6.897 Algorithmic Introduction to Coding Theory	October 22, 2001
Lecture 10	
Lecturer: Madhu Sudan	*Scribe: Aram Harrow*

Today we move on to the second phase of our course: Algorithms in Coding theory. We will introduce some of the algorithmic tasks associated with coding theory. We'll then move on to a specific algorithm for decoding Reed-Solomon codes.

1 Algorithmic Tasks

1.1 Algorithmic tasks from Shannon's theory

Shannon's theorem already introduced specified two basic algorithmic tasks, namely *Encoding* and *Decoding*. The final objective is only defined as a combination of the two, given a channel that fixes the error model. Specifically the Shannon algorithmic challenge, restricted to the binary symmetric channel is the following:

> **The Shannon Problem:** Let $D_{p,n}$ be the distribution on n independently chosen random bits where each bit is 1 with probability p and 0 with probability $1-p$. Given $R < 1 - H(p)$, find an *efficiently computable* family of functions $\{E_n\}_n$ and $\{D_n\}_n$ where $E_n : \{0,1\}^{Rn} \to \{0,1\}^n$ and $D_n : \{0,1\}^n \to \{0,1\}^{Rn}$ so as to maximize the error exponent, i.e.,
>
> $$\lim_{n\to\infty} \left\{ -\frac{1}{n} \log \left(\Pr_{m \leftarrow U_{Rn}, \eta \leftarrow D_n} [D_n(E_n(m) + \eta) = m] \right) \right\}.$$

In the above challenge one may plug in any notion of efficiency one feels comfortable with. We will use the notion of "polynomial time computability" for starters. Later, we will switch to more efficient notions such as linear time computability.

Unfortunately this challenge is too complex to handle all at once. The Hamming setup provides a nice modular breakdown of the task. To motivate this, suppose we introduced the Hamming version of the problem, where an *adversary* is allowed to choose the message and introduce errors, up to a specified limit t. Then the above problem transforms to:

> **The Hamming Problem:** Given $R > 0$ find the largest τ and an efficiently computable family of functions $\{E_n\}_n$ and $\{D_n\}_n$ where $E_n : \{0,1\}^{Rn} \to \{0,1\}^n$ and $D_n : \{0,1\}^n \to \{0,1\}^{Rn}$ such that:
>
> $$\forall \eta \in B(\mathbf{0}, \tau n), \quad D_n(E_n(m) + \eta) = m.$$

In this setup one can decouple, at least partially, two steps in the goal: First the image of the encoding function better be a good error-correcting code. Next, this error-correcting code better have a good decoding algorithm. Thus by fixing the intermediate point — namely, the error correcting code, we get two independent problems, whose goals can be decoupled. Still the task of defining the problems precisely is riddled with subtleties. So we will do so slowly.

1.2 Encoding

What could be be subtle about the following question? "Given a family of codes $\mathcal{C} = \{C_n\}_n$ with C_n being an $(n, k, d)_q$ code, find an efficient family of algorithms $E_n : \Sigma^k \to \Sigma^n$ so that the image of E_n

equals C_n." This definition works for most purposes, and we will be happy with it. However it will be good to recognize that there are issues here that might involve choices.

The main issue, stated in "data structural language", is the distinction between *preprocessing* and *querying*. Preprocessing is the work we do during the design phase of the code. Query processing is the work we do to encode a given message, with all the help that the preprocessing stage may have given.

The goals of preprocessing are often misunderstood. Unless one pins down the family of codes quite succinctly *efficiency* with respect to this task may not make a lot of sense. (An example of a question that does not make sense given what we know is: "How fast can the generator matrix of an algebraic-geometry code be constructed?" We know some specific families for which this question is well-defined — but in general, how do we even represent the algebraic geometry code whose generator matrix is to be constructed.)

The issue that does make sense to focus on, is to ask that the output of the preprocessing stage be succinct. If the code is linear, then a choice for the output is obvious: Output the generator matrix (or almost equivalently, the parity check matrix). But a more satisfactory answer, applicable to all codes and not just the linear ones, is to output a circuit describing the query processing function totally constructively. (Such a circuit would be composed of binary logic gates hooked up as an acyclic digraph with k input wires and n output wires. Computing the output of the circuit, given a fixed input takes time linear in the size of the circuit.)

What are the right goals for the "query processing" phase? The goal is fairly obvious here: We would like the encoding of the message to be performed as efficiently as possible. For linear codes, if we force the preprocessing phase to output a generator matrix, then the query processing is just a matrix-vector product and can be computed in polynomial (specifically $O(n^2)$) time. Later in the course, we will focus on the task of doing this more efficiently — but for now, we will consider this good enough. On the other hand, if the preprocessing phase outputs an encoding circuit, then the size of the output of the first phase *is* the running time of the query processing phase.

We conclude with the following two algorithmic tasks:

> **Preprocessing Problem:** For a fixed family of codes $\mathcal{C}$, given an index i, compute an encoding circuit (or the generator matrix) for the ith code of the family.

A weak requirement is that the above produce an output that has size that is a polynomial in n_i — the block length of the ith code. A stronger requirement is that the above process take time polynomial in n_i. The good news here is that for linear codes the weak requirement can always be fulfilled.

The second algorithmic task, corresponding to the "query phase" is:

> **Encoding Problem:** Given an encoding circuit E (or the generator matrix) for the ith member of a family of codes $\mathcal{C}$, and a message $\mathbf{m}$, compute its encoding.

Of course, in this sense, the encoding problem is trivial to solve in linear time in the size of E.

As mentioned earlier, we won't dwell on the subtleties introduced in this section during this course. But they ought to be kept in mind, when designing and evaluating the "utility" of new codes.

1.3 Decoding

Informally, the problem to be formalized is: Given a corrupted version $\mathbf{r} \in \mathbb{F}_q^n$ of an encoding of message $\mathbf{m}$, compute $\mathbf{m}$. However the problem is not well-posed yet, since we haven't quite given a formal definition of $\mathbf{m}$. Another issue is the specification of the encoding function E. When is this specified? And how much time are we given to design the decoding algorithm?

For the first issue: How is $\mathbf{m}$ defined as a function of $\mathbf{r}$, the natural definition based on the Shannon challenge described above, is the *maximum likelihood vector*, i.e., the vector $\mathbf{m}$ for which the probability that the received vector is $\mathbf{r}$ is largest, given the probabilistic channel. This motivates the following problem:

> **Maximum Likelihood Decoding**: Given a channel Channel corrupting strings in $\mathbb{F}_q^n \to \mathbb{F}_q^n$, an encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$, and a received vector $\mathbf{r} \in \mathbb{F}_q^n$ find $\mathbf{m} \in \mathbb{F}_q^k$ that maximizes $\Pr_{\text{Channel}}[\mathbf{r} = \text{Channel}(E(\mathbf{m}))]$.

When the channel noise model is that of the q-ary symmetric channel, then this question simplifies to the Hamming problem below:

> **Nearest Codeword Problem**: Given an encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$, and a received vector $\mathbf{r} \in \mathbb{F}_q^n$ find $\mathbf{m} \in \mathbb{F}_q^k$ that minimizes $\Delta(\mathbf{r}, E(\mathbf{m}))$.

The problems above are both quite hard. On the one hand, they don't allow the decoding algorithm to preprocess the code to be decoded. Further, they ask for error-correction possibly well beyond the distance of the code. So it should not be a surprise that versions of these problems become NP-hard. In particular, Berlekamp, McEliece, and van Tilborg [4] showed that the *Nearest Codeword Problem* above is NP-hard, even when the encoding functions are restricted to be linear functions given by their generator matrix. Later, Bruck and Naor [5] showed that one could even fix the family of codes, and thus the decoding algorithm could be allowed to preprocess the code, and the problem remains hard (unless NP has polynomial sized circuits).

Yet neither of these results are sufficiently negative to put a damper on the goal of meeting Shannon's challenge algorithmically. First - they only rule out decoding for some codes (and the codes are not particularly nice). Furthermore, they only rule out decoding these codes when the number of errors is much more than the distance of the code. To get a sense of the positive results in coding theory, we should look at some of the more reasonable decoding problems.

1.3.1 Reasonable decoding problems

The main restrictions that allow for algorithmic results are the following:

- We should only try to decode some fixed, well-known code (where we know the minimum distance etc.).
- We should place limits on the number of errors, and only expect to correct a number of errors in proportion to the distance.

The following three questions are natural questions along this line:

> **Unambiguous decoding:** For a fixed family of codes C, given an index i and a vector $\mathbf{r} \in \mathbb{F}_q^n$ find a codeword $\mathbf{c} \in C_i$, such that $\Delta(\mathbf{c}, \mathbf{r}) < \Delta(C_i)/2$ if such a codeword exists.

Note that the underlying assumption is that $\Delta(C_i)$ is known, or can be computed efficiently. Further, the question is posed so that the answer is unique and somehow reasonable to expect to compute, given the distance of the code. Actually the question remains reasonable for slightly larger values of the error, say up to the minimum distance of the code, though in such cases the answer is no longer unique or unambiguous. Here we can pose two slightly different questions. We call the first of these the "bounded distance decoding" problem, though this term is often used to allude to slightly related questions in the literature (and sometimes this is the *unambiguous decoding problem*).

> **Bounded distance decoding:** For a fixed family of codes C, and error function t, given an index i and a vector $\mathbf{r} \in \mathbb{F}_q^n$ and find *any* codeword $\mathbf{c} \in C_i$, such that $\Delta(\mathbf{c}, \mathbf{r}) \leq t(i)$ if such a codeword exists.

The bounded distance decoding problem is solvable for some codes for some functions $t(i)$ that are noticaebly larger than $\Delta(C_i)/2$. This is done by solving a slightly harder problem called the "list-decoding" problem described below. In principle, we would like to solve this problem for $t(i) = \Delta(C_i)$, but we know of no interesting codes where we can do this.

> **List decoding:** For a fixed family of codes C, and error function t, given an index i and a vector $\mathbf{r} \in \mathbb{F}_q^n$ and find a list of *all* codewords $\mathbf{c} \in C_i$, such that $\Delta(\mathbf{c}, \mathbf{r}) \leq t(i)$.

Both the bounded distance decoding problem and the list decoding problem are really families of problems (even once we fix the code $\mathcal{C}$) parametrized by the error function $t(\cdot)$. The larger this function for which an algorithm works, the better the algorithm.

In the sequel we will give an unambiguous decoding algorithm for Reed Solomon codes. But first we solve a problem that was too trivial to even pose above!

2 Erasure-decoding problem

Before we move on to the issue of handling actual *errors*, lets resolve one simple problem — that of decoding from erasures, at least in the case of linear codes. The erasure decoding problem is the following:

> **Erasure-decoding problem:**
> GIVEN: An $k \times n$ generator matrix $\mathbf{G}$ for a linear code C over $\mathbb{F}_q$ and a vector $\mathbf{r} \in (\mathbb{F}_q \cup \{?\})^n$, where ? is a special symbol denoting an erasure.
> FIND: A (or all) codeword $\mathbf{c} \in C$ satisfying $c_i = r_i$ for all $i \in [n]$ such that $r_i \neq ?$.

The algorithm for this part is quite simple. Suppose s symbols in $\mathbf{r}$ are erased. Let $\mathbf{r}'$ be the $(n-s)$-dimensional vector obtained by projecting $\mathbf{r}$ onto the non-erased symbols and let $\mathbf{G}'$ be the matrix obtained by projecting $\mathbf{G}$ onto columns corresponding to non-erasures. Let $\mathbf{m}$ be any solution to $\mathbf{m}\mathbf{G}' = \mathbf{r}'$ and let $\mathbf{c} = \mathbf{m}\mathbf{G}$. Then $\mathbf{c}$ is a solution to the erasure decoding problem.

When is this solution unique? We can't really look at the linear system and try to argue it can't have multiple solutions. So, we will use the properties of the code. Note that that if the code has minimum distance d, then $d-1$ erasures still allow for recovery of the codeword, information theoretically, since $d-1$ errors are detectable. In our case, this implies that if $s < d$, then the solution is unique.

Even if the solution is not unique, we don't have to give up hope. Since the solution to the erasure-decoding problem is obtained by solving some linear systems, we can actually enumerate all solutions to the erasure decoding problem. The set of solutions take the form $\{\mathbf{c} + \mathbf{b}\mathbf{M} | \mathbf{b} \in \mathbb{F}_q^t\}$ for some non-negative t, and some vector $\mathbf{c}$ and some $t \times n$ matrix $\mathbf{M}$. Furthermore the vector $\mathbf{c}$ and matrix $\mathbf{M}$ can be found efficiently. Thus the erasure decoding problem for linear codes can be solved as satisfactorily as one could hope for!

3 Unambiguous decoding for Reed-Solomon codes

Let us recall the unambiguous decoding problem for Reed-Solomon codes.

> **Reed-Solomon decoding:**
> GIVEN: n distinct elements $x_1, \ldots, x_n \in \mathbb{F}_q$, $y_1, \ldots, y_n \in \mathbb{F}_q$ and parameters k and $t \leq \frac{n-k}{2}$.
> FIND: A polynomial $p \in \mathbb{F}_q[x]$ of degree less than k, such that $p(x_i) \neq y_i$ for at most t values of $i \in [n]$.

The problem is quite a non-trivial one. At first glance the best we can say about it is that it is in NP — i.e., if such a polynomial exists, then this fact can be verified efficiently, *given* p. However it is not even clear that it lies in co-NP, i.e. given a received vector $\mathbf{y}$, is there a short proof that no polynomial p of degree less than k such that $p(x_i) \neq y_i$ for at most t values of i. Yet a polynomial time solution can be found for this problem. This solution dates back to 1960 when Peterson [10]. came up with an algorithm essentially solving this problem. Strictly speaking, the solution actually applied only to binary BCH codes. The decoding for the Reed-Solomon case is from the extension of Gorenstein and Zierler [7] who generalized BCH codes to the non-binary case and observed that the algorithm generalizes, and also that Reed-Solomon codes are special cases of BCH codes (in this incestuous world).

The Peterson-Gorentstein-Zierler algorithm actually ran in time $O(n^3)$. Later Berlekamp [2] and Massey [9] speeded this algorithm so that it ran in $O(n^2)$ time. Currently the fastest versions run in time $O(n \text{poly} \log n)$ time [8]. We will not give these faster algorithms, but actually give a simple algorithm, running in $O(n^3)$ time, due to Welch and Berlekamp [16, 3]. The actual exposition we give is from [6] (see also [14, Appendix A]).

3.1 Error-locating polynomial

The crux of the decoding algorithm is an object called the error-locating polynomial. We define this polynomial below. We warn the reader that, at least from its definition, this polynomial is not necessarily easier to find than the solution polynomial p. However its properties end up helping us.

Definition 1 (Error locating polynomial) *Given a vector* $\mathbf{x}$ *and a recieved vector* $\mathbf{r}$ *that is within a distance of* t *from some polynomial* p *of degree less than* k*, a polynomial* $E(x)$ *is called an* error-locating polynomial *if it has degree* t *and satisfies* $E(x_i) = 0$ *if* $p(x_i) \neq y_i$*.*

Note that such a polynomial always does exist: Simply take a set $T \subseteq [n]$ that contains all the error locations and satisfies $|T| = t$ and let $E(x) = \prod_{i \in T}(x - x_i)$.

Next, we define one more polynomial, that is also as hard to find as p, or E. This is the polynomial $N(x) \stackrel{\text{def}}{=} p(x) \cdot E(x)$. While each of these polynomials E, p and N is hard to find, together they are easier to find! Below we list some properties of the triple of polynmials p, E and N.

$$\left.\begin{array}{l} E(x_i) = 0 \text{ if } y_i \neq p(x_i) \text{ and } \deg_x(E) = t \\ N(x) = p(x)E(x) \text{ and } \deg_x(N) < k + t \\ \forall i \in [n] \quad N(x_i) = p(x_i)E(x_i) = y_i E(x_i) \end{array}\right\} \tag{1}$$

The last equation above might require some explanation. Note that if $E(x_i) = 0$, then $p(x_i)E(x_i) = y_i E(x_i) = 0$. When $E(x_i) \neq 0$, we know $p(x_i) = y_i$ and so we still have $p(x_i)E(x_i) = y_i E(x_i)$.

Now how do we find any of the above? Turns out, if we just ignore all references to p in Equation (1) and just look for E and N satisfying the remaining conditions (and this suffices to pin them down essentially uniquely)! We describe this precisely next.

3.2 The Welch-Berlekamp algorithm

We start by describing the major steps in the Welch-Berlekamp algorithm. A priori, it may not be clear as to why this algorithm is either correct, or efficient. We will argue this later.

Welch-Berlekamp Algorithm:

Given: $n, k, t \leq \frac{n-k}{2}$ and n pairs $\{(x_i.y_i)\}_{i=1}^n$ with x_i's distinct.

Step 1: Find polynomials N and E satisfying the following, if they exist:

$$\left.\begin{array}{l} \deg_x(E) = t \\ \deg_x(N) < k+t \\ \forall i \in [n] \quad N(x_i) = y_i E(x_i) \end{array}\right\} \tag{2}$$

Step 2: Output $N(x)/E(x)$.

3.2.1 Efficiency

We first note that the algorithm above can be implemented to run in polynomial ($O(n^3)$) time. The second step is obvious, and all we need to argue this for is the first step. However the first step is essentially just solving a linear system. Specifically, we need to find coefficients $\langle N_0, \ldots, N_{k+t-1}\rangle$ and $\langle E_0, \ldots, E_t\rangle$ such that for all i,

$$\sum_{j=0}^{k+t-1} N_j x_i^j = y_i \sum_{j=0}^{t} E_j x_i^j$$

Note that for every i, the constraint above is just a linear constraint in the unknowns. So solving the system above, just amounts to solving a linear system. The only catch is that we need $E_t \neq 0$ and this is not a linear constraint. But notice we can actually force $E_t = 1$ since otherwise we can just divide all coefficients by E_t to get an alternate solution which satisfies the requirements. So, finding a solution to Step 1 just amounts to solving a linear system and hence can be done efficiently.

3.2.2 Correctness

Next we argue the correctness. For this part, assume p is a polynomial of degree $< k$ that agrees with the given set of points at all but t points. In such a case we will show that a pair of polynomials N and E satisfying (2) do exist. However we will not be able to claim that there is a unique such pair. Instead we we will show that *any* pair of polynomials satisfying (refl10:eqn:two) have the same ratio (and this ratio equals p).

Claim 2 *There exists a pair of polynomials E and N satisfying (2) with the property that $N(x)/E(x) = p(x)$.*

Proof We just take E to be an error-locating polynomial for p and the given set of points, and let $N(x) = p(x)E(x)$. Then N and E satisfy all the requirements. ■

We now move to the more interesting claim. Note that it suffices to argue that $N/E = N'/E'$ for any pair of solutions to (2) since the claim above can then be used to see that this ratio must be p.

Claim 3 *Any solutions (N, E) and (N', E') to (2) satisfy*

$$\frac{N(x)}{E(x)} = \frac{N'(x)}{E'(x)}$$

or equivalently $N(x)E'(x) = N'(x)E(x)$.

Proof Note that the degrees of the polynomials $N(x)E'(x)$ and $N'(x)E(x)$ is at most $k - 1 + 2t$. Furthermore, from (2) we have, for every $i \in [n]$,

$$y_i E(x_i) = N(x_i) \text{ and } N'(x_i) = y_i E'(x_i).$$

Multiplying the two we get:

$$y_i E(x_i) N'(x_i) = y_i E'(x_i) N(x_i).$$

Now we claim that actually

$$E(x_i) N'(x_i) = E'(x_i) N(x_i).$$

This equality is obvious if $y_i \neq 0$, by cancellation. But why is it true is $y_i = 0$? Well, in such a case, we note (from (2) again) that $N(x_i) = N'(x_i) = 0$ and so again we have $E(x_i)N'(x_i) = E'(x_i)N(x_i) = 0$. Now if $n > k + 2t - 1$, then we get that the polynomials $N \cdot E'$ and $E \cdot N'$ agree on more points than their degree, and hence they are identical, as desired. ■

We summarize with the following theorem:

Theorem 4 *The* **Welch-Berlekamp algorithm** *solves the* **Unambiguous Reed-Solomon decoding problem** *in $O(n^3)$ time.*

As mentioned earlier, this is not the fastest possible algorithm for solving the unambiguous decoding problem. In fact, even the algorithm from [16] is faster and runs in time $O(n^2)$. If we look at the steps of the algorithm given here, one notes that all that the algorithm needs to be able to is solve a "rational function interpolation" problem and a "polynomial division" problem efficiently. Turns out both steps can be solved efficiently, in $O(n \text{poly} \log n)$ time. For the task of polynomial division, such an algorithm dates back to Sieveking [13]. The text by Aho, Hopcroft, and Ullman [1, Chapter 8] is an excellent source for material on this algorithm (and other fast algorithms for polynomial manipulation). The task of rational function interpolation on the other hand reduces to a "GCD-like" computation, which in turn was shown to be efficiently computable in the 1970s, by Schönhage [12]. Madhu's hastily written notes [15] and Michael Rosenblum's more careful writeup [11] fill in the details of the reduction and sketch the algorithm.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Anaylsis of Computer Algorithms.* Addison Wesley Publishing Company, 1974.

[2] Elwyn Berlekamp. *Algebraic Coding Theory.* McGraw Hill, New York, 1968.

[3] Elwyn Berlekamp. Bounded distance +1 soft-decision Reed-Solomon decoding. *IEEE Transactions on Information Theory*, 42(3):704–720, 1996.

[4] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978.

[5] Jehoshua Bruck and Moni Naor. The hardness of decoding linear codes with preprocessing. *IEEE Transactions on Information Theory*, 36(2), March 1990.

[6] Peter Gemmell and Madhu Sudan. Highly resilient correctors for multivariate polynomials. *Information Processing Letters*, 43(4):169–174, September 1992.

[7] Daniel Gorenstein and Neal Zierler. A class of error-correcting codes in p^m symbols. *Journal of the Society for Industrial and Applied Mathematics*, 9:207–214, June 1961.

[8] Jørn Justesen. On the complexity of decoding Reed-Solomon codes (corresp.). *IEEE Transactions on Information Theory*, 22(2):237–238, March 1976.

[9] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, January 1969.

[10] W. Wesley Peterson. Encoding and error-correction procedures for Bose-Chaudhuri codes. *IEEE Transactions on Information Theory*, 6:459–470, 1960.

[11] Michael Rosenblum. A fast algorithm for rational function approximations. Available from `http://theory.lcs.mit.edu/~madhu/FT01/notes/rosenblum.ps`, November 1999.

[12] Arnold Schönhage. Schnelle berechnung von ketterbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.

[13] Malte Sieveking. An algorithm for division of power series. *Computing*, 10:153–156, 1972.

[14] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, October 1992. Also appears as *Lecture Notes in Computer Science*, vol. 1001, Springer, 1996.

[15] Madhu Sudan. Notes on an efficient solution to the rational function interpolation problem. Available from `http://theory.lcs.mit.edu/~madhu/FT01/notes/rational.ps`, 1999.

[16] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic block codes. *US Patent Number 4,633,470*, December 1986.