PRINCETON UNIV. F'17 COS 521: ADVANCED ALGORITHM DESIGN

Lecture 23: Protecting against Information Loss: Coding Theory

Lecturer: *Matt Weinberg* Scribe:*Sanjeev Arora*

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 1 (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 2 (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 $\{1, 2, \ldots, k\}$, say $S_1, S_2, \ldots, S_m$. 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.)

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 $E : \{0,1\}^n \to \{0,1\}^m$ and at the other end it should be *decoded* using a function $D : \{0,1\}^m \to \{0,1\}^n$ 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 η_1, η_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.)

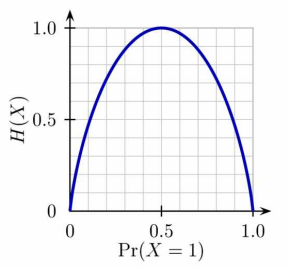

Figure 1: The graph of $H(X)$ as a function of X.

THEOREM 1
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 Ball(y) denote the set of strings that differ from y in at most $2pm$ indices. The number of strings in Ball$(E(x))$ is at most

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

which is at most $2^{H(2p)m}$ by Stirling's approximation.[1]

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

[1] We won't do the complete calculation, but here's some intuition: when $p \leq 1/4$, of the terms above, $\binom{m}{2pm}$ is the largest. Using Stirling's approximation, this is at most $(\frac{m}{2pm})^{2pm} \cdot (\frac{m}{(1-2p)m})^{(1-2p)m} = (1/2p)^{2pm} \cdot (1/(1-2p))^{(1-2p)m} = 2^{-2p \log_2(2p)m - (1-2p) \log_2(1-2p)m} = 2^{H(2p)m}$. Of course, this ignores the previous $2pm-1$ terms, but the use of Stirling's approximation was weak, and they balance out. Again, just intuition.

follows. The first string is assigned an arbitary string in $\{0,1\}^m$. At step i the ith string is assigned an arbitary string that lies outside 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 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 Ball$'(E(x))$ and 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 $m > \frac{n}{1-H(p)}$. □

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

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d.$$

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

Lemma 2 (Polynomial Interpolation)
For any set of $n+1$ pairs $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ 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 $a_0, a_1, \ldots, a_n$ be the coefficients of the desired polynomial. Then the constraint $g(x_i) = y_i$ corresponds to the following linear system.

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

Figure 2: Linear system corresponding to polynomial interpolation; matrix on left side is *Vandermonde*.

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 $\prod_{i \leq n} \prod_{j<i} (x_i - x_j)$. 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:

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

Above, one can verify that when $k \neq i$, the term $y_i \prod_{j \neq i} \frac{x_k - x_j}{x_i - x_j} = 0$, as there is a term $(x_k - x_k)$ in the product. So the only non-zero term is when $i = k$, and it is exactly y_k. So $g(x_k) = y_k$. The previous linear algebra shows that this polynomial is the unique solution. □

Corollary 3
If a degree d has more than d roots (i.e., points where it takes zero value) then it is the zero polynomial.

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 2. 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 $u_1, u_2, \ldots, v_n \in Z_q$ and defining the encoding to be $p(u_1), p(u_2), \ldots, p(u_n)$.

Suppose the channel corrupts k of these values, where $n - k \geq d + 1$. Let $v_1, v_2, \ldots, v_n$ 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 $k < \frac{n-d}{2} - 1$ $(n > d + 2k + 1)$.

Lemma 4
There exists a nonzero degree k polynomial $e(x)$ and a polynomial $c(x)$ of degree at most $d + k$ such that

$$c(u_i) = e(u_i)p(v_i) \qquad \text{for } i = 1, 2, \ldots, n. \tag{1}$$

Proof: Let $I \subseteq \{1, 2, \ldots, n\}$, with $|I| = k$ be the subset of indices i such that v_i has been corrupted. Then (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. □

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 (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 4 guarantees this system is feasible, and thus can be solved in polynomial time by Gaussian elimination. Note at this point that we can just recover *some* candidate $e(\cdot), c(\cdot)$. If we recovered exactly the $e(\cdot), c(\cdot)$ defined in the proof of Lemma 4, we'd be in great shape: we can exactly see where the errors are by the divisors of $e(\cdot)$. But maybe there are multiple solutions and the polynomial we recover isn't the desired $e(\cdot)$. Below, we'll show that no matter what polynomial we recover, we can locate the errors (and then do polynomial interpolation).

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 5
If $n > d + 2k + 1$ then any solution $c(x), e(x)$ to the system of Lemma 4 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 (meaning that $c(x) = e(x)p(x)$ as polynomials. □

Let's just check now what we can do with this code as compared to repetition. Say that the error rate is $1/(4 \log_2 q)$, and we have a total of $(d + 1) \log_2 q$ bits to send. If we want to tolerate independent errors, then we need to repeat each bit $\Omega(\log d \log_2 q)$ times to guarantee that each bit (we didn't go over how to prove this in class, but it's essentially by showing that you need this many repetitions to get a Chernoff bound to work for each bit, and that the Chernoff bound is approximately tight). So the total communication would be $\Omega(d \log_2 q \log(d \log_2 q))$.

If instead we use the Reed-Solomon code, we need to send $n > d + 2k + 1$ pairs of points in Z_q, where k is the number of tolerated errors. Because each bit is flipped with probability $1/(4 \log_2 q)$, we expect the total number of mistaken bits to be $d/4$, and let's assume in the worst case that they occur all in different pairs. So we want to tolerate $\approx k = d/2$ errors, and therefore we need to send $O(d)$ pairs of points in Z_q (and each pair takes $2 \log_2 q$ bits to represent). So our total communication will be just $O(d \log_2 q)$.

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 2): 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.