6.897 Algorithmic Introduction to Coding Theory September 25, 2002

Lecture 6

Lecturer: Madhu Sudan *Scribe: Vitaly Feldman*

1 Overview

Today's lecture will cover the following topics:

- Simplified review of Wozencraft ensemble
- Codes from other codes:
  - Parity, Puncturing, Restriction, Direct product, Subfield subcodes
  - Concatenation
- Forney codes
- What is "explicit"?
- Justesen codes

2 Wozencraft Ensemble Revisited

Let us consider the following special case of Wozencraft ensemble. The collection consists of codes $C_\alpha : \{0,1\}^k \to \{0,1\}^{2k}$. As defined in the previous lecture we think of $\{0,1\}^k$ as field $\mathbb{F}_{2^k}$, and for every $\alpha \in \mathbb{F}_{2^k}$ we define

$$C_\alpha : x \to (x, \alpha x) \ .$$

As we have seen, for $\alpha \neq \beta$, C_α and C_β do not share non-zero codewords (this follows from the fact that (x,y) can belong only to $C_{x^{-1}y}$). This means that there are at most $|B(\hat{0},d)| - 1$ α's which do not produce a code of distance d. In particular, there exists an α such that C_α has distance $d \approx 2kH^{-1}(1/2)$. Besides that, most of α's have distance at least $\frac{1}{2}d = \frac{1}{2}(2k)H^{-1}(\frac{1}{2})$. As can be seen from the proof, it is possible to construct the entire ensemble of the codes in time $2^{O(n)}$.

3 Codes Built from Other Codes

We are now interested in the idea of constructing new codes from the existing (atomic) ones (e.g., Hamming or Hadamart codes). It is often used when a code of a very specific length (say 353) is required for a particular application. In such situations we take a code produced by one of the known procedures and apply some operations to get a code of required length and with similar properties. Another application is in proving of lower bounds. By assuming the existence of code with certain parameters we can transform it into a code with parameters which are known to be impossible.

Let us examine several operations of this kind.

1. Parity check bit

   Given a code $C : \{0,1\}^k \to \{0,1\}^n$ we produce a code $C' : \{0,1\}^k \to \{0,1\}^{n+1}$ by adding a parity check bit to every codeword, that is $C'(x) = C(x) \cdot \oplus$ where $\oplus$ is the parity of all the bits in $C(x)$. As we have seen this construction transforms $(n,k,d)_2$ code to $(n+1,k,d+1)_2$ if d is odd. This operation unfortunately works only for odd d's and as can be seen from the following example such universal operation cannot exist for even d's. We have $[k+1,k,2]_2$ code (simple parity check) but there does not exist a $(k+2,k,3)_2$ code (Hamming bound).

2. Puncturing

   Puncturing is an operation that deletes one coordinate from codewords. It can be easily seen that such an operation transforms an $(n, k, d)_q$ code into an $(n-1, k, d-1)_q$ code. This operation is in some sense the opposite of the previous one, and is not very beneficial as we are usually interested in increasing distance/codeword length.

3. Restriction

   This operation is used when we want to reduce the codeword length without reducing the distance. The operation consists of choosing the most frequently appearing symbol in the first (or any other) coordinate (denote it by a) and then erasing the first coordinate and all the codewords which have the first symbol differing from a. The number of remaining codeword is at least $1/q$ of the original and the operation clearly preserves distance. That is, from an $(n, k, d)_q$ code the operation produces an $(n-1, k-1, d)_q$ code.

4. Direct product $C_1 \otimes C_2$

   Given two codes $C_1 = (n_1, k_1, d_1)_q$ and $C_2 = (n_2, k_2, d_2)_q$ we can generate an $(n_1 n_2, k_1 k_2, d_1 d_2)_q$ code. A message of length $k_1 k_2$ can be seen as a $k_1 \times k_2$ matrix. On each row of this matrix we apply C_2 thus forming a $k_1 \times n_2$ matrix. On each column of this result we apply C_1 to get an $n_1 \times n_2$ matrix representing a codeword of length $n_1 n_2$. The proof that the distance of this code is $d_1 d_2$ is left as an exercise. If this transformation is done "carefully" on linear codes then the composition is commutative, i.e., the same code would be obtained by first applying C_1 to columns and then applying C_2 to rows. This operation applied multiple times exponentially separates n from k and d and therefore cannot produce asymptotically good codes.

5. Sub-field (sub-alphabet) sub-codes

   This operation is used to try to get a code with a smaller alphabet from a code with a larger alphabet. Given alphabets $\Sigma' \subseteq \Sigma$ and a code $C \subseteq \Sigma^n$ we define $C' = C \cap \Sigma'^n$. This operation reduces the alphabet size from $q = |\Sigma|$ to $q' = |\Sigma'|$ and does not modify n and d. We cannot a priori know the number of codewords in C' (there could be none). Although this operation might seem to be not particularly useful, BCH codes can be seen as produced using this operation from Reed-Solomon codes. Besides that, this is the only operation so far that modifies (in particular, reduces) the alphabet size.

6. Concatenation $C_1 \cdot C_2$

   We start from two codes:

   outer: $C_1 = (N, K, D)_Q$ and
   inner: $C_2 = (n, k, d)_q$

   such that $Q = q^k$ so that the alphabet of the outer code is in one-to-one correspondence with the messages of the inner. The concatenation $C_1 \cdot C_2$ operates as follows. Given a message in[1] Q^K it first applies C_1 onto this message to get a string $s \in Q^N$. Then, on every symbol of s, viewed as a message in q^k it applies C_2 and concatenates the results, i.e., the resulting codeword is $C_2(s_1) \cdot C_2(s_2) \cdots C_2(s_N)$.

   The resulting code has the following parameters. Number of messages is $Q^k = (q^k)^k = q^{k^2}$. Length of codewords is obviously nN. Two distinct messages will differ in at least D positions after the application of C_1 and difference in each of these positions will result in d different positions after the application of C_2. Therefore the distance of the concatenation is at least dD. That is,

$$(N, K, D)_{q^k} \cdot (n, k, d)_q \Rightarrow (nN, kK, dD)_q \ .$$

[1]For brevity we denote both the alphabet and its size by the same letter Q or q.

This operation was invented by Forney[2] in 1966 and is the most common technique for constructing asymptotically good codes.

4 Forney Codes

The concatenation of codes gives us a way to use large-alphabet codes as an ingredient in our constructions. The only codes over a large alphabet we know are Reed-Solomon codes. Let us examine how RS codes can be utilized. We take an $[n, n/2, n/2]_n$ RS code (existing when n is a power of 2 for example). Our first attempt is concatenation with Hadamart code:

$$[n, n/2, n/2]_n \cdot [n, \log n, n/2]_2 \Rightarrow [n^2, \frac{n}{2}\log n, \frac{n^2}{4}]_2 \ .$$

The result has an asymptotically good distance but not an asymptotically good ratio. On the other hand, the codeword length of this code is polynomial in the message length. In a similar way, by concatenating $RS \cdot RS \cdot H$ we can get a $[k \log k, k, \frac{1}{8}k \log k]_2$ code (roughly). Therefore it is possible to reduce a "blow up" to a logarithmic factor. Ratio of Hadamart code vanishes to zero and therefore by concatenating with Hadamart code it is impossible to get asymptotically good codes. Therefore we will now try to "plug in" an asymptotically good code instead of Hadamart code to reduce the alphabet size. In the beginning of this lecture we saw an $[n, n/2, H^{-1}(1/2)2\log n]_2$ code (Wozencraft ensemble). By plugging it in we obtain:

$$[n, n/2, n/2]_n \cdot [2\log n, \log n, H^{-1}(1/2)2\log n]_2 \Rightarrow [2n\log n, \frac{n}{2}\log n, H^{-1}(1/2)\log n]_2 \ .$$

As we have seen in the beginning of this lecture, the code that we have plugged in is constructible in $2^{O(\log n)} = poly(n)$ time and therefore we have obtained an asymptotically good polynomially-constructible code.

Remark We did not show that a concatenation of two linear codes is linear. The proof of this fact requires n of particular form and a careful choice of representation of inner and outer alphabets. It is left as an exercise.

5 What is "explicit"?

In the beginning of this lecture we promised explicit asymptotically good codes, while Forney's construction has a "not-so-explicit" procedure of searching in polynomial-size space. This raises a "philosophical" question of what constructions can be considered "explicit". Something explicit is something we *know* and *knowing* for computer scientists means computing. Therefore we should look at the computational aspects of code construction. The computational requirements of Forney's construction are somewhat high and there are other computational requirements which will force a code to look more explicit. Let us examine the ways to define explicitness. For simplicity we assume a linear code and we consider the complexity of constructing its generator matrix G.

- Computable
- Polynomial time constructible (here we are)
- Logarithmic space constructible (does not allow brute force verification that a code has a given distance)

[2]Forney is an MIT EECS faculty member and teaches a related course on information theory and coding (6.451).

- Locally polynomially computable. Implicit construction of generator matrix s.t. for every indices i and j, $G_{i,j}$ is computable in time polynomial in $|i|$ and $|j|$. This definition is obviously very stringent. Can be made even more stringent by requiring log-space computation instead of polynomial-time.

For non-linear codes we can consider complexity of construction of encoding algorithm. In particular, we could define "explicit" to require polytime(or logspace) construction of encoding circuit.

6 Justesen Codes

The idea of concatenation could be used to construct more explicit codes than Forney codes. Simple idea that we are going to demonstrate is due to Justesen. In Forney construction we searched for an asymptotically good code in a space of polynomial size and applied it to all the coordinates produced by an outer code. But the codes applied to different coordinates need not to be the same. In fact, the outer codeword has n coordinates and our space of linear codes is also of size n (as usual for "careful" choice of n). Therefore we could apply different codes to each coordinate. If our space contained only a single code of distance d this would not make any sense. But as we have seen, almost all the codes in our space have good distance (at least $d/2$). More formally, Justesen defined generalized notion of concatenation as follows. Given an $[N, K, D]_Q$ code C and a vector of inner $[n, k, d_i]_q$ codes $(C_1, C_2, \ldots, C_N)$ such that for all but ϵN inner codes $d_i \geq d$, he defined $C \cdot (C_1, C_2, \ldots, C_N)$ to be the code obtained by applying C_i on i-th coordinate of outer code (and concatenating the results). When applying this scheme on two different codewords we get two outer codewords that differ in D coordinates. At most ϵN inner codes are "bad" and therefore the resulting codewords will differ in at least $(D - \epsilon N)d$ positions. Therefore we have obtained $[nN, kK, (D - \epsilon N)d]_q$ code.

Using this idea Justesen multiplied an $[n, n/2, n/2]_{2^l=n}$ family of RS codes (with explicit specification of the fields) by Wozencraft ensemble (which is very explicit). This construction is "very, very, very" explicit, in particular, at least locally polynomially computable.

For the efficient decoding of Forney and Justesen codes it is sufficient to be able to decode the outer code efficiently since the inner code can be decoded in polynomial time by brute force.