6.440 Essential Coding Theory March 10, 2008

Lecture 10

Lecturer: Madhu Sudan *Scribe: Jonathan Schneider*

1 Overview

Thus far in the course we have primarily been concerned with the *combinatorics* of error-correcting codes. We have presented several explicit examples of codes (e.g. Hamming codes and Reed-Solomon codes) and demonstrated bounds on what codes are achievable (for example, the Gilbert-Varshamov bound).

Today we talk about the *algorithmic* aspects of error-correcting codes; that is, how to efficiently implement these codes in polynomial time on a computer. There are four main algorithmic problems we will discuss.

- **Encoding**: Given a message m, determine which codeword we should transmit.
- **Error-Detecting**: Having received an encoded message y, determine whether any errors occured during transmission.
- **Erasure-Correcting**: Given a partially-erased (yet otherwise error-free) encoded message y (i.e., some characters in y are replaced by '?'), recover the original message.
- **Error-Correcting**: Having received an encoded message y (possibly containing errors), recover the original message m.

Since these problems are generally hard in the case of arbitrary codes (it is unclear how exactly to store the information in an 'arbitrary' code efficiently, for example), we will focus on linear codes. In all of the below examples, we will assume that our code is a linear code and that we are given the generator matrix G.

2 Encoding

Encoding a linear code is easy; given a message m, we simply send the codeword given by mG. This matrix multiplication takes $O(n^2)$ time.

For many classes of codes, we can do even better. For example, for Reed-Solomon codes, there exist encoders that can encode in $\tilde{O}(n)$ time (that is, n times something polylogarithmic in n). We can encode other classes of codes in linear time; we will see these later in the course.

3 Error-Detecting

Clearly, if there are enough errors, we cannot always detect whether an error has occured. However, if we are guaranteed that $\Delta(y, C) \leq d - 1$ where y is our received message, C is our set of codewords, and d is the distance of our code, then we can always determine whether $y \in C$.

One way to do this is simply to try to solve the linear system $xG = y$ for x; if it has a solution, then $y \in C$, and otherwise $y \notin C$. This takes $O(n^3)$ time to do naively. We can do better by computing the parity check matrix H (how to do so efficiently was described in Problem Set 1) and checking whether $Hy = 0$; this takes $O(n^2)$ time.

Again, for many classes of codes, we can get an almost linear runtime; for Reed-Solomon codes, there exists an error-detection algorithm that runs in $\tilde{O}(n)$ time.

4 Erasure-Correction

Our goal in erasure-correction for a linear code is the following. Given $y \in (\mathbb{F}_q \cup \{?\})^n$, we wish to find $x \in \mathbb{F}_q^k$ such that $(xG)_i = y_i \forall i$ such that $y_i \neq ?$. (Intuitively, we are replacing some of the characters in y by '?' and we wish to see whether we can still recover the original message).

We can once again solve this by solving the linear system $xG' = y'$ for a unique x, where G' and y' have columns erased at the locations of the '?' characters in y. This works for up to $d-1$ erasures, but it may work for even more depending on where the '?' characters are located and whether the corresponding columns of G are linearly independent.

5 Decoding

Unlike the previous three tasks, there is no simple linear-algebraic way to decode an arbitrary linear code. For this reason, decoding is a much more tricky and interesting problem.

Formally, given $y \in \mathbb{F}_q^n$ and some number of errors t, our goal is to find $x \in F_q^k$ such that $\Delta(xG, y) \leq t$. In theory, we should be able to support anywhere up to $t = \lfloor \frac{d-1}{2} \rfloor$ errors (from the Hamming bound, this is the maximum that is theoretically possible). This problem is known in the literature as the "Bounded-Error Decoding Problem".

Another related problem that we might wish to solve instead is the "Maximum Likelihood Problem" (or alternatively, the "Nearest Codeword Problem"). In this problem, given y, we wish to find a x which minimizes $\Delta(xG, y)$. (We can both consider the average-case complexity situation where the message is drawn from some distribution and the worst-case complexity situation where the codeword is adversarially chosen; the first problem is due to Shannon, and the second problem is due to Hamming).

Little is known about the hardness of these problems for arbitrary linear codes. Luckily, there are efficient algorithms that solve the above problems for classes of linear codes. In the remainder of this section, we're going to examine this problem for the case of Reed-Solomon codes.

5.1 Reed-Solomon Decoding Problem

For the case of Reed-Solomon codes, the decoding problem reduces to the following problem.

Definition 1 *Given a description of a Reed-Solomon code consisting of the field $\mathbb{F}_q$, degree k, and points $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ with $\alpha_i \in \mathbb{F}_q$, and an encoded message $(\beta_1, \beta_2, \ldots, \beta_n)$ with $\beta_i \in \mathbb{F}_q$, the* Reed-Solomon decoding problem *is to find a polynomial $M(x)$ such that $\deg M < k$ and if $S = \{i | M(\alpha_i) \neq \beta_i\}$, then $|S| \leq t$.*

5.2 History of Reed-Solomon Decoding

Before we start giving our description of our Reed-Solomon Decoding algorithm, we present a few notes on the history of this problem and how it was solved.

BCH codes, despite being a specific case of Reed-Solomon codes, were actually invented first (in 1959). Reed-Solomon codes were invented a year later in 1960. At the same time, Peterson gave an $O(n^3)$ algorithm for solving the decoding problem for BCH codes; this was a particularly remarkable achievement seeing as, at that of history, few people were inclined to search for particularly efficient algorithms (our current notions of polynomial time as a desired algorithmic complexity class stem from work by Edmonds and Cobham in 1965 and 1966).

Three years later, in 1963, Gorenstein and Zievler noticed that BCH codes are a specific case of Reed-Solomon codes. By observing this, they were also able to transform Peterson's algorithm for decoding BCH codes into an algorithm for decoding Reed-Solomon codes.

The next result came in 1972, when Berlekamp and Massey presented an $O(n^2)$ time algorithm for this problem; this is probably the most cited and best known algorithm for decoding Reed-Solomon codes. Berlekamp and Welch later simplified this to an $O(nt)$ time algorithm, where t was the actual number of errors that occurred.

Finally, in 1992, a paper by Gemmell and Sudan presented a new exposition of this algorithm. This is the version we will follow in the following sections.

5.3 Reed-Solomon Decoding Algorithm

We could ostensibly decode a message by enumerating over all possible sets of indices where $M(\alpha_i) \neq \beta_i$ and performing polynomial interpolation for each of these sets. However, if there are up to t errors, this set could have size up to $O(q^t)$, and this method is therefore much too slow. Ideally, we need some method of quickly determining this set of indices.

One way to encapsulate this intuition is via the idea of an "Error Locator Polynomial". Consider a nonzero polynomial $E(x)$ that satisfies the following two conditions:

- For each i, either $E(\alpha_i) = 0$ or $M(\alpha_i) = \beta_i$.
- The degree of E is as small as possible.

Note that, the degree of E is minimized when it equals $|S|$, since $E(x) = \prod_{i \in S}(x - \alpha_i)$. In particular, $\deg E \leq t$.

Now, for all i, either $M(\alpha_i) - \beta_i = 0$, or $E(\alpha_i) = 0$. This implies that $E(\alpha_i)(M(\alpha_i) - \beta_i) = 0$. Therefore, $E(\alpha_i)M(\alpha_i) = \beta_i E(\alpha_i)$. Write $N(\alpha_i) = E(\alpha_i)M(\alpha_i)$.

We have therefore reduced the problem of finding M to the problem of finding three polynomials M, E, and N that satisfy the following conditions:

- $\deg M < k$
- $\deg E \leq t$
- $\deg N < k + t$
- $E \neq 0$
- For all i, $N(\alpha_i) = E(\alpha_i)M(\alpha_i) = \beta_i E(\alpha_i)$.

This problem might not look simpler at first glance, but we can make the key observation that we can rewrite these constraints as a linear system in the coefficients of N and E. This therefore gives rise to the following algorithm idea: solve this linear system for the coefficients of N and E, and then compute $M = N/E$. Specifically, we perform the following steps.

1. Find $(N, E) \neq (0, 0)$ such that for all i, $N(\alpha_i) - \beta_i E(\alpha_i) = 0$ by solving the associated linear system.
2. If N/E is a polynomial of degree less than k, then output $\hat{M} = \frac{N}{E}$ (proved that $|S| \leq t$).

Is this correct? There are a few things that could possibly go wrong:

- No such pair (N, E) might exist.
- Many such pairs (N, E) might exist.
- Our output $\hat{M}$ might not equal the original message M.

We will show that we need not worry about any of these possibilities. We first show that such a pair (N, E) must exist.

Lemma 2 *There exists a pair $(N, E) \neq (0, 0)$ such that for all i, $N(\alpha_i) - \beta_i E(\alpha_i) = 0$.*

Proof If $|S| \leq t$ (which is one of our assumptions), then the error-locator polynomial $E(x) = \prod_{i \in S}(x - \alpha_i)$ gives rise to such a pair (N, E). ■

We next show that, even though multiple pairs (N, E) might exist, they all give rise to the same value of $\hat{M}$.

Lemma 3 *Suppose* (N_1, E_1) *and* (N_2, E_2) *are both valid nonzero pairs. Then* $N_1(x)E_2(x) = N_2(x)E_1(x)$ *provided that* $k + 2t < n$.

Proof For all i in $[n]$, we have that

$$N_1(\alpha_i)E_2(\alpha_i) = E_1(\alpha_i)\beta_i E_2(\alpha_i) = E_1(\alpha_i)N_2(\alpha_i)$$

Note that since E and N have degrees at most t and $k + t - 1$ respectively, both $N_1(x)E_2(x)$ and $N_2(x)E_1(x)$ have degree at most $k + 2t - 1$. Since these two polynomials agree on $n \geq (k + 2t - 1) + 1$ points, they must be equal. ■

Remark Here we have used the fact that

$$\deg PQ \leq \deg P + \deg Q$$

This is essentially the key algebraic fact we are using in this algorithm.

It now follows from the above two lemmas that $\hat{M}$ must equal M; we know that M is attainable by the first lemma, and by the second lemma we know that only one possible value of $\hat{M}$ is attainable. This proves the correctness of this algorithm.

5.4 Correcting Erasures and Errors

We know that, for Reed-Solomon Codes, we can tolerate up to $d - 1$ erasures and up to $\frac{d-1}{2}$ errors. What about combinations of erasures and errors? We present the following fact without proof.

Theorem 4 *If when using a Reed-Solomon code, we have* s *erasures and* t *errors, with* $s + 2t < d$, *we can correct them in polynomial time.*

5.5 Generalization

As we mentioned above, the key algebraic result that our algorithm relied on is the fact that the degree of the product of two polynomials grows additively instead of multiplicatively (as it often does in other instances of products).

With this as motivation, we can present the following abstraction of our Reed-Solomon code result. For $u, v \in \mathbb{F}^n$, define the coordinate wise product $u * v$ be defined as $(u_1v_1, u_2v_2, \ldots, u_nv_n)$. For subsets S and T of $\mathbb{F}^n$, let $S * T$ be the set $\{u * v | u \in S, v \in T\}$.

Now, if S has dimension k and T has dimension t, then $S * T$ will generically have dimension kt. However, in the case that S corresponds to a Reed-Solomon code of dimension k and T has a Reed-Solomon Code of dimension t, then $S * T$ has dimension at most $k + t$.

We can therefore consider the following abstract decoding problem.

Definition 5 *Given a linear code* $C = [n, k, d]$, $(\mathcal{E}, \mathcal{N})$ *form a* t*-error locating pair if*

- *The distances* $\Delta(C)$ *and* $\Delta(\mathcal{N})$ *are both sufficiently large (how large is left as an exercise to the reader)*
- $\mathcal{E}$ *is a linear code with* $\dim(\mathcal{E}) > t$
- $\mathcal{E} * C \subset N$

A straightforward variant on our algorithm presented above can solve this abstract decoding problem. This yields decoding algorithms for all algebraic codes and corrects roughly $d/2$ errors (i.e., the best possible)..