ldots 2t, \quad (3)$$

where S_i is called the i^{th} syndrome.

Now if there were no errors then all the $2t$ syndromes defined above would be zero. Thus if one or more of these syndromes is non-zero it is an indication that errors have occurred in transmission.

Now if there were no errors then all the $2t$ syndromes defined above would be zero. Thus if one or more of these syndromes is non-zero it is an indication that errors have occurred in transmission.

The decoding problem can be restated as follows:

Given $S_1, S_2, \ldots S_{2t}$ find $e(x)$. When restated in this manner, the decoding problem looks like a problem in interpolation. We will obtain a formulation that allows the application of Euclid's polynomial gcd algorithm.

Let the Hamming weight of the error pattern e be at most t, that is, there are at most t non-zero components of the error pattern. Recall that we are then guaranteed that there is exactly one 'closest' codeword.

If the j^{th} non-zero component (counting from the left) of the error pattern is the digit e_k (where the subscripts of e begin with 0) then define $X_j = \alpha^k$.

For example, say $n = 7$. Then $e(x)$ can be written as $e(x) = e_0 + e_1x + e_2x^2 + \ldots + e_6x^6$. If the first and third coefficients are non-zero, i.e. e_1 and e_3 are non-zero, then the first non-zero coefficient is e_1 and the second is e_3. Thus $X_1 = \alpha^1$ and $X_2 = \alpha^3$, where α generates the cyclic group of order n in the multiplicative group of $GF(2^m)$. The X_j's are called the *error locations* as if α^j is known then j can be found. Note that this does not solve the decoding problem completely, as the *error*

We define the error magnitude Y_j at location X_j to be e_k, where as mentioned earlier, e_k is the j th non-zero component of the error vector.

magnitudes also need to be determined. We define the error magnitude Y_j at location X_j to be e_k, where as mentioned earlier, e_k is the j^{th} non-zero component of the error vector.

From equation (3), using the definitions of X_j and Y_j and the assumption that at most t errors have occurred we have

$$S_i = \sum_{j=1}^{t} Y_j X_j^i, i = 1, 2, \ldots n-1. \qquad (4)$$

Define the following series

$$\frac{1}{1 - X_j x} = 1 + X_j x + X_j^2 x^2 + \ldots \qquad (5)$$

Multiplying both sides of equation 5 by $Y_j X_j$ and summing both sides from 1 to t we have, using equation (4)

$$\sum_{j=1}^{t} \frac{Y_j X_j}{1 - X_j x} = S_1 + S_2 x + \ldots .S_{2t} x^{2t-1} + \ldots \qquad (6)$$

Combining the terms on the left hand side of equation (6) into a single fraction it can be written as

$$\frac{\sum_{j=1}^{t} Y_j X_j \prod_{k=1,k\neq j}^{t}(1 - X_k x)}{\prod_{j=1}^{t}(1 - X_j x)}. \qquad (7)$$

Replacing the numerator of equation (7) by $\omega(x)$ and the denominator by $\sigma(x)$ we can rewrite equation (6) as

$$\frac{\omega(x)}{\sigma(x)} = S_1 + S_2 x + S_3 x^2 + \ldots S_{2t} x^{2t-1} + \ldots . \qquad (8)$$

Equation (8) has the following properties;

1. The denominator $\sigma(x)$ on the left hand side is of degree one greater than that of the numerator.

2. The roots of the polynomial in the denominator are the inverses of the error locations. Therefore this polynomial is called the *error locator polynomial.*

3. The numerator and denominator on the left hand side are relatively prime polynomials. This can be verified by checking that none of the roots of the denominator are roots of the numerator.

4. Consider the derivative wrt x of the denominator. This is

$$\sigma'(x) = \sum_{j=1}^{t} -X_j \prod_{k=1, k\neq j} (1 - X_k x). \tag{9}$$

If this is evaluated at a particular value X_l^{-1} and we form $\frac{\omega(X_l^{-1})}{\sigma'(X_l^{-1})}$, we get

$$\frac{\omega(X_l^{-1})}{\sigma'(X_l^{-1})} = \frac{Y_l X_l}{-X_l} = -Y_l. \tag{10}$$

The right hand side is the negative of the error magnitude at the error location l. The polynomial $\omega(x)$ is therefore called the *error evaluator polynomial.*

The numerator and denominator on the left hand side are relatively prime polynomials. This can be verified by checking that none of the roots of the denominator are roots of the numerator.

Now suppose we compute both sides of equation modulo the polynomial x^{2t} and we set

$$S(x) = S_1 + S_2 x + S_3 x^2 + \dots S_{2t} x^{2t}, \tag{11}$$

we have the equation

$$\frac{\omega(x)}{\sigma(x)} \equiv S(x) (\text{mod } x^{2t}). \tag{12}$$

Equation (12) is called the key equation because if we can solve it and obtain the polynomials $\sigma(x)$ and $\omega(x)$, Euclid's algorithm is an iterative algorithm to find the gcd of a pair of polynomials. Our aim is to exploit the properties of the algorithm and those of the polynomials in the key equation so that $\sigma(x)$ and $\omega(x)$ fall out at the i th stage of the algorithm for some i.

then using properties 2 and 4 above, we can get the error locations as well as the error magnitudes of the error pattern, and thus a solution to the decoding problem. Note however, that only the polynomial $S(x)$ of degree at most $2t$ and with coefficients in $\mathbb{F}_{2^m}$ is known. We may pose the problem as follows: Given an approximation $S(x)$ to a rational function (which in this case is $\frac{\omega(x)}{\sigma(x)}$ where both the numerator and the denominator are unknown to the decoder), which agrees with the power series expansion of the rational function to the order of the first $2t$ terms, find a rational function where the numerator and denominator are relatively prime, the degree of the denominator is at most t, and the degree of the numerator is one less than that of the denominator.

In the next section we show how the Euclid algorithm for polynomial gcd's solves this problem.

4. Decoding Using Euclid's Algorithm

We state the gcd problem for polynomials as follows:

Given polynomials $a(x)$ and $b(x)$ with coefficients in a field, and where degree $(a(x)) \geq$ degree $(b(x))$, find polynomials $s(x)$ and $t(x)$ such that the following equation is satisfied:

$$s(x)a(x) + t(x)b(x) = r(x), \qquad (13)$$

where $r(x)$ is the gcd of $a(x)$ and $b(x)$.

Euclid's algorithm which we will presently describe, is an iterative algorithm to find the gcd of a pair of polynomials. Our aim is to exploit the properties of the algorithm and those of the polynomials in the key equation so that $\sigma(x)$ and $\omega(x)$ fall out at the i^{th} stage of the algorithm for some i.

We make use of the properties of the Euclid algorithm (not proved here) displayed in *Box* 2.

Box 2. Properties of the Euclid gcd Algorithm

At stage i of the iteration there will be four polynomials

$s_i(x), t_i(x), q_i(x)$ and $r_i(x)$. The first two are the ith multipliers, the next two are the ith quotient and the ith remainder. The equations that hold at each stage are:

$$r_{i-2}(x) = q_i(x)r_{i-1}(x) + r_i(x), \tag{a}$$

where

$$deg(r_i(x)) < deg(r_{i-1}(x)), \tag{b}$$

$$s_i(x) = s_{i-2}(x) - q_i(x)s_{i-1}(x), \tag{c}$$

$$t_i(x) = t_{i-2}(x) - q_i(x)t_{i-1}(x). \tag{d}$$

The initial conditions are:

$s_{-1}(x) = 1$	$t_{-1}(x) = 0$	$r_{-1}(x) = a(x)$
$s_0(x) = 0$	$t_0(x) = 1$	$r_0(x) = b(x)$

Three properties that hold during iteration are:

A: $s_i(x)a(x) + t_i(x)b(x) = r_i(x)$

B: degree($s_i(x)$)+ degree($r_{i-1}(x)$)= degree($b(x)$)

C: degree($t_i(x)$)+ degree($r_{i-1}(x)$) = degree($a(x)$)

Property A (*Box* 2) can be rewritten as:

$$t_i(x)b(x) \equiv r_i(x)\text{mod}(a(x)). \tag{14}$$

Equation (14) looks like the key equation with $a(x)$ playing the role of x^{2t}, and $t_i(x), b(x)$ and $r_i(x)$ the roles of $\sigma(x), S(x)$ and $\omega(x)$ respectively. As can be seen, the degree of the $t_i(x)$'s increases with the iteration i whereas the degree of the r_i's decreases with i. Using this fact, we get from property C

$$degree(t_i(x)) + degree(r_i(x)) < degree(a(x)). \tag{15}$$

The next lemma and theorem help identify the step at which we should halt when using Euclid's algorithm for decoding. We present these results without proofs. Readers interested in the proofs can look them up in the excellent book by McEliece [4].

Lemma 4.1 indicates that if there is any pair of integers whose sum is one less than the degree of $a(x)$, and where one of them is greater than the degree of the gcd, then there is a unique stage i in the Euclid algorithm where the degree of $t_i(x)$ is bounded above by one of the integers and the degree of $r_i(x)$ bounded above by the other.

Lemma 4.1 *Given two non-negative integers u and v with $v \geq$ degree(gcd($a(x), b(x)$)), satisfying $u + v =$ degree $a(x) - 1$, there exists a unique index j, $0 \leq j \leq n$ such that*

$$degree(t_j(x)) \leq u, \tag{16}$$

and

$$degree(r_j(x)) \leq v. \tag{17}$$

The lemma indicates that if there is any pair of integers whose sum is one less than the degree of $a(x)$, and where one of them is greater than the degree of the gcd, then there is a unique stage i in the Euclid algorithm where the degree of $t_i(x)$ is bounded above by one of the integers and the degree of $r_i(x)$ bounded above by the other. In other words, at least one of the bounds is violated before or after this unique index is reached in the iterative process. This is true for all pairs of integers satisfying the assumptions. Now $r_i(x)$ plays the role of $\omega(x)$ and we know that its degree of the latter is at most $t - 1$. This gives us a clue as to when to stop the iteration. However we need the next theorem to indicate how the polynomials obtained from Euclid's algorithm are related to the ones of interest to us.

Theorem 4.2. *Let $r(x)$ and $t(x)$ be polynomials satisfying*

$$t(x)b(x) \equiv r(x)\text{mod}(a(x)), \tag{18}$$

where degree $(t(x))+$ degree $(r(x)) <$ degree $(a(x))$.

Then there exists a unique index j and a polynomial $\lambda(x)$ such that

$$t(x) = \lambda(x)t_j(x), \tag{19}$$

$$r(x) = \lambda(x) r_j(x). \qquad (20)$$

We now have all the results we need to apply Euclid's algorithm to the solution of the key equation. We state this as a theorem.

Theorem 4.3. *If $\sigma(x)$ and $\omega(x)$ are the error locator and the error evaluator polynomials for an $(n, n-2t)$ Reed–Solomon code, and the error pattern has weight at most t, then*

$$\sigma(x) = \lambda t_j(x), \qquad (21)$$

$$\omega(x) = \lambda r_j(x), \qquad (22)$$

where r_j and t_j are the polynomials obtained from the application of Euclid's algorithm to the polynomials $a(x) = x^{2t}$ and $b(x) = S(x)$, where j is the first index at which the degree of $r_j(x)$ drops to below t. The constant λ is chosen so that $\lambda t_j(x)$ has constant term 1.

Thus the error locator polynomial $\sigma(x)$ and the error evaluator polynomial $\omega(x)$ can be found by simply applying Euclid's algorithm to x^{2t} and $S(x)$ and stopping when the degree of the remainder drops to below t!

Suggested Reading

[1] Elwyn Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[2] Y Sugiyama, M Kasahara, S Hirasawa and T Namekawa, *A method for solving key equation for decoding Goppa codes, Information and Control*, Vol.27, pp. 87–99, 1975.

[3] Priti Shankar, Error Correcting Codes: The Reed–Solomon Codes, *Resonance*, Vol.2, No.3, pp.33-47, 1997.

[4] R J McEliece, The Theory of Information and Coding, *Encyclopedia of Mathematics and its Applications*, Addison Wesley, 1977.

Address for Correspondence
Priti Shankar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012, India.
Email:
priti@csa.iisc.ernet.in