Mathematics 366 **Reed Solomon Codes** A. Hulpke

We have seen already, in the form of check digits, a method for detecting errors in data transmission. In general, however, one does not only on to recognize that an error has happened, but be able to correct it. This is not only necessary for dealing with accidents; it often is infeasible to build communication systems which completely exclude errors, a far better option (in terms of cost or guaranteeable capabilities) is to permit a limited rate of errors and to install a mechanism that deals with these errors. The most naïve schema for this is complete redundancy: retransmit data multiple times. This however introduces an enormous communication overhead, because in case of a single retransmission it still might not be possible to determine which of two different values is correct. In fact we can guarantee a better error correction with less overhead, as we will see below.
The two fundamental problems in coding theory are:

Theory: Find good (i.e. high error correction at low redundancy) codes.

Practice: Find efficient ways to implement encoding (assigning messages to code words) and decoding (determining the message from a possibly distorted code word).

In theory, almost (up to ε) optimal encoding can be achieved by choosing essentially random code generators, but then decoding can only be done by table lookup. To use the code in practice there needs to be some underlying regularity of the code, which will enable the algorithmic encoding and decoding. This is where algebra comes in.
A typical example of such a situation is the compact disc: not only should this system be impervious to fingerprints or small amounts of dust. At the time of system design (the late 1970s) it actually was not possible to mass manufacture compact discs without errors at any reasonable price. The design instead assumes errors and includes capability for error correction. It is important to note that we really want to correct all errors (this would be needed for a data CD) and not just conceal errors by interpolating erroneous data (which can be used with music CDs, though even music CDs contain error correction to deal with media corruption). This extra level of error correction can be seen in the fact that a data CD has a capacity of 650 MB, while 80 minutes of music in stereo, sampled at 44,100 Hz, with 16 bit actually amount to $80 \cdot 60 \cdot 2 \cdot 44100 \cdot 2$ bytes $=$ 846720000 bytes $= 808mB$, the overhead is extra error correction on a data CD.
The first step towards error correction is interleaving. As errors are likely to occur in bursts of physically adjacent data (the same holds for radio communication with a spaceship which might get disturbed by cosmic radiation) subsequent data ABCDEF is actually transmitted in the sequence ADBECF. An error burst – say erasing C and D – then is spread over a wider area AXBEXF which reduces the local error density. We therefore can assume that errors occur randomly distributed with bounded error rate.
The rest of this document now describes the mathematics of the system which is used for error correction on a CD (so-called Reed-Solomon codes) though for simplicity we will choose smaller examples than the actual codes being used.
We assume that the data comes in the form of elements of a field F.
The basic idea behind linear codes is to choose a sequence of code generators $g_1, \ldots, g_k$ each of length $> k$ and to replace a sequence of k data elements $a_1, \ldots, a_k$ by the linear combination

$a_1g_1 + a_2g_2 + \cdots + a_kg_k$, which is a slightly longer sequence. This longer sequence (the *code word*) is transmitted. We define the *code* to be the set of all possible code words.
For example (assuming the field with 2 elements) if the g_i are given by the rows of the *generator matrix* $\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$ we would replace the sequence a,b,c by a,b,c,a,b,c, while (a code with far better error correction!) the matrix $\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$ would have us replace a,b,c by

$$a,b,a+b,c,a+c,b+c,a+b+c.$$

We notice in this example that there are $2^3 = 8$ combinations, any two of which differ in at least 3 places. If we receive a sequence of 7 symbols with at most one error therein, we can therefore determine **in which place the error occurred** and thus can correct it. We thus have an *1-error correcting code*.
Next we want to assume (regularity!) that *any cyclic shift of a code word* is again a code word. We call such codes *cyclic codes*. It now is convenient to represent a code word $c_0,\ldots,c_{n-1}$ by the polynomial $c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}$[1]. A cyclic shift now means to multiply by x, identifying x^n with $x^0 = 1$. The transmitted messages therefore are not really elements of the polynomial ring $F[x]$, but elements of the factor ring $F[x]/I$ with $I = \langle x^n - 1\rangle \lhd F[x]$. The linear process of adding up generators guarantees that the sum or difference of code words is again a code word. A cyclic shift of a code word $I + w$ then corresponds to the product $(I + w)\cdot(I + x)$. As any element of $F[x]/I$ is of the form $I + p(x)$, demanding that cyclic shifts of code words are again code words thus implies that the product of any code word $(I + w)$ with any factor element $(I + r)$ is again a code word. We thus see that the code (i.e. the set of possible messages) is an ideal in $F[x]/I$. We have seen in homework problem 47 that any such ideal — and thus any cyclic code – must arise from an ideal $I \leq A \lhd F[x]$, by problem 45 we know that $A = \langle d\rangle$ with d a divisor of $x^n - 1$.
Together this shows that the code must be of the form $\langle I + d\rangle$ with $d \mid (x^n - 1)$. If $d = d_0 + d_1x + \cdots + d_sx^s$, we get in the above notation the generator matrix

$$\begin{pmatrix} d_0 & d_1 & \cdots & d_s & 0 & 0 & \cdots & 0 \\ 0 & d_0 & d_1 & \cdots & d_s & 0 & \cdots & 0 \\ 0 & 0 & d_0 & d_1 & \cdots & d_s & \cdots & 0 \\ & & & \vdots & & & & \\ 0 & 0 & \cdots & 0 & d_0 & d_1 & \cdots & d_s \end{pmatrix}.$$

As d is a divisor of $x^n - 1$ any further cyclic shift of the last generator can be expressed in terms of the other generators. One can imagine the encoding process as evaluation of a polynomial at more points than required for interpolation. This makes it intuitively clear that it is possible to recover the correct result if few errors occur. The more points we evaluate at, the better error correction we get (but also the longer the encoded message gets).

[1] Careful: We might represent elements of F also by polynomials. Do not confuse these polynomials with the code polynomials

The task of constructing cyclic codes therefore has been reduced to the task of finding suitable (i.e. guaranteeing good error correction and efficient encoding and decoding) polynomials d.

The most important class of cyclic codes are BCH-codes[2], which are defined by prescribing the zeroes of d. One can show that a particular regularity of these zeroes produces good error correction. Reed-Solomon Codes (as used for example on a CD) are an important subclass of these codes.

To define such a code we start by picking a field with q elements. Such fields exists for any prime power $q = p^m$ and can be constructed as quotient ring $\mathbb{Z}_p[x]/\langle p(x)\rangle$ for p an irreducible polynomial of degree m. (Using fields larger than $\mathbb{Z}_p$ is crucial for the mechanism.) The code words of the code will have length $q-1$. If we have digital data it often is easiest to simply bundle by bytes and to use a field with $q = 2^8 = 256$ elements.

We also choose the number $s < \frac{q-1}{2}$ of errors per transmission unit of $q-1$ symbols which we want to be able to correct. This is a trade-off between transmission overhead and correction capabilities, a typical value is $s = 2$.

It is known that in any finite field there is a particular element α (a *primitive root*), such that $\alpha^{q-1} = 1$ but $\alpha^k \neq 1$ for any $k < q-1$. (This implies that all powers α^k ($k < q-1$) are different. Therefore every nonzero element in the field can be written as a power of α.) As every power $\beta = \alpha^k$ also fulfills that $\beta^{q-1} = 1$ we can conclude that

$$x^{q-1} - 1 = \prod_{k=0}^{q-1} (x - \alpha^k).$$

The polynomial $d(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)\cdots(x-\alpha^{2s})$ therefore is a divisor of $x^{q-1}-1$ and therefore generates a cyclic code. This is the polynomial we use. The dimension (i.e. the minimal number of generators) of the code then is $q-1-2s$, respectively $2s$ symbols are added to every $q-1-2s$ data symbols for error correction.

For example, take $q = 8$. We take the field with 8 elements as defined in GAP via the polynomial x^3+x+1, the element α is represented by `Z(8)`. We also choose $s = 1$. The polynomial defining the code is therefore $d = (x-\alpha)(x-\alpha^2)$.

```
gap> x:=X(GF(8),"x");;
gap> alpha:=Z(8);
Z(2^3)
gap> d:=(x-alpha)*(x-alpha^2);
x^2+Z(2^3)^4*x+Z(2^3)^3
```

We build the generator matrix from the coefficients of the polynomial:

```
gap> coe:=CoefficientsOfUnivariatePolynomial(d);
[ Z(2^3)^3, Z(2^3)^4, Z(2)^0 ]
gap> g:=NullMat(5,7,GF(8));;
```

[2] R.C. BOSE (1901-1987), since 1970 on professor for statistics here at CSU; D.K. RAY-CHAUDHURI, professor at Ohio State (now retired); A. HOCQUENGHEM

```
gap> for i in [1..5] do g[i]{[i..i+2]}:=coe;od;Display(g);
 z^3 z^4   1   .   .   .   .
   . z^3 z^4   1   .   .   .             z = Z(8)
   .   . z^3 z^4   1   .   .
   .   .   . z^3 z^4   1   .
   .   .   .   . z^3 z^4   1
```

To simplify the following description we convert this matrix into RREF. (This is just a base change on the data and does not affect any properties of the code.)

```
gap> TriangulizeMat(g); Display(g);
   1   .   .   .   . z^3 z^4
   .   1   .   .   .   1   1             z = Z(8)
   .   .   1   .   . z^3 z^5
   .   .   .   1   . z^1 z^5
   .   .   .   .   1 z^1 z^4
```

What does this mean now: Instead of transmitting the data ABCDE we transmit ABCDEPQ with the two error correction symbols defined by

$$\begin{aligned} P &= \alpha^3 A + B + \alpha^3 C + \alpha D + \alpha E \\ Q &= \alpha^4 A + B + \alpha^5 C + \alpha^5 D + \alpha^4 E \end{aligned}$$

For decoding and error correction we want to find (a basis of) the relations that hold for all the rows of this matrix. This is the nullspace[3] of the matrix (and in fact is the same as if we had not RREF).

```
gap> h:=NullspaceMat(TransposedMat(g));;Display(h);
 z^3   1 z^3 z^1 z^1   1   .             z = Z(8)
 z^4   1 z^5 z^5 z^4   .   1
```

When receiving a message ABCDEPQ we therefore calculate two *syndromes*[4]

$$\begin{aligned} S &= \alpha^3 A + B + \alpha^3 C + \alpha D + \alpha E + P \\ T &= \alpha^4 A + B + \alpha^5 C + \alpha^5 D + \alpha^4 E + Q \end{aligned}$$

If no error occurred both syndromes will be zero. However now suppose an error occurred and instead of ABCDEPQ we receive A'B'C'D'E'P'Q' where $A' = A + E_A$ and so on. Then we get the syndromes

$$\begin{aligned} S' &= \alpha^3 A' + B' + \alpha^3 C' + \alpha D' + \alpha E' + P' \\ &= \alpha^3 (A + E_A) + (B + E_B) + \alpha^3 (C + E_C) + \alpha (D + E_D) + \alpha (E + E_E) + (P + E_P) \\ &= \underbrace{\alpha^3 A + B + \alpha^3 C + \alpha D + \alpha E + P}_{= 0 \text{ by definition of the syndrome}} + \alpha^3 E_A + E_B + \alpha^3 E_C + \alpha E_D + \alpha E_E + E_P \\ &= \alpha^3 E_A + E_B + \alpha^3 E_C + \alpha E_D + \alpha E_E + E_P \end{aligned}$$

[3] actually the right nullspace, which is the reason why we need to transpose in GAP, which by default takes left nullspaces

[4] These syndromes are not unique. Another choice of nullspace generators would have yielded other syndromes.

and similarly

$$T' = \alpha^4 E_A + E_B + \alpha^5 E_C + \alpha^5 E_D + \alpha^4 E_E + E_Q$$

If symbol A' is erroneous, we have that $T' = \alpha S'$, the error is $E_A = \alpha^4 S'$ (using that $\alpha^3 \cdot \alpha^4 = 1$).
If symbol B' is erroneous, we have that $T' = S'$, the error is $E_B = S'$.
If symbol C' is erroneous, we have that $T' = \alpha^2 S'$, the error is $E_C = \alpha^4 S'$.
If symbol D' is erroneous, we have that $T' = \alpha^4 S'$, the error is $E_D = \alpha^6 S'$.
If symbol E' is erroneous, we have that $T' = \alpha^3 S'$, the error is $E_E = \alpha^6 S'$.
If symbol P' is erroneous, we have that $S' \neq 0$ but $T' = 0$, the error is $E_P = S'$.
If symbol Q' is erroneous, we have that $T' \neq 0$ but $S' = 0$, the error is $E_Q = T'$.

For an example, suppose we want to transmit the message

$$A = \alpha^3, \quad B = \alpha^2, \quad C = \alpha, \quad D = \alpha^4, \quad E = \alpha^5.$$

We calculate

$$\begin{aligned} P &= \alpha^3 A + B + \alpha^3 C + \alpha D + \alpha E = \alpha^6 \\ Q &= \alpha^4 A + B + \alpha^5 C + \alpha^5 D + \alpha^4 E = 0 \end{aligned}$$

```
gap> a^3*a^3+a^2+a^3*a+a*a^4+a*a^5;
Z(2^3)^6
gap> a^4*a^3+a^2+a^5*a+a^5*a^4+a^4*a^5;
0*Z(2)
```

We can verify the syndromes for this message:

```
gap> S:=a^3*a^3+a^2+a^3*a+a*a^4+a*a^5+a^6;
0*Z(2)
gap> T:=a^4*a^3+a^2+a^5*a+a^5*a^4+a^4*a^5+0;
0*Z(2)
```

Now suppose we receive the message

$$A = \alpha^3, \quad B = \alpha^2, \quad C = \alpha, \quad D = \alpha^3, \quad E = \alpha^5, \quad P = \alpha^6, \quad Q = 0.$$

We calculate the syndromes as $S = 1$, $T = a^4$:

```
gap> S:=a^3*a^3+a^2+a^3*a+a*a^3+a*a^5+a^6;
Z(2)^0
gap> T:=a^4*a^3+a^2+a^5*a+a^5*a^3+a^4*a^5+0;
Z(2^3)^4
```

We recognize that $T = \alpha^4 S$, so we see that symbol D is erroneous. The error is $E_D = \alpha^6 S = \alpha^6$ and thus the correct value for D is $\alpha^3 - \alpha^6 = \alpha^4$.

```
gap> a^3-a^6;
Z(2^3)^4
```

It is not hard to see that if more (but not too many) than s errors occur, the syndromes still indicate that an error happened, even though we cannot correct it any longer. (One can in fact determine the maximum number of errors the code is guaranteed to detect, even if it cannot correct it any longer.) This is used in the CD system by combining two smaller Reed-Solomon codes in an interleaved way. The result is called a Cross-Interleave Reed-Solomon code (CIRC): Consider the message be placed in a grid like this:

$$\begin{matrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \end{matrix}$$

The first code acts on rows (i.e. ABCD, EFGH,...) and corrects errors it can correct. If it detects errors, but cannot correct them, it marks the whole row as erroneous, for example:

$$\begin{matrix} A & B & C & D \\ X & X & X & X \\ I & J & K & L \end{matrix}$$

The second code now works on columns (i.e. AXI, BXJ) and corrects the values in the rows marked as erroneous. (As we know which rows these are, we can still correct better than with the second code alone!) One can show that this combination produces better performance than a single code with higher error correction value s.

The Ubiquitous Reed-Solomon Codes

by Barry A. Cipra, Reprinted from *SIAM News*, Volume 26-1, January 1993

In this so-called Age of Information, no one need be reminded of the importance not only of speed but also of accuracy in the storage, retrieval, and transmission of data. It's more than a question of "Garbage In, Garbage Out." Machines do make errors, and their non-man-made mistakes can turn otherwise flawless programming into worthless, even dangerous, trash. Just as architects design buildings that will remain standing even through an earthquake, their computer counterparts have come up with sophisticated techniques capable of counteracting the digital manifestations of Murphy's Law.

What many might be unaware of, though, is the significance, in all this modern technology, of a five-page paper that appeared in 1960 in the *Journal of the Society for Industrial and Applied Mathematics*. The paper, "Polynomial Codes over Certain Finite Fields," by Irving S. Reed and Gustave Solomon, then staff members at MIT's Lincoln Laboratory, introduced ideas that form the core of current error-correcting techniques for everything from computer hard disk drives to CD players. Reed-Solomon codes (plus a lot of engineering wizardry, of course) made possible the stunning pictures of the outer planets sent back by Voyager II. They make it possible to scratch a compact disc and still enjoy the music. And in the not-too-distant future, they will enable the profitmongers of cable television to squeeze more than 500 channels into their systems, making a vast wasteland vaster yet.

"When you talk about CD players and digital audio tape and now digital television, and various other digital imaging systems that are coming–all of those need Reed-Solomon [codes] as an integral part of the system," says Robert McEliece, a coding theorist in the electrical engineering department at Caltech.

Why? Because digital information, virtually by definition, consists of strings of "bits"–0s and 1s–and a physical device, no matter how capably manufactured, may occasionally confuse the two. Voyager II, for example, was transmitting data at incredibly low power–barely a whisper–over tens of millions of miles. Disk drives pack data so densely that a read/write head can (almost) be excused if it can't tell where one bit stops and the next one (or zero) begins. Careful engineering can reduce the error rate to what may sound like a negligible level–the industry standard for hard disk drives is 1 in 10 billion–but given the volume of information processing done these days, that "negligible" level is an invitation to daily disaster. Error-correcting codes are a kind of safety net–mathematical insurance against the vagaries of an imperfect material world.

The key to error correction is redundancy. Indeed, the simplest error-correcting code is simply to repeat everything several times. If, for example, you anticipate no more than one error to occur in transmission, then repeating each bit three times and using "majority vote" at the receiving end will guarantee that the message is heard correctly (e.g., 111 000 011 111 will be correctly heard as 1011). In general, n errors can be compensated for by repeating things $2n+1$ times.

But that kind of brute-force error correction would defeat the purpose of high-speed, high-density information processing. One would prefer an approach that adds only a few extra bits to a given message. Of course, as Mick Jagger reminds us, you can't always get what you want–but if you try, sometimes, you just might find you get what you need. The success of Reed-Solomon codes bears that out.

In 1960, the theory of error-correcting codes was only about a decade old. The basic theory of reliable digital communication had been set forth by Claude Shannon in the late 1940s. At the same time, Richard Hamming introduced an elegant approach to single-error correction and double-error detection. Through the 1950s, a number of researchers began experimenting with a variety of error-correcting codes. But with their SIAM journal paper, McEliece says, Reed and Solomon "hit the jackpot."

The payoff was a coding system based on *groups* of bits–such as bytes–rather than individual 0s and 1s. That feature makes Reed-Solomon codes particularly good at dealing with "bursts" of errors: Six consecutive bit errors, for example, can affect at most two bytes. Thus, even a double-error-correction version of a Reed-Solomon code can provide a comfortable safety factor. (Current implementations of Reed-Solomon codes in CD technology are able to cope with error bursts as long as 4000 consecutive bits.)

Mathematically, Reed-Solomon codes are based on the arithmetic of finite fields. Indeed, the 1960 paper begins by defining a code as "a mapping from a vector space of dimension $m$over a finite field K into a vector space of higher dimension over the same field." Starting from a "message" $(a_0, a_1, ..., a_{m-1})$, where each a_k is an element of the field K, a Reed-Solomon code produces $(P(0), P(\alpha), P(\alpha^2), ..., P(\alpha^{N-1}))$, where N is the number of elements in K, α is a generator of the (cyclic) group of nonzero elements in K, and $P(x)$ is the polynomial $a_0 + a_1 x + ... + a_{m-1}x^{m-1}$. If N is greater than m, then the values of P overdetermine the polynomial, and the properties of

finite fields guarantee that the coefficients of *P*–i.e., the original message–can be recovered from any *m* of the values.

Conceptually, the Reed-Solomon code specifies a polynomial by "plotting" a large number of points. And just as the eye can recognize and correct for a couple of "bad" points in what is otherwise clearly a smooth parabola, the Reed-Solomon code can spot incorrect values of P and still recover the original message. A modicum of combinatorial reasoning (and a bit of linear algebra) establishes that this approach can cope with up to *s* errors, as long as *m*, the message length, is strictly less than $N - 2s$.

In today's byte-sized world, for example, it might make sense to let *K* be the field of degree 8 over Z_2, so that each element of *K* corresponds to a single byte. In that case, $N = 2^8 = 256$, and hence messages up to 251 bytes long can be recovered even if two errors occur in transmitting the values $P(0), P(\alpha), ..., P(\alpha^{255})$. That's a lot better than the 1255 bytes required by the say-everything-five-times approach.

Despite their advantages, Reed-Solomon codes did not go into use immediately–they had to wait for the hardware technology to catch up. "In 1960, there was no such thing as fast digital electronics"–at least not by today's standards, says McEliece. The Reed-Solomon paper "suggested some nice ways to process data, but nobody knew if it was practical or not, and in 1960 it probably wasn't practical."

But technology did catch up, and numerous researchers began to work on implementing the codes. One of the key individuals was Elwyn Berlekamp, a professor of electrical engineering at the University of California at Berkeley, who invented an efficient algorithm for decoding the Reed-Solomon code. Berlekamp's algorithm was used by Voyager II and is the basis for decoding in CD players. Many other bells and whistles (some of fundamental theoretic significance) have also been added. Compact discs, for example, use a version called cross-interleaved Reed-Solomon code, or CIRC.

Reed, now a professor of electrical engineering at the University of Southern California, is still working on problems in coding theory. Solomon, recently retired from the Hughes Aircraft Company, consults for the Jet Propulsion Laboratory. Reed was among the first to recognize the significance of abstract algebra as the basis for error-correcting codes.

"In hindsight it seems obvious," he told *SIAM News*. However, he added, "coding theory was not a subject when we published that paper." The two authors knew they had a nice result; they didn't know what impact the paper would have. Three decades later, the impact is clear. The vast array of applications, both current and pending, has settled the question of the practicality and significance of Reed-Solomon codes. "It's clear they're practical, because everybody's using them now," says Berlekamp. Billions of dollars in modern technology depend on ideas that stem from Reed and Solomon's original work. In short, says McEliece, "it's been an extraordinarily influential paper."