CS 880: Advanced Complexity Theory	3/3/06
Lecture 15: Coding Theory part II	
Instructor: Dieter van Melkebeek	Scribe: Scott Diehl, Jeff Kinne

In the last lecture we introduced the traditional paradigm of coding theory: a source encodes a message and sends the message through some (possibly noisy) channel to a receiver, who decodes the message. In the setting of data compression, we assume no noise on the transmission channel and wish to minimize the amount of information that is transmitted over the channel (through clever encoding/decoding schemes). We are interested in the setting of worst case error correcting codes. Today we give bounds on how good these codes can be asymptotically.

1 Introduction

We are interested in the setting of worst-case error correcting codes.

Definition 1. *A code $f : \Sigma^k \to \Sigma^n$ is an (n, k, d) code if for any two codewords of f, they differ in at least d positions.*

The rate of the code is $r = \frac{k}{d}$, and the relative distance is $\delta = \frac{d}{n}$.

For a family of codes $\mathcal{C}$, we define the asymptotic rate and relative distance:

$$r(\mathcal{C}) = \liminf_{n\to\infty} r(\mathcal{C}_n)$$

$$\delta(\mathcal{C}) = \liminf_{n\to\infty} \delta(\mathcal{C}_n).$$

We will assume the use of the binary alphabet $\Sigma = \{0, 1\}$, although most of the arguments in this lecture generalize to any alphabet. In the worst case error correction setting, the goal is to achieve an efficiently encodable and decodable code that maximizes both the rate and the relative distance of the code, thus minimizing the amount of information that needs to be transmitted and maximizing the number of errors that can be corrected.

In the next lecture we will present a code based on expanders that has both $r(\mathcal{C}) > 0$ and $\delta(\mathcal{C}) > 0$. Such codes are called *good*. Today we will see a number of impossibility results on how good a code can be, placing upper bounds on the following quantity that measures the relation of the rate and relative distance of all possible codes.

Definition 2. *For $0 \le \delta \le 1$,*

$$R(\delta) = \sup(\{r \mid \exists \text{ a code } \mathcal{C} \text{ s.t. } r(\mathcal{C}) \ge r \text{ and } \delta(\mathcal{C}) \ge \delta\}).$$

Consider the extremes of δ. For $\delta = 0$, different codewords do not have any distance between them. This can clearly be accomplished with rate 1, so $R(0) = 1$. For $\delta = 1$, this means that every 2 codewords differ in all positions. This means that once a symbol in the alphabet is used for a position, it cannot be used for that position in any other codeword. Hence there will at most $|\Sigma|$ codewords, and the rate is asymptotically 0, meaning $R(1) = 0$. The bounds we will present in this lecture are presented graphically in figure (1) (recall that for $0 \le \delta \le 1$, $H(\delta) = \delta \log(\frac{1}{\delta}) + (1 - \delta) \log(\frac{1}{1-\delta})$, which is the entropy of the 0/1 distribution with $\Pr[X = 0] = \delta$).

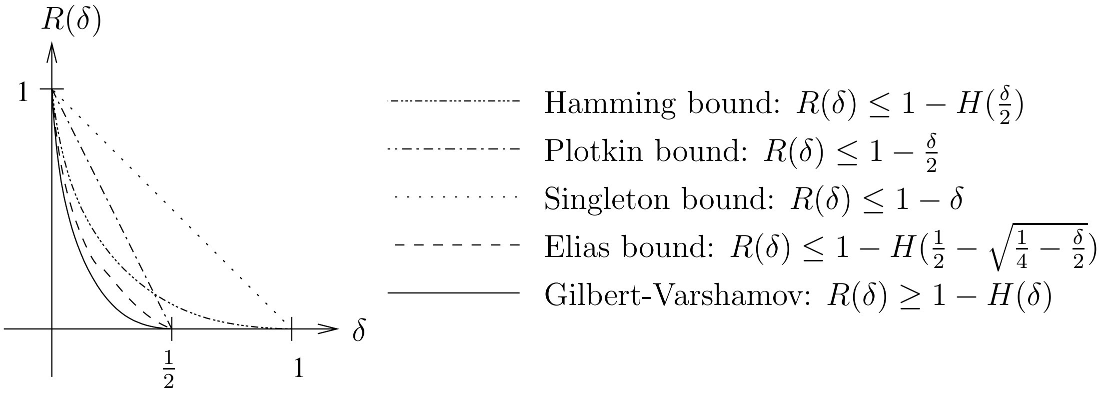

Figure 1: Upper and lower bounds on $R(\delta)$. The lower bound (i.e. possibility result) was presented in the last lecture. The upper bounds (i.e. impossibility results) are presented in this lecture.

Note that these bounds on $R(\delta)$ are for arbitrary codes: even if a code with certain $R(\delta)$ is realizable, there may still be no efficient code with such $R(\delta)$. In this sense, these upper bounds are quite strong (even inefficient codes are ruled out), while the lower bound presented last lecture is somewhat weak (the construction did not give an efficient and explicit code).

2 Singleton bound

The first upper bound which we present is based on a technique of iteratively "puncturing" a code.

Theorem 1 (Singleton Bound)**.** $R(\delta) \leq 1 - \delta$.

Proof. Consider all codewords of an (n, k, d) code given by $f : \{0,1\}^k \to \{0,1\}^n$. For the first bit position, either 0 or 1 has higher probability over all codewords - suppose 0 is the more popular. We create a new code f_1 by removing the first bit from the code. If $f(x_1, x_2, ..., x_k) = (0, y_2, ..., y_n)$, we set $f_1(x_1, x_2, ..., x_k) = (y_2, ..., y_n)$. If $f(x_1, x_2, ..., x_k) = (1, y_2, ..., y_n)$, then $(x_1, x_2, ..., x_k)$ is no longer in the domain of f_1.

Thus, we have removed one bit from the output of f and have removed at most $1/2$ of the strings from the domain by constructing f_1. Each string from the domain of f_1 agreed on the first bit according to f, so the codewords of f_1 still disagree on at least d bits. From these facts it is clear that f_1 is a $(n-1, k-1, d)$ code. If we repeat the puncturing $n - d$ times, we obtain a $(d, k-(n-d), d)$ code given by f_{n-d}, so its rate is $r = \frac{d}{d} = 1$. Recall from the introduction that such a code can have at most $|\Sigma|$ codewords, meaning that the domain of f_{n-d} can be desribed by a single bit, and we conclude that $k - (n - d) \leq 1$. If we rearrange and divide by n, we get

$$\frac{k}{n} = r \leq 1 - \delta + o(1) \tag{1}$$

As $n \to \infty$, (1) gives the desired inequality. $\square$

Notice how the proof worked: we started with an arbitrary code and iteratively punctured it until we reached a point that it could not be punctured any more. This can be done in general whenever we have an inequality for r, and we will end up with a line for the upper bound. This fact is used again in the proof of the Plotkin bound.

2.1 Reed-Solomon code

The Reed-Solomon code is a code for which equation (1) is an equality.

Definition 3 (Reed-Solomon code)**.** *Let $\mathbb{F}$ be a finite field. For $x_1, ..., x_k \in \mathbb{F}$ define the Reed-Solomon encoding of $(x_1, ..., x_k)$ to be:*

$$RS(x_1, ..., x_k) = (p(y_1), p(y_2), ..., p(y_n))$$

where p is the polynomial over $\mathbb{F}$ of degree at most $k-1$ obtained by interpreting $x_1, \ldots, x_k$ as coefficients, and $y_1, ..., y_n$ are all of the field elements (i.e. $n = |\mathbb{F}|$).

This code is based on the fact that 2 distinct degree j polynomials over $\mathbb{F}$ can agree in at most j points. This means that 2 distinct inputs to RS will map to codewords that are different on at least $n-(k-1)$ letters. Hence the RS code is a (n, k, d) where $d = n-(k-1)$, $r = 1 - \delta + \frac{1}{n}$ and equation (1) is an equality.

This would seem to indicate that the Reed-Solomon code can choose k and n to lie anywhere on the line $R(\delta) \leq 1 - \delta$. This is not true because the Reed-Solomon code is very poor when viewed as a binary code. To do this, we pick $\mathbb{F} = GF(2^a)$, and then each x_i and $p(y_i)$ are a bits corresponding to an element of the field. If $p(y_i) \neq p(y_j)$, this only tells us that at least one of the bits are different, so this gives an extra $\log(n)$ factor in the relative distance of the code. The relative distance of the code when viewed as a binary code is $\frac{1-\frac{k-1}{n}}{\log(n)}$, which goes to 0 as $n \to \infty$. Hence the Reed-Solomon code lies at the point $(\delta, R(\delta)) = (0, 1)$.

3 Hamming bound

The next upper bound we prove is based off of a volume argument by considering the Hamming balls centered at each codeword.

Theorem 2 (Hamming bound)**.** *$R(\delta) \leq 1 - H(\frac{\delta}{2})$.*

Proof. Consider balls in $\{0,1\}^n$ with radius $\lfloor \frac{d-1}{2} \rfloor$ and centered on codeword x (where $B(x, s)$ contains all elements of $\{0,1\}^n$ that differ with x in at most s bit positions). For any (n, k, d) code, the balls for different codewords must be disjoint (because of our choice of radius). Because the balls are disjoint, if we the number of points in the union is the same as the sum over all balls of the number of points in each ball. Because the union of all the balls can at most cover the space, we have

$$2^k VOL(B(\cdot, \lfloor \frac{d-1}{2} \rfloor)) \leq 2^n. \tag{2}$$

Recall from the last lecture that $VOL(B(x, \lfloor \frac{d-1}{2} \rfloor)) \leq 2^{H(\lfloor \frac{d-1}{2} \rfloor)} = 2^{H(\frac{\delta}{2} n)}$. When we plug this in, take the logarithm and divide by n, we get

$$r + H(\frac{\delta}{2}) \leq 1.$$

□

3.1 Hamming code

The Hamming code is a linear code for which equation (2) is an equality.

Definition 4 (Hamming code)**.** *For $\ell > 0$, let Ham be the $\ell \times 2^\ell - 1$ boolean matrix that has columns equal to all binary strings of length ℓ that are not all 0. Ham is also referred to as the parity check matrix.*

For a vector x of length $2^\ell - 1$, x is a codeword if and only if the product $Ham \cdot x$ is all zero.

We have defined the code by using the parity check matrix. Note that because this is a linear code, there is some matrix G that would be used for encoding. The special property of this code (as compared to general linear codes) is that there is an efficient way to decode (the Ham matrix is used to trivially detect errors, something slightly more sophisticated can be done to correct errors).

Notice that Ham has full rank because it contains an $\ell \times \ell$ sub-matrix with full rank (for example the identity on ℓ rows is contained in Ham). Thus, the set of codewords that Ham maps to 0 has size $2^\ell - 1 - \ell$. Recall that for a linear code, the distance of the code is the smallest weight of a non-zero codeword. The smallest weight non-zero x with $Ham \cdot x = 0$ is equal to the smallest number of columns in Ham that are linearly dependent, which is 3 (for example, take the first 3 columns that correspond to the binary expansions of 1, 2, and 3).

Therefore, the Hamming code is a $(n, k, d) = (2^\ell - 1, 2^\ell - 1 - \ell, 3)$ code, and in particular, $d = 3$ means that it can only correct at most 1 error. For any codeword x, consider the value $VOL(B(x, \lfloor \frac{d-1}{2} \rfloor)) = VOL(B(x, 1))$. $B(x, 1)$ is the set of words that are at Hamming distance at most 1 from x, and is of size $n+1$. When this is plugged into equation (2), it gives an equality. The Hamming code is called a *perfect code* because it exactly partitions the space of possible codewords and doesn't waste any space. As with the Reed-Solomon code, the Hamming code is asymptotically at $(\delta, R(\delta)) = (0, 1)$.

4 Plotkin bound

Our next bound shows that in particular, no any code with $\delta = \frac{1}{2}$ must have $r = 0$, which matches the lower bound given by Gilbert-Varshamov.

Theorem 3 (Plotkin bound)**.** $R(\delta) \leq 1 - \frac{\delta}{2}$.

Proof. We first show that the result follows given the following claim.

Claim 1. *Any binary code with $d \geq n/2$ has at most $2n$ codewords.*

For an (n, k, d)-code, the number of codewords is 2^k, so by Claim 1, we have $2^k \leq 2n$. Taking the logarithm and dividing by n yields that $\frac{k}{n} = r \leq \frac{\log(2n)}{n}$, meaning that $R(\delta) = 0$ when $\delta \geq \frac{1}{2}$.

As in the proof of the Singleton bound, puncturing an (n, k, d)-code $n - 2d$ times yields a $(2d, k - (n - 2d), d)$ code. Applying Claim 1 yields that $2^{k-(n-2d)} \leq 4n$. Taking the logarithm and dividing by n, we conclude that $r \leq 1 - \frac{\delta}{2} + \frac{\log(4d)}{n} = 1 - \frac{\delta}{2} + o(1)$. All that remains is to prove the claim.

Proof of claim: Suppose we have a code with $d \geq n/2$. We will embed the codewords into $\mathbb{R}^n$ by mapping each bit of a codeword so that $0 \to 1$ and $1 \to -1$. If we consider vectors for distinct codewords, the product of each bit will be 1 if they agree on that bit and -1 if they don't agree. Because we have $d \geq n/2$, $\langle x, y \rangle \leq 0$ for x and y corresponding to distinct codewords.

Using this observation, we can bound the number of codewords in our code by bounding the number of vectors in $\mathbb{R}^n$ with coordinates equal to 1 and -1, where each pair of vectors has non-positive inner product. We accomplish the latter by induction, showing that there are most $2n$ such vectors. The base case in $\mathbb{R}^2$ is trivial. Suppose we have more that $2n$ vectors in $\mathbb{R}^n$ that satisfy the conditions. Then there must be a coordinate that can be removed such the conditions will become unsatisfied on at most 2 of the vectors. This contradicts the induction hypothesis that we have more than $2(n-1)$ vectors in $\mathbb{R}^{n-1}$ with non-positive inner product. □

□

4.1 Hadamard code

The Hadamard code is a linear code for which Claim 1 is tight.

Definition 5 (Hadamard code)**.** *Given a binary string x of length ℓ, the Hadamard encoding of x is:*

$$Had(x) = \text{ inner product of } x \text{ with all possible strings of length } \ell.$$

Equivalently, we can define the Hadamard matrix Had, which is $2^\ell \times 2^\ell$ with entries in $\{1, -1\}$. Each row and column is indexed by an ℓ bit string over $\{0, 1\}$. We define $Had_{x,y} = (-1)^{\langle x,y \rangle}$, and the Hadamard code of x is the x^{th} row of Had.

Notice that the inner product of x with y will be one for exactly half of the ℓ bit strings y, so every codeword has $\frac{2^\ell}{2}$ ones. Recall that the distance of a linear code is the smallest Hamming weight of a non-zero codeword. Therefore, the Hadamard code is a $(n, k, d) = (2^\ell, \ell, \frac{2^\ell}{2})$ code. Notice that the relative distance stays $\frac{1}{2}$, but the rate goes to 0 as $n \to \infty$, so the Hadamard code lies at the point $(\delta, R(\delta)) = (\frac{1}{2}, 0)$.

However, we don't yet have an equality for Claim 1 since the number of codewords is 2^ℓ. This can be fixed by including the complement of each codeword produced by Had, resulting in twice as many codewords. Such a code achieves equality for Claim 1.

If we define G to be the matrix such that $G \cdot x = Had(x)$, notice that this matrix is related to the Hamming matrix. In particular, $G = Ham^T$ (i.e. the generator matrix of the Hadamard code is the transpose of the parity check matrix of the Hamming code). When this is true, we say that the two codes are *duals* of each other. Each linear code has both a generator matrix G and parity check matrix H. Then the dual of code $\{G, H\}$ is $\{H^T, G^T\}$. If the code $\{G, H\}$ maps k bits to n bit codes, then $\{H^T, G^T\}$ will map $n - k$ bits to n bit codes.

5 Elias bound

We conclude with a better bound that follows from a combinatoric argument which proceeds by computing the distance of the code in two different ways

Theorem 4 (Elias bound)**.** $R(\delta) \leq 1 - H(\frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\delta}{2}})$.

Proof. Let ρ be a parameter that we will optimize later, and x^* be the n bit word (not necessarily a codeword) such that the Hamming ball $B(x^*, \rho n)$ contains as many codewords as possible. Let C be the number of codewords in this ball. We first count C by deriving upper and lower bounds on the pairwise Hamming distance of all the codewords in the ball.

Because all of the codewords have distance at least d from each other, we have

$$\sum_{y \neq z \text{ codewords in } B(x^*, \rho n)} h(y,z) \geq C(C-1)d, \tag{3}$$

where $h(x,y)$ denotes the Hamming distance of x and y. Alternatively, we can compute the pairwise distance by summing over all bit positions rather than all strings. Let c_i be the fraction of codewords in the ball with $y_i \neq x_i^*$. This approach yields

$$\sum_{y \neq z \text{ codewords in } B(x^*, \rho n)} h(y,z) = \sum_{i=1}^{n} 2C \cdot c_i \cdot C(1-c_i). \tag{4}$$

Because the radius of the ball is ρn, we have

$$\sum_{i=1}^{n} c_i \leq \rho n.$$

Notice that equation (4) is maximal when each of the c_i are equal and as close to $\frac{1}{2}$ as possible (we have used this fact many times before), so an upper bound follows when $c_i = \rho$ for all i:

$$\Sigma_{i=1}^{n} 2C \cdot c_i \cdot C(1-c_i) \leq n \cdot 2C \cdot \rho(1-\rho). \tag{5}$$

Thus, we have derived both a lower bound (given by (3)) and an upper bound (given by (4) and (5)) on the pairwise Hamming distance of codewords in $B(x^*, \rho n)$. Together they result in the condition $C \cdot 2\rho(1-\rho) \geq (C-1)\delta$. Rearranging gives us

$$\underbrace{[\delta - 2\rho(1-\rho)]}_{(**)} C \leq \delta.$$

When $(**)$ is positive, this implies C is some constant depending on ρ and δ.

Let us count C in a different way. We know the average number of codewords in a ball is equal to the size of the ball times the probability that a random element is a codeword. Since C is maximal over all balls, this gives us

$$C \geq VOL(B(\cdot, \rho n)) \cdot \Pr[\text{random word is a codeword}] \geq 2^{H(\rho)n} \cdot \frac{2^k}{2^n} = 2^{\overbrace{(H(\rho) + r - 1)}^{(***)} n}.$$

If $(***)$ is positive, then C goes to ∞ as $n \to \infty$. If $(**)$ also holds, this contradicts the requirement that C is a constant. Therefore, as long as $(**) > 0$, it must be true that $H(\rho) + r - 1 \leq 0$. By optimizing ρ to ensure $(**) > 0$, we get $\rho = \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\delta}{2}}$ which in turn gives the desired bound on the rate. $\square$

6 Concluding remarks

We point out that there are better bounds than we have seen (for example the Linear Programming bound), but as they generally require more background and/or effort to prove, we do not present them here. Also, note that all of the codes we have given as examples in this lecture have either rate 0 or relative distance 0 asymptotically, so we have not yet seen a *good* code. The Justensen code was the first *good* code in the sense we have defined. In the next lecture, we will see a construction of a *good* code that uses expanders. This code will have the advantage of also being linear time encodable and decodable.