6.895 Essential Coding Theory	September 8, 2004

Lecture 1

Lecturer: Madhu Sudan — *Scribe: Piotr Mitros*

1 Administrative

`Madhu Sudan`
`madhu@mit.edu`
`http://theory.lcs.mit.edu/~madhu/FT04`

To do:

- Sign up for scribing – everyone must scribe, even listeners. Send e-mail to MADHU@MIT.EDU
- Get added to mailing list
- Look at problem set 1. Part 1 due in 1 week.

2 Overview of Class

Historical overview. The field started by a mathematician, Hamming (1940s-1950). Hamming was looking at magnetic storage devices, where data was stored in 32 bit chunks, but bits would occasionally be flipped. He was investigating how we could do something better. There are several possible solutions:

Naive solution: Replicate everything twice. Will correct a one bit error, but for every 3 bits, we only have 1 real bit, so the rate of usage is only $\frac{1}{3}$.

Better solution: Divide everything into 7 bit blocks. In each block, store 4 real bits such that for any one bit flipped, we can figure out which bit was flipped. Hamming did this with the following encoding process:

$$G = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right)$$

Encoding:

$$(b_1, b_2, b_3, b_4) \longrightarrow (b_1, b_2, b_3, b_4) \cdot G$$

Here, the multiplication is over F_2.

Claim: If $a, b \in \{0, 1\}, a \neq b$ then $a \cdot G$ and $b \cdot G$ differ in ≥ 3 coordinates.

This implies that we can correct any one bit error, since with a one bit error, we will be one bit away from the correct string, and at least 2 bits away from the incorrect string. Note we have not proven this claim yet.

We can verify the claim by enumerating over 16 possibilities, but this gives no insight. What's interesting is how Hamming arrived at G. He showed:

$$\exists H \text{ (matrix) s.t. } G \cdot H = 0 \text{ (See pset 1)}$$

$$H = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ & \vdots & \\ 1 & 1 & 1 \end{pmatrix}$$

$$\left\{xG|x \in \{0,1\}^4\right\} = \{y|yH = 0\}$$

How few coordinates can $y = aG$ and $z = bG$ disagree on? Equivalent to asking same question when $yH = zH = 0$.

We can weaken it by asking how few coordinates can $x = y - z$ be non-zero on, given that $xH = 0$?

We'll make <u>subclaim 1</u>: If x has only 1 non-zero entry, then $x \cdot H \neq 0$.

We'll make <u>subclaim 2</u>: If x has only 2 non-zero entries, then $x \cdot H \neq 0$.

Subclaim 1 is easy to verify. If we have exactly one non-zero value, $x \cdot H$ is just a row of H. All of the rows of H are non-zero, so $x \cdot H \neq 0$. Claim 2 is almost as easy — here, $x \cdot H \neq 0$ is equal to the XOR of two rows of H. No two rows of H are equal, however, so the XOR of any two rows will be non-zero. Hence, $x \cdot H \neq 0$.

We can do the same thing with larger H matrices. H must have non-zero, unique rows, and as few columns as possible. We can build it with 31 rows, 5 columns. This is just the enumeration of all non-zero 5 bit integers. We call this H_5. From here, we need to figure out what kind of G_5 it has.

For $H_5 \exists G_5$ that has 31 columns, $(31 - 5) = 26$ rows, and:

$$\left\{xG_5|x \in \{0,1\}^{26}\right\} = \{y|yH_5 = 0\}$$

G_5 also has full column rank, so it maps bit strings uniquely.

Note that our efficiency is now $\frac{26}{31}$, so we're asymptotically approaching 1. In general, we can encode $n - \log_2(n+1)$ bits to n bits, correcting 1 bit errors.

2.1 Decoding

We can figure out if an error happened by verifying whether $y \cdot H = 0$. If we have an error, we need to determine which bit was flipped. We could brute-force finding this error by trying to flip each bit until $y \cdot H = 0$. We can, however, do better. Let:

$$y = c + e_i$$

Where c is the original code, and e_i is the error vector. Then, notice:

$$y \cdot H = (c + e_i) \cdot H = c \cdot H + e_i \cdot H = 0 + e_i \cdot H = e_i \cdot H$$

This is simply the ith row of H. Note that, as Hamming defined H, the ith row of H is just the bit string for i. As a result, $y \cdot H$ directly returns the location of the bit in which there is the error.

2.2 Theoretical Bounds

2.2.1 Definitions

Define the **Hamming Distance** as $\Delta(x,y) = \sum_i x_i \ominus y_i$, or the number of bits by which x and y differ. Define a **ball** around string x of radius (integer) t as $B(x,t) = \{y \in \Sigma^n | \Delta(x,y) \leq t\}$, or the set of strings that differ by at most t bits.

We can describe a code in terms of the maximum radius we can assign to balls around all codewords such that the balls remain disjoint. A code C of distance $2t+1$ is t-error correcting, and vica-versa.

2.2.2 Hamming Bound

The idea is that we can show that the ball around each codeword occupies some volume of space. If we multiply that volume needed to correct by the number of codewords, we get the minimum amount of space needed for the code.

Definitions: $K = |C|$ is the number of words in the code. $k = \log_2 K$ is the number of bits in each unencoded word. $n \geq k$ is the number of bits in each encoded word.

$$K \cdot \mathrm{Vol}(t, n) \leq |\Sigma|^n$$

For binary and one-bit errors,

$$K(n+1) \leq 2^n$$

Taking log of both sides,

$$k \leq n - \log_2(n+1)$$

Notice that the 26 bit Hamming code is as good as possible:

$$31 - 5 = 26$$

3 Themes

Taking strings and writing them so they differ in many coordinates. This is called an **error correcting code**. In general, you have a finite alphabet Σ. Popular examples: $\{0, 1\}$, ASCII, some finite field. Σ^n is the set of all words. To correct errors, we pick a subset code $C \subset \Sigma^n$. Elements of C are called codewords. We would like to build codes with 2 nice properties:

1. We wish to build <u>large codes</u> (maximize $|C|$). We'd like to be able to encode as many messages as possible, and high rate of usage.

2. We'd like to have a <u>large minimum distance</u> so we can correct significant errors.

Impossible to maximize both simultaneously, so we run into problems of tradeoff.

Hamming has an implicit assumption of worst-case. Note that this is not necessarily the theory used in modern CD players, cell phones, etc. which are based on average case.

Goals of course:

- Construct nice codes. "Optimize large $|C|$ vs. large min distance"
- Nice encoding algorithms (computationally efficient — polynomial, linear, or sublinear time)
- Decoding algorithms
- Theoretical bounds/limitations
- Applications

Hamming's paper did all of the above — it described a code, an efficient encoding algorithm (polytime. *Challenge: Construct linear-time encoding algorithm for Hamming code. Probably possible in general case. Hopefully not too hard.*). It also described an elegant decoding algorithm, and gave a theoretical bound on the performance of the code.

4 Appendix on Algebra

A field is an algebraic structure that allows addition, subtraction, multiplication, and division by nonzero elements.