

## Chapter 20

# Protecting against information loss: coding theory

Computer and information systems are prone to data loss—lost packets, crashed or corrupted hard drives, noisy transmissions, etc.—and it is important to prevent actual loss of important information when this happens. Today’s lecture concerns *error correcting codes*, a stepping point to many other ideas, including a big research area (usually based in EE departments) called *information theory*. This area started with a landmark paper by Claude Shannon in 1948, whose key insight was that data transmission is possible despite noise and errors if the data is *encoded* in some redundant way.

EXAMPLE 47 (ELEMENTARY WAYS OF INTRODUCING REDUNDANCY) The simplest way to introduce *redundancy* is to repeat each bit, say 5 times. The cons are (a) large inefficiency (b) no resistance to *bursty* error, which may wipe out all 5 copies.

Another simple method is *checksums*. For instance suppose we transmit 3 bits  $b_1, b_2, b_3$  as  $b_1, b_2, b_3, b_1 \oplus b_2 \oplus b_3$  where the last bit is the *parity* of the first three. Then if one of the bits gets flipped, the parity will be incorrect. However, if two bits get corrupted, the parity becomes correct again! Thus this method can detect when a single bit has been corrupted. It is useful in settings where errors are rare: if an error in the checksum is detected, the entire information/packet can be retransmitted.

A cleverer checksum method used by some cloud services is to store three bits  $b_1, b_2, b_3$  as 7 bits on 7 servers:  $b_1, b_2, b_3, b_1 \oplus b_2, b_1 \oplus b_3, b_2 \oplus b_3, b_1 \oplus b_2 \oplus b_3$ . It is easily checked that: if up to three servers fail, each bit is still recoverable, and in fact by querying at most 2 servers. A cleverer design of such *data storage codes* recently saved Microsoft 13% space on its cloud servers.

EXAMPLE 48 (GENERALIZED CHECKSUMS) A trivial extension of the checksum idea is to encode  $k$  bits using  $2^k$  checksums: take the parity of all possible subsets. This works to protect the data even if close to half the bits get flipped (though we won’t prove it; requires some Fourier analysis).

Another form of checksums is to designate some random subsets of  $\{1, 2, \dots, k\}$ , say  $S_1, S_2, \dots, S_m$ . Then encode any  $k$  bit vector using the  $m$  checksums corresponding to

these subsets. This works against  $\Omega(m)$  errors but we don't know of an efficient decoding algorithm. (Decoding in  $\exp(k)$  time is no problem.)

## 20.1 Shannon's Theorem

Shannon considered the following problem: a message  $x \in \{0,1\}^n$  has to be sent over a channel which flips every bit with probability  $p$ . How can we ensure that the message is recovered correctly at the other end? A couple of years later Hamming introduced a related notion whereby the channel flips up to  $p$  fraction of bits —and can adversarially decide which subset of bits to flip. He was concerned that real channels exhibit burstiness: make a lot of errors in one go and then no errors for long periods. By Chernoff bounds, Shannon's channel is a subcase (whp) of the Hamming channel since the chance of flipping more than  $p + \epsilon$  fraction of bits in total is  $\exp(-\Theta(n))$ . Both kinds of channels have been studied since then and we will actually use Hamming's notion today.

Shannon suggested that the message be encoded using a function  $E : \{0,1\}^n \rightarrow \{0,1\}^m$  and at the other end it should be *decoded* using a function  $D : \{0,1\}^m \rightarrow \{0,1\}^n$  with the property that  $D(E(x) \oplus \eta) = x$  for any noise vector  $\eta \in \{0,1\}^m$  that is 1 in at most  $pm$  indices and 0 in the rest. (Here  $\oplus$  of two bit vectors denotes bitwise parity.)

Clearly, such a decoding is possible if for every two messages  $x, x'$  their encodings differ in more than  $2pm$  bits: then  $E(x) \oplus \eta_1$  will not be confused for  $E(x') \oplus \eta_2$  for any two noise vectors  $\eta_1, \eta_2$  that only are nonzero in  $pm$  bits. We say such a code has *minimum distance* at least  $2pm$ .

The famous entropy function appearing in the following theorem is graphed below. (The notion of Entropy used in the 2nd law of thermodynamics is closely related.)

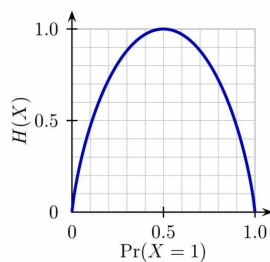


Figure 20.1: The graph of  $H(X)$  as a function of  $X$ .

### THEOREM 33

Such  $E, D$  do not exist if  $m < \frac{n}{1-H(p)}$ , and do exist for  $p \leq 1/4$  if  $m > \frac{n}{1-H(2p)}$ . Here  $H(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$  is the so-called entropy function.

PROOF: We only prove *existence*; the method does not give efficient algorithms to encode/decode. For any string  $y \in \{0,1\}^m$  let  $\text{Ball}(y)$  denote the set of strings that differ

from  $y$  in at most

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{2pm},$$

which is at most  $2^{H(2p)m}$  by Stirling's approximation.

Define the encoding function  $E$  using the following greedy procedure. Number the strings in  $\{0, 1\}^n$  from 1 to  $2^n$  and one by one assign to each string  $x$  its encoding  $E(x)$  as follows. The first string is assigned an arbitrary string in  $\{0, 1\}^m$ . At step  $i$  the  $i$ th string is assigned an arbitrary string that lies outside  $\text{Ball}(E(x))$  for all  $x \leq i - 1$ .

By design, such an encoding function satisfies that  $E(x)$  and  $E(x')$  differ in at least  $2pm$  fraction. Thus we only need to show that the greedy procedure succeeds in assigning an encoding to each string. To do this it suffices to note that if  $2^m > 2^n 2^{H(2p)m}$  then the greedy procedure never runs out of strings to assign as encodings.

The nonexistence is proved in a similar way. Now for  $y' \in \{0, 1\}^m$  let  $\text{Ball}'(y)$  be the set of strings that differ from  $y$  in at most  $pm$  indices. By a similar calculation as above, this has cardinality about  $2^{H(p)m}$ . If an encoding function exists, then  $\text{Ball}'(E(x))$  and  $\text{Ball}'(E(x'))$  must be disjoint for all  $x \neq x'$  (since otherwise any string in the intersection would not have an unambiguous encoding). Hence  $2^n \times 2^{H(p)m} < 2^m$ , which implies that  $m > \frac{n}{1-H(p)}$ .  $\square$

## 20.2 Finite fields and polynomials

Below we will design error correcting codes using polynomials over finite fields. Here *finite field* will refer to  $Z_q$ , the integers modulo a prime  $q$ . Recall that one can define  $+$ ,  $\times$ ,  $\div$  over these numbers, and that  $x \times y = 0$  iff at least one of  $x, y$  is 0. A degree  $d$  polynomial  $p(x)$  has the form

$$a_0 + a_1x + a_2x^2 + \cdots + a_dx^d.$$

It can be seen as a function that maps  $x \in Z_q$  to  $p(x)$ .

LEMMA 34 (POLYNOMIAL INTERPOLATION)

For any set of  $n + 1$  pairs  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$  where the  $x_i$ 's are distinct elements of  $Z_q$ , there is a unique degree  $n$  polynomial  $g(x)$  satisfying  $g(x_i) = y_i$  for each  $i$ .

PROOF: Let  $a_0, a_1, \dots, a_n$  be the coefficients of the desired polynomial. Then the constraint  $g(x_i) = y_i$  corresponds to the following linear system.

This system has a unique solution iff the matrix on the left is invertible, i.e., has nonzero determinant. This is nothing but the famous *Vandermonde* matrix, whose determinant is  $\prod_{i < j} (x_j - x_i)$ . This is nonzero since the  $x_i$ 's are distinct. Thus the system has a solution. Actually the solution has a nice description via the Lagrange interpolation formula:

$$g(x) = \sum_{i=0}^n y_i \prod_{j \neq i} \frac{(x - x_j)}{x_i - x_j}.$$

$\square$

COROLLARY 35

If a degree  $d$  has more than  $d$  roots (i.e., points where it takes zero value) then it is the zero polynomial.

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

Figure 20.2: Linear system corresponding to polynomial interpolation; matrix on left side is *Vandermonde*.

### 20.3 Reed Solomon codes and their decoding

The Reed Solomon code from 1960 is ubiquitous, having been used in a host of settings including data transmission by NASA vehicles and the storage standard for music CDs. It is simple and inspired by Lemma 34. The idea is to break up a message into chunks of  $\lfloor \log q \rfloor$  bits, where each chunk is interpreted as an element of the field  $Z_q$ . If the message has  $(d+1)\lfloor \log q \rfloor$  bits then it can be interpreted as coefficients of a degree  $d$  polynomial  $p(x)$ . The encoding consists of evaluating this polynomial at  $n$  points  $u_1, u_2, \dots, u_n \in Z_q$  and defining the encoding to be  $p(u_1), p(u_2), \dots, p(u_n)$ .

Suppose the channel corrupts  $k$  of these values, where  $n - k \geq d + 1$ . Let  $v_1, v_2, \dots, v_n$  denote the received values. If we knew which values are uncorrupted, the decoder could use polynomial interpolation to recover  $p$ . Trouble is, the decoder has no idea which received value has been corrupted. We show how to recover  $p$  if  $k < \frac{n-d}{2} - 1$ .

LEMMA 36

There exists a nonzero degree  $k$  polynomial  $e(x)$  and a polynomial  $c(x)$  of degree at most  $d + k$  such that

$$c(u_i) = e(u_i)v_i \quad \text{for } i = 1, 2, \dots, n. \quad (20.1)$$

PROOF: Let  $I \subseteq \{1, 2, \dots, n\}$ , with  $|I| = k$  be the subset of indices  $i$  such that  $v_i$  has been corrupted. Then (20.1) is satisfied by  $e(x) = \prod_{i \in I} (x - u_i)$  and  $c(x) = e(x)p(x)$  since  $e(u_i) = 0$  for each  $i \in I$  and nonzero outside  $I$ .  $\square$

The polynomial  $e$  in the previous proof is called the *error locator polynomial*. Now note that if we let the coefficients of  $c, e$  be unknowns, then (20.1) is a system of  $n$  equations in  $d + 2k + 2$  unknowns. This system is *overdetermined* since the number of constraints exceeds the number of variables. But Lemma 36 guarantees this system is feasible, and thus can be solved in polynomial time by Gaussian elimination.

We will need the notion of a polynomial *dividing* another. For instance  $x^2 + 2$  divides  $x^3 + x^2 + 2x + 2$  since  $x^3 + x^2 + 2x + 2 = (x^2 + 2)(x + 1)$ . The algorithm to divide one polynomial by another is the obvious analog of integer division.

LEMMA 37

If  $n > d + 2k + 1$  then any solution  $c(x), e(x)$  to the system of Lemma 36 satisfies (i)  $e(x)$  divides  $c(x)$  as a polynomial (ii)  $c(x)/e(x)$  is  $p(x)$ .

PROOF: The polynomial  $c(x) - e(x)p(x)$  has a root at  $u_i$  whenever  $v_i$  is uncorrupted since  $p(u_i) = v_i$ . Thus this polynomial, which has degree  $d + k$ , has  $n - k$  roots. Thus if  $n - k > d + k + 1$  this polynomial is identically 0.  $\square$

## 20.4 Code concatenation

Technically speaking, the Reed-Solomon code only works if the error rate of the channel is less than  $1/\log_2 q$ , since otherwise the channel could corrupt one bit in *every* value of the polynomial.

To allow error rate  $\Omega(1)$  one uses *code concatenation*. This means that we encode each value of  $p$ —which is a string of  $t = \lceil \log_2 q \rceil$  bits—with another code that maps  $t$  bits to  $O(t)$  bits and has minimum distance  $\Omega(t)$ . Wait a minute: you might say. If we had such a code all along then why go to the trouble of defining the Reed-Solomon code?

The reason is that we do have such a code by Shannon's construction (or by trivial checksums; see Example 48): but since we are only applying it on strings of size  $t$  it can be encoded and decoded in  $\exp(t)$  time, which is only  $q$ . Thus if  $q$  is polynomial in the message size, we still get encoding/decoding in polynomial time.

This technique is called code concatenation. One can also use any other error correcting code instead of Shannon's trivial code.