CS 229r Essential Coding Theory Feb 23, 2017

Lecture 10

Instructor: Madhu Sudan *Scribes: Jason Goodman*

1 Definitions

1.1 Explicitness

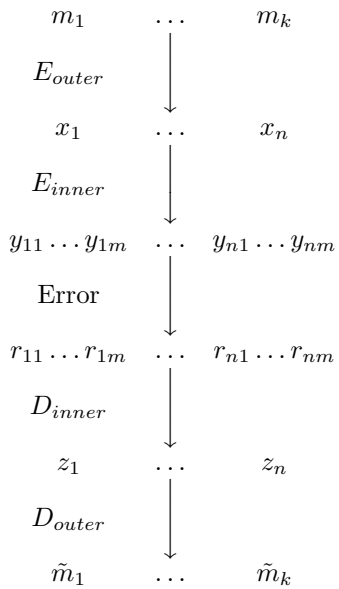
Our goal this lecture is to find good explicit codes, as opposed to non-constructive codes. Since we're in a finite setting, we'll take this to mean codes with polynomial-time encoding and decoding algorithms. We'll see that concatenated codes can be efficiently decoded to correct half of their minimum distances and achieve Shannon capacity on any memoryless channel, though we pay a price of rate and distance for the explicitness.

For intuition of a non-constructive result in a finite setting, consider an argument by the pigeonhole principle that a function with more inputs than outputs has a pair of inputs with the same output. The only way to find such a pair in general is enumerate all possibilities, which we consider non-constructive.

1.2 Parameters

Recall that we're composing an outer code over a large alphabet with an inner code over a small alphabet. For our purposes, consider concatenating codes with parameters $[n, k, d]_{\mathbb{F}_{2^l}}$ and $[m, l, d_0]_2$ to create an $[nm, kl, dd_0]_2$ code, using a linear conversion from $\mathbb{F}_{2^l}$ to $\mathbb{F}_2^l$ to keep the resulting code linear. Also ensure that $m = O(\log k)$ and the outer code is explicit and has a decoding algorithm that corrects $d/2$ errors—e.g. use Reed-Solomon, BCH, or algebraic geometry codes. We don't need the inner code to be explicit.

Call an unencoded message $m_1 \dots m_k$, where each $m_i \in \mathbb{F}_{2^l}$; call the result of the outer encoding $x_1 \dots x_n$, where each $x_i \in \mathbb{F}_{2^l}$ and can be viewed as members of $\mathbb{F}_2^l$; and call each x_i's encoding under the inner code $y_{i1} \dots y_{im}$.



Note that if the outer and inner codes are linear and we use a linear conversion betweeb alphabets, we get a linear code. Its generator matrix is the composition of an expanded version of the outer code's generator matrix—each coordinate becomes an $l \times l$ submatrix that acts on members of $\mathbb{F}_2^l$—composed with a matrix

that applies the inner code's generator to each chunk of m bits—a string of n copies of the inner code's generator matrix along the diagonal and zeros elsewhere.

2 Simple decoding algorithm

Because $m = O(\log k)$, we can decode the inner code by brute force, which will take time $O(2^m) = k^{O(1)}$. Then just apply the outer code's efficient decoding algorithm.

Theorem 1. *This decoding algorithm corrects $d_0 d/4$ errors.*

Proof. Starting with the outer code, we know $\tilde{m}_1 \dots \tilde{m}_k = m_1 \dots m_k$ if $\#\{j | z_j \neq x_j\} \leq d/2$. That is, the inner code must be correct everywhere but up to $d/2$ blocks. The inner code can correct d_0 errors, so we need $\#\{j | \#\{i | r_{ji} \neq y_{ji}\} \geq d_0\} \leq d/2$. Therefore, we can always correct up to $d_0 d/4$ errors. With more errors, it's possible for more than $d/2$ blocks to go wrong in the inner decoding. □

This result is unsatisfying, since we'd like to correct half of the code's distance, $d_0 d/4$. Something we aren't taking advantage of is that we can tell if inner code blocks are likely to be incorrect by their distances from the nearest inner codewords.

3 Improved decoding algorithm

3.1 Errors and erasures decoding

Consider the problem of decoding a message where some symbols have been changed to other values (error), while some others have been replaced with question marks (erasure). We can adapt, for example, a Reed-Solomon code to take advantage of the fact that erasures are evident to the decoder.

Start with an $[n, k, d]$ code. If there are s erasures, we effectively have an $[n - s, k, d - s]$ code, which can now correct t errors if $t < (d - s)/2$. Equivalently, erasures have half the cost of errors; we can correct s erasures and t errors if $2t + s < d$.

We'll use this insight to have the inner code's decoding declare more ambiguous inner blocks to be erasures for the outer code rather than choose the closest inner codeword.

3.2 Algorithm

Our approach, initially randomized, will be to consider the number of perceived errors e_i for every inner block i. That is, the distance between $r_{i1} \dots r_{im}$ and $E_{inner}(z'_i)$, where z' is the brute force decoding of $r_{i1} \dots r_{im}$. We'll now define a new decoding for the inner code that can specify erasures; then we can use errors and erasures decoding for the outer code.

For each block i, if $e_i \geq d_0/2$, then $z_i =?$. Otherwise, declare an erasure with probability $e_i/(d_0/2)$, which is proportional to e_i.

3.3 Analysis

Theorem 2. *This decoding algorithm corrects $d_0 d/2$ errors.*

We won't complete the proof (exercise) but will give an overview of its structure. Most of the work is showing that if the number of errors is less than $dd_0/2$, then $Exp[2t + s] < d$, where s and t are random variables for the number of erasures and errors coming out of the inner decoding. To derandomize this algorithm, first show that the expectation holds if we choose a single threshold between 0 and $dd_0/2$ uniformly at random and declare blocks to be erasures if they're further than the threshold from a codeword. Then we know some particular threshold does as well as the expectation, giving a deterministic algorithm.

To show the expectation, let $\tilde{e}_i$ be the actual number of errors in block i, as opposed to the perceived number, and consider possible outcomes when $e_i < d_0/2$:

In the case that $\tilde{e}_i = e_i$, we either get the right block or declare an erasure, so using indicator random variables for contributions to t and s from i, $Exp[2t_i + s_i] = Exp[s_i] = \tilde{e_i}/(d_0/2)$. In the case that $\tilde{e}_i \neq e_i$, $\tilde{e}_i \geq d_0 - e_i$ by the triangle inequality, since the inner code's distance is d_0. So we have $Exp(2t_i + s_i) = 2\left(1 - \frac{d_0 - \tilde{e_i}}{d_0/2}\right) + \frac{d_0 - \tilde{e_i}}{d_0/2} \leq \frac{\tilde{e_i}}{d_0/2}$.

4 Achieving Shannon capacity

Recall that a Binary Symmetric Channel BSC(p) flips each bit independently with probability p. Its capacity is $1 - H(p)$, and we're interested in finding a code with polynomial-time encoding and decoding algorithms and rate $1 - H(p) - \epsilon$.

We can start by breaking the input into blocks, now of size roughly $2 \log k$, and using a brute force algorithm for each block. This gives an error probability per block to be bounded by $1/k^{1.5}$ and runs in time exponential in the block length and therefore polynomial in k. A union gives a small probability of incorrect decoding, but polynomially rather than exponentially small in k.

To get a smaller failure probability, we'll put an outer code around the brute force one, analogous to the approach above. Notice that an inner code doesn't preserve the channel's properties, so we'll want to use a code with worst-case guarantees like Reed-Solomon rather than attempt to analyze a messier error distribution.

Theorem 3. *Concatenations of Reed-Solomon codes with rate $1 - \epsilon$ and the above brute-force code create explicit codes arbitrarily close to the capacity of a BSC channel (ϵ away) with error probability $exp(-k)$.*

Proof. The probability of an error in bock i is exponentially small in the length of the block $<< \epsilon/2$. The probability that at least $\epsilon/2$ blocks are decoded in error is then $\leq exp(-\epsilon^2 n)$ by a Chernoff bound. □

This result can be generalized to any memoryless channel (exercise).

5 History

Forney describes explicit codes to achieve Shannon capacity on a memoryless channel in his 1966 paper. Since then, several others have claimed to do the same thing "for the first time." How are they different?

Mid 90s: Turbo codes give empirical results from purpose-built hardware.

Late 90s: research on irregular LDPC codes clarifies that the relevant research goal is bringing down large constant factors. In particular, encoding and decoding should take time polynomial in ϵ, the gap between a code's rate and its channel's capacity.

2005-12: Polar codes are shown to meet this specification: Explicit codes that run in time poly(ϵ).

Recent: Reed-Muller codes achieve capacity. This is a weaker result dealing with erasure rather than error that doesn't show that its runtimes depend reasonably on ϵ, but the math is nice.