6.440 Essential Coding Theory March 10, 2008

Lecture 10

Lecturer: Madhu Sudan *Scribe: Aleksander Mądry*

1 Today: Algorithmic problems

Today we will define the basic algorithmic problems arising in Coding Theory. We will also present decoding algorithm for Reed-Solomon codes.

2 Main algorithmic tasks in Coding Theory

There is a couple basic algorithmic tasks that we are performing in Coding Theory:

1. Encoding - having a message m we want to obtain its encoding c_m
2. Error-detection - having received some string y we would like to determine whether there occurred some error during transmission i.e. whether y corresponds to a codeword
3. Erasure-correction - assuming that some parts of transited string got erased (i.e. in respective positions we have '?' instead of an actual letter) we want to infer what were the letters erased
4. Error-correction - having received some string y we would like to determine what is (are, in case of list decoding) the codeword(s) whose transmission could occur in y

We say that we 'know' the actual code if we know how to perform all of the above tasks efficiently. Here efficiently means in polynomial time (later we will look sometime at linear time implementations of some codes) or more precisely, we would like to have a polynomial-time generator that for given length of the codeword will generate (small) circuits that perform the above task. Note that according to this requirement we don't 'know' the codes that correspond to Gilbert-Varshamov bound even if we know that they exist.

3 The case of linear codes

In case of linear codes the first three tasks turn out to be easy provided that we have a generator matrix $G \in \mathbb{F}^{k\times n}$.

3.1 Encoding

Having some message $m \in \mathbb{F}^k$ the encoding process consists of simple multiplication $y = m \cdot G$, then $y \in \mathbb{F}^n$ is the codeword corresponding to m. This is in contrast with the situation of the general, non-linear codes where neither of this task has a standard solution.

3.2 Error-detection

As it was already suggested in the Problem Set I, having G we can easily compute a matrix $H \in \mathbb{F}^{n \times (n-k)}$ such that $GH = 0$. Now, if $y \in \mathbb{F}^n$ is received string then $yH = 0$ iff y is a codeword. So, if the number of errors that can occur in the considered channel is not larger than $d - 1$, where d is the distance of considered code, then $yH = 0$ iff there was no error during transmission. In this way we obtain an efficient method for detecting errors.

3.3 Erasure-correction

To state the erasure-correction problem more precisely, let $y \in \{\mathbb{F}, ?\}^n$ be received string, where ? means that the corresponding letter has been erased (note that we assume that erasure is the only possible error here). We would like now to compute a message $m \in \mathbb{F}^k$ such that for all $1 \leq i \leq n$ either $y_i =?$ or $(mG)_i = y_i$ i.e. m is such that its encoding sent through the channel could result in y.

We solve this problem by reducing it to solving a linear system as follows. Let $I \subseteq [n]$ be the subset of indices of y that do not correspond to ?. We form a linear system:

$$x \cdot G_I = y_I$$

where x is the vector of k variables and G_I, y_I correspond to G, y respectively with removed columns being not in I. Clearly, any solution x^* to this system gives us a solution to original problem.

The question is: how many erasures can we handle when interested in determining uniquely the message m ?

First, we note that certainly not more than $d - 1$. The reason is that if we take a codeword c^* of the code that has minimal, but non-zero, weight then $w(c^*) = d$. So, since 0^k is a codeword in every linear code, by considering erasures that erase all the non-zero entries of c^* we see that then there will be at least two solution c^* and 0^k.

On the other hand, if our channel can cause at most $d - 1$ erasures then, upon transmitting an encoding of a codeword c, by definition of d, there is no other codeword c' whose encoding differs from encoding of c on at most $d - 1$ places. So, there is a unique (and thus correct) solution to the above linear system.

4 The decoding problem

We are focusing now on the task of error-correction. This tends to be the hardest of the problems arising in code design - even in case of linear codes there is no standard approach to it. We have two different definitions of decoding problem (we state them in the setting of linear codes, but they easily extend to the general case):

- (Shanon - average-case complexity): In this regime, given $y \in \mathbb{F}^n$ we would like to find $m \in \mathbb{F}^k$ such that $Pr[y|mG \text{ transmitted}]$ is maximal, where we assume here that the message m is chosen at random from some distribution. In other words, we want to solve the problem for 'most' y.
- (Hamming - worst-case complexity): Here we have, given $y \in \mathbb{F}^n$ find $m \in \mathbb{F}^k$ such that $\Delta(y, mG)$ is minimal. In other words, we want to be able to decode even if the pattern of errors is adversarial. Of course, in order to make the whole process decode the message that was actually sent, we need to have the number of errors to be no more than $\lfloor \frac{d}{2} \rfloor$. Sometimes, we consider a bit relaxed notion of decoding when we allow the algorithm to return a short list of possible messages (including the one that was actually sent) - then the number of errors can exceed this bound.

4.1 Decoding for Reed-Solomon codes

For the case of Reed-Solomon codes the task of decoding boils down to the following problem.

Given $C = (\mathbb{F}, \alpha_1, \ldots, \alpha_n, k)$ and $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$, find a polynomial $p(x) = \sum_{i=0}^{k-1} m_i x^i$ $(deg(p) \leq k-1)$ such that $\#$ errors $\equiv \{i | p(\alpha_i) \neq y_i\} \leq t$, where t is the maximal number of possible errors introduced by the channel. In our proof we assume that $t \leq \lfloor \frac{n-k}{2} \rfloor$ and recall that $n - k$ is the distance of Reed-Solomon code.

The first polynomial-time (in fact $O(n^3)$) algorithm for this problem was proposed by Peterson in 1960 - this is a remarkable fact since back then few people felt any inclination to look for 'efficient' algorithms. To be precise, Peterson algorithm was proposed for BCH codes, but a subsequent work of Gorenstein-Zierler (1963) showed that for proper choices of the field, the Reed-Solomon codes can be seen as both the generalization and a specialization of BCH codes. So, with this insight Peterson algorithm may be applied to Reed-Solomon codes. The next result in this topic is due to Berlekamp (1970). He showed an $O(n^2)$ algorithm, which was subsequently applied in other areas by Massey (so the algorithm is sometimes referred to as Berlekamp-Massey algorithm). In the early 80s Justesen proposed an $O(n \log^2 n)$ algorithm. Finally, in 1986 Welch and Berlekamp simplified significantly the Berlekamp algorithm and made its complexity to be $O(nt)$ where t is the actual number of errors that occurred. In 1992 a paper by Gemmell and Sudan, among other results, presented a nice exposition of the Reed-Solomon decoding algorithm, which we will follow now.

The key idea behind decoding the codes is so called *error-locating polynomial* $E(x)$. To define it, let us fix some $r \in F^n$ such that $\Delta(r, \langle p(\alpha_i)\rangle_{i=1}^n) \leq t \leq \lfloor \frac{n-k}{2} \rfloor$. Let $Err = \{i | p(\alpha_i) \neq r_i\}$. Then $E(x) \equiv \Pi_{i \in Err}(x - \alpha_i)$ and $deg(E(x)) \leq t$. We should note the following relations between $p(x)$ and $E(x)$.

1. For all i, $p(\alpha_i) = r_i$ or $i \in Err$. The latter is equivalent to saying that $E(\alpha_i) = 0$. So, we can conclude from this that for all i, $E(\alpha_i)p(\alpha_i) = E(\alpha_i)r_i$.
2. If we define $N(x) \equiv E(x)p(x)$ then $deg(N(x)) \leq k - 1 + t$
3. $E \neq 0$ and $deg(E(x)) \leq t$.

The problem with $E(x)$ and $p(x)$ is that we don't know neither of them. In fact, knowledge of one of them is equivalent to the knowledge of the other. So, the recipe for the decoding algorithm is just forgetting about $p(x)$ and trying to find any pair of polynomials that satisfy the above conditions when treated as $N(x)$ and $E(x)$ respectively. More precisely, the algorithm will look for a pair of polynomials $(N(x), E(x))$ such that

1. For all i, $N(\alpha_i) = E(\alpha_i)r_i$
2. $deg(N(x)) \leq k - 1 + t$
3. $E(x) \neq 0$ and $deg(E(x)) \leq t$.

Then, after finding such $(N(x), E(x))$ the algorithm outputs the result of $\frac{N(x)}{E(x)}$ if such exists. Otherwise, it just gives up.

First, we check whether this algorithm can be efficiently implemented. Division of polynomials can be easily performed in polynomial-time. On the other hand, computing the pair $(N(x), E(x))$ can be seen as a solving of a linear system, where we treat the coefficients of the polynomial as variables and impose an additional condition that $E(x) \neq 0$ (in fact we can just impose a constraint that the highest-order coefficient of $E(x)$ is 1).

Now, it remains to convince ourselves that such algorithm returns the correct answer. We start with the following lemma:

Lemma 1 *A pair $(N(x), E(x))$ satisfying all three condition and such that $\frac{N(x)}{E(x)} = p(x)$ exists.*

Proof

It is sufficient to take $E(x)$ as defined at the beginning and $N(x) = E(x)p(x)$. ■

It is worth noting that Lemma 1 guarantees existence, but not uniqueness. In fact, given any pair being a solution, if we multiply it by the same polynomial (whose degree is small enough to not violate the degree constraints that we want to impose on $(N(x), E(x))$) then we obtain another pair that satisfied the desired conditions. However, the next claim shows that any solution is good for us.

Claim 2 *If $(N_1(x), E_1(x))$ and $(N_2(x), E_2(x))$ are both satisfying all the required conditions then $\frac{N_1(x)}{E_1(x)} = \frac{N_2(x)}{E_2(x)}$*

Proof We will show that $N_1(x)E_2(x) = N_2(x)E_1(x)$ which is equivalent to desired statement. In order to do it, we note that $deg(E_2(x)N_1(x)) \leq t+k+t-1 = k+2t-1 < n$ by our assumption that $t \leq \lfloor \frac{n-k}{2} \rfloor$. By the same reasoning $deg(E_1(x)N_2(x)) < n$. Now, we notice that, by condition 2, for all $1 \leq i \leq n$, $N_1(\alpha_i)E_2(\alpha_i) = E_1(\alpha_i)E_2(\alpha_i)r_i = E_1(\alpha_i)N_2(\alpha_i)$. So, if these both products of polynomials are equal in n different points and have degree at most $n-1$, we can conclude, by the Fundamental Theorem of Algebra, their equality on all points. ■

This concludes the proof that the above algorithm works.

4.2 Power of algebraic approach?

As we saw in the previous section, the usage of algebraic methods leads to very simple, but powerful results. To, at least partially, explain the reason of this success we should note that the key property of polynomials that we used is the fact that the degree of the product of polynomials grows additively instead of growing multiplicatively as it is usually the case in the vector space settings. More precisely, if for vectors $u, v \in \mathbb{F}^n$ we define an operation $u \star v = (u_1v_1, \ldots, u_nv_n)$ and then we will generalize it to sets of vectors $S, T \subseteq \mathbb{F}^n$, $S \star T = \{u \star v | u \in S, v \in T\}$. We will notice that for two vector subspaces S, T of dimension k and l respectively, $dim(S \star T) \equiv dim(span(S \star T)) \leq k \cdot l$ and there are examples where this inequality is tight. However, if S and T correspond to Reed-Solomon codes with dimensions k and $t+1$ respectively, then, since its connection to polynomials, $dim(S \star T) \leq k+t$ only.