CSE 533: Error-Correcting Codes **(Autumn 2006)**

Lecture 12: Near-Capacity Polynomial-Time Codes for BSC_p

11/08/2006

Lecturer: Venkatesan Guruswami *Scribe: Alexander Jaffe*

After recapping last lecture, we return to studying the Binary Symmetric Channel, in order to present a code that nearly achieves Shannon's capacity. We will also begin to discuss expander-based codes in the Hamming environment, which are a particularly combinatorial (rather than geometric or algebraic) approach to good codes.

1 Review of Last Lecture

Last time, we discussed:

- The Welch-Berlekamp decoding algorithm for Reed-Solomon codes.
- A simple decoding algorithm for concatenated codes, which corrects $< \frac{dD}{4}$ errors for the Justesen code.
- A stronger, randomized decoding algorithm for the Justesen code due to [Forney '66]. This Generalized Minimum Distance (GMD) algorithm can correct fewer than $\frac{dD}{2}$ errors.
- A fully functional derandomized version of GMD.

Derandomized GMD can generally be summarized as follows. Each inner block z_i is encoded as y_i, with some weight w_i representing the likelihood that the block was decoded incorrectly. For sorted $w_{i_1} < w_{i_2} < \ldots w_{i_n}$, we erase all blocks past some threshold Θ, leaving only $w_{i_1}, w_{i_2}, \ldots, w_{i_j}$ to be decoded. We try this for all distinguishing values of Θ, that is, for all possible values of j. Picking the j that minimizes the distance between that codeword and the received codeword gets the most likely codeword; this is guaranteed to be correct for fewer than $\frac{dD}{2}$ errors. Note that this requires running the decoding algorithm $d/2$ times, in order to try at most every possible weight threshold, ($0 \leq w_i \leq d/2$). Thus the overall running time is $O(d \cdot Time_{RS-Decoding})$.

2 Background and Motivation

For this lecture, we will concentrate on the relationship between the rate of the code and the fraction of errors we can correct. This is a more direct study of the quality of a code than our our previous discussions, which compared relative distance and rate.

We previously saw that Reed-Solomon lets us correct, for rate R and distance $d = 1 - R$, up to $\frac{d}{2} - o(1)$ errors. We know that it is impossible to correct more than half a code's distance in errors

and still return a unique answer. We will relax this uniqueness requirement later in the course, but for our current consideration, Reed-Solomon is in some sense the best we can do. Unfortunately, the alphabet size is very large, and impractical for many purposes.

In the binary case, we have seen that it is possible to correct up to a $\frac{\delta\Delta}{2}$ fraction of errors. This was achieved by concatenating an outer code of distance Δ with an inner code of distance δ and the best possible rate $(1 - H(\delta))$, (as given by the GV bound). From the Zyablov bound, this achieves an overall rate of $R = (1 - \Delta)(1 - H(\delta))$.

If our goal is to correct a maximal number of errors, then we can set the parameters of the above concatenated code accordingly. Letting $\delta = \frac{1}{2} - \varepsilon$ and $\Delta = 1 - \varepsilon$ allows us to correct a $\frac{1}{4} - \varepsilon$ fraction of errors. However, the rate of such a code is only $\Omega(\varepsilon^3)$, which is not particularly desirable. In fact, this does not meet the best bound on rate for such high correctability, namely $\Omega(\varepsilon^2)$, given by the GV bound.

In this lecture, we take a somewhat different approach. Rather than maximizing the number of errors we can correct, our goal will be to correct only a small number of errors with high probability, but to achieve the best possible rate while doing so. Assume that we wish to correct a fraction γ of errors, with $\gamma \to 0$. Then we will construct a concatenated code with outer distance $\Delta = 2\sqrt{\gamma}$, and inner distance $\delta = \sqrt{\gamma}$. From the Zyablov bound, the rate is roughly $(1-2\sqrt{\gamma})(1-\sqrt{\gamma}\log\frac{1}{\gamma}) \leq 1 - 2\sqrt{\gamma}\log\frac{1}{\gamma} = 1 - O(\sqrt{\gamma\log\frac{1}{\gamma}})$. Note that this rate approaches 1 as γ approaches 0; thus we come arbitrarily close to capacity if we are willing to correct a sufficiently small fraction of errors. We will formalize this more after we describe the code in detail.

3 A Concatenated Code over BSC_p

In this section, we prove the following main theorem.

Theorem 3.1. *There exist codes constructible in poly$(n)2^{\text{poly}(\frac{1}{\varepsilon})}$ time that are within ε of capacity of BSC_p, which can be encoded in poly$(\frac{n}{\varepsilon})$ time, and decoded in poly$(n)2^{\text{poly}(\frac{1}{\varepsilon})}$ time.*

For the most part, this resolves the question posed by Shannon's Theorem. However, some exceptions (instances of further work) will be given.

3.1 Overview of the Code

We construct another example of an concatenated code. This time, we will use both an inner and outer code with high rate, in order to ensure that the overall code achieves a rate that is as high as possible.

The Shannon Theorem states that capacity for reliable communication over BSC_p is a $1 - H(p)$ fractional rate. Thus far, for $0 < p < \frac{1}{4}$, we have shown how to communicate on BSC_p reliably with positive rate and polynomial time encoding and decoding. We would now like to be able to achieve maximally good rate, though it means we will be capable of correcting a much smaller fraction of errors.

We will now give rate $1 - H(p) - \varepsilon$ codes, for any arbitrary ε. These codes will be explicit, featuring polynomial-time encoding and decoding algorithms E and D, which satisfy the following.

$$Pr_{\text{noise of BSC}_p}[D(E(m) + \text{noise}) \neq n] \leq 2^{-\Omega(n)}$$

We call this family of code C^* – we will at times abuse notation and refer to a particular member of the family as C^*. We will show that the $2^-\Omega(n)$ bounds holds, but truthfully any negligible function on n (a function that approaches 0 faster than any inverse polynomial on n) would be sufficient. Such a bound on the probability of error is the very definition of a good code on channel BSC$_p$.

The code is due to [Forney 66]. We can push $1 - H(p) - \varepsilon$ arbitrarily close to the bound of Shannon's Theorem, so this code essentially resolves work on maximizing rate over BSC$_p$.

3.2 Construction

We start by considering the ideal inner code: a binary linear code of rate $1 - O(\sqrt{\gamma \log \frac{1}{\gamma}})$, where γ is some function $\gamma(\varepsilon)$. Such a code must exist, by the GV bound. We can think of the rate as $1 - C\gamma$, for some C_γ s.t. $C_\gamma \to 0$ as $\gamma \to 0$. This code has very large rate, but corrects a tiny fraction of errors.

We could achieve much of this section by concatenating a Reed-Solomon code with such a good code found by search, but we do things somewhat differently in order to keep the alphabet binary. Furthermore, though such a code would introduce little loss in rate, it would would correct too few errors. Instead, we use the Justesen code at the outer level for its good rate. The fact that the Justesen code is itself a concatenation code will be irrelevant here – we treat is a black box, simply for its behavioral properties.

We now consider what properties the desired inner code should have. It must be close to capacity itself, for concatenation can only decrease the rate. (The rate of a concatenated code is the product of the rates of the inner and outer codes.) C_{in} should thus be a $1 - H(\delta) - \sqrt{\gamma}$ binary linear code s.t. $Pr_{\text{noise of BSC}_p}[\text{MLD}(C_{in}(m) + noise) \neq m] \leq \frac{\gamma}{4}$. Here MLD is Maximum-Likelihood Decoding, in which we pick the codeword that is most likely to have been transmuted to the received string under the error model. By the Shannon Theorem, we know that such a code exists. We can ensure such an error bound provided $b \geq b(p, \gamma) \geq \frac{1}{\gamma^2} log(\frac{1}{\gamma})$. We are thus given an indication of how large the block size b should be, since this drives down the error probability.

Suppose we can decode the inner code very reliably on BSC$_p$. Then with high probability each block is correct, independently from one another. At the outer level, there may be a small number of blocks that are incorrect, but the redundancy of the outer code will almost certainly be able to correct for these few.

Since we can construct such a good C_{in}, it is natural to ask why we do not use C_{in} to encode the entire message as a single block. Such a code would be very effective, but we unfortunately know of no way to perform the decoding in polynomial time (or to find such a code quickly, though this is less important). Thus such a code is only feasible when used on a block of constant or logarithmic size in n.

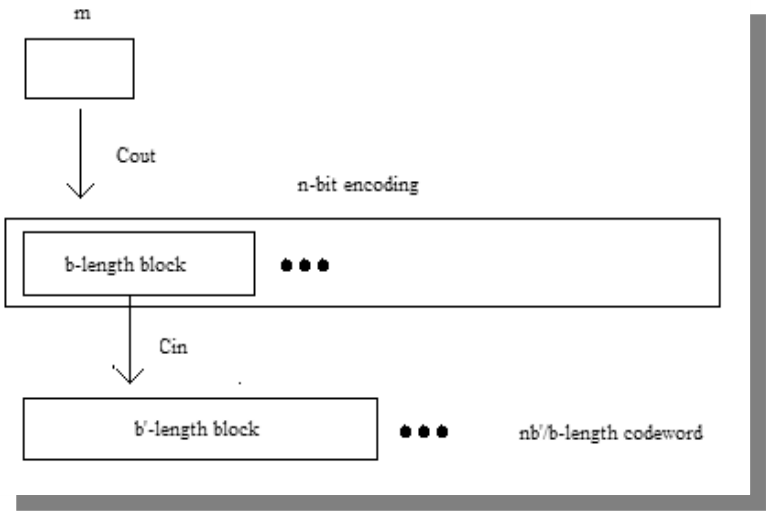

Figure 1: Encoding process of C^*

We search for an appropriate C_{in} in the following way. Because a constant b has been chosen for the dimension of C_{in}, any function of b will only be an (potentially large) constant. We can thus perform exhaustive search to find an optimal generator matrix. We try each possible $b \times b'$ generator matrix. For each of these matrices, and each possible non-zero b-length message m, we consider the probability that m decodes to the zero-vector. By summing these error probabilities, we can compute the total probability that a codeword altered by the channel decodes to the wrong codeword. We can then take the generator matrix that minimizes this probability.

3.3 Encoding and Decoding

Inner encoding is performed via the generator matrix, and thus takes polynomial time in b. Maximum Likelihood Decoding (MLD) can be implemented in $2^{O(bb')}$ time, by encoding every possible message with the generator matrix, and choosing the message that maps to the codeword closest to our received bit string. Despite the exponential dependence on b, this is still constant on n. We use the Justesen code as our outer code, initially encoding the message using it. We then split the codeword of the Justesen code into n/b blocks of length b. We map each of these blocks via an inner code to blocks of some length $b' = \ell b$. Our final codewords consist of the concatenation of these inner codewords. Decoding C^* simply consists of reversing this process: decoding each block with the inner code, then decoding the concatenation of those outputs with the outer code.

Decoding each block in step 1 takes $2^{O(bb')}$ time, (technically a constant). The outer code, as we know, decodes in time polynomial in n. Let $n' := \frac{n}{b}$, the number of blocks. Note that our overall block length is $N = n'b' = n\frac{b'}{b}$. Then the total running time of the decoding algorithm is $O(n') * 2^{O(bb')} + \text{poly}(n) = 2^{O(bb')} + \text{poly}(n) = \text{poly}(n)$. Ignoring the constant inner code decoding time, it is in fact possible to achieve an overall time that is linear in n.

We also know that the concatenation code is linear because each inner code is linear – when we

sum two codewords of the overall code we sum each pair of inner codewords independently, and the same applies for scalar multiplication. Thus, $C^* = C_{out} \Delta C_{in}$ is an explicit, binary, linear code of rate $(1 - C_\gamma)(1 - H(p) - \gamma) \leq 1 - H(p) - 2C_\gamma$, with polynomial-time encoding and decoding.

Recall that C_{in} has optimal rate. Hence by increasing the block size b, (with a corresponding increase in the constant factor of the running time) we improve the rate of the overall code. In fact, for an arbitrary ε, we can achieve rate $1 - H(p) - \varepsilon$ in time $\text{poly}(n)2^{\text{poly}(1/\varepsilon)}$, getting arbitrarily close to capacity. For a given ε away from maximal rate, we can correct up to a γ fraction of errors, when $C_\gamma < \frac{\varepsilon}{2}$. $\gamma = \varepsilon^3$ will suffice.

It is worth noting that work continues on improving the constant factors in the inner decoding algorithm's running time. With some cleverness, it is possible to construct the generator matrix in time $2^{O(b^2)}$. However, as this is only a constant improvement, we will not go into detail on the method here.

If we wish the error probability to approach 0, then we must select our block length such that all but a γ fraction of the blocks are correct. However, to give ourselves some slack in the computation, we will set b so that we can correct only $\frac{\gamma}{4}$ blocks.

The decoding algorithm for the overall code is stated explicitly below.

Decode(s):

1. Run C_{in} decoder on each sequential block of b' bits of s.
2. Run C_{out} decoder on the concatenation of the outputs from step 1.
3. Output the result of step 2.

This simple procedure should be familiar from a previous concatenated code.

3.4 Reliability

We wish to prove that the probability of error for the code we have described is negligible in n – in particular we prove the stronger probability of error of $2^{-\Omega(n)}$.

Let us consider under what conditions the overall decoder produces an error. It is clear that if the fraction of incorrectly decoded blocks is less than γ, then the fraction of incorrect bits in the input to C_{out} is also less than γ. Thus the overal decoding algorithm decodes its input to an incorrect codeword only if greater than γ fraction of the blocks are decoded incorrectly by C_{in}. By the independence of each block and a union bound, the probability of this event is:

$$Pr[\text{error}] \leq \binom{n'}{\gamma n'} (\frac{\gamma}{4})^{\gamma n'} \leq (\frac{n'e}{\gamma n'})^{\gamma n'} (\frac{\gamma}{4})^{\gamma n'} = (\frac{e}{4})^{\gamma n/b} = 2^{-\Omega(n)}.$$

As intended. This probability serves as an upper bound on the overall probability of error. We could potentially get a tighter expression by using a Chernoff Bound, since the events are i.i.d.; the bound above will suffice however.

Implementation Note: It is possible to push the exponential dependency on b out of the running time and into the space requirements. This is accomplished by precomputing a complete lookup

table for each possible input, mapping it to the correct codeword. This table would clearly have size that is exponential in b, but would allow lookup in $O(b)$.

We have now proved the main theorem. It is still an open problem to construct a code that approaches capacity with only a polynomial dependency on ε in the running time. However, what we have constructed in this section is quite satisfactory. It in some sense solves Shannon's problem fully. Note that the key to this success is in fact concatenation on the Reed-Solomon code: the only outer code we have seen that has the necessary properties here is the Justesen code, which itself requires the Reed-Solomon code at its outer level.

4 Expander Codes

We now return to the Hamming domain. We will introduce codes inspired by a construction of [Gallager 1960]. These will achieve positive rate and relative distance, with linear time decoding.

4.1 Sparse Parity Check Matrices

We take the parity check matrix view. A sparse parity-check matrix is one in which each column has O(1) non-zero entries. In some cases, we might also add the constraint that each row has O(1) non-zero entries. The key feature of such matrices is that they are *sparse*: there are only O(n) non-zero entries in total. This is powerful, because it is in general possible to decode from a parity check matrix in time proportional to its Hamming weight.

Gallager showed that when picking such matrices randomly, increasing sparsity pushes a code arbitrarily close to the GV bound. In contrast, we will attempt to deterministically construct codes with such good rate, using a tool called *expander graphs*. Using such graphs in the construction of codes is a relatively new technique, dating back to 1996 at the earliest.

A parity check matrix has n columns, and $n - k$ rows. We can represent it alternatively as a *factor graph*. This is a bipartite graph, with n 'variable' nodes on the left side (V), each corresponding to one column, and $n - k$ 'check' nodes on the right (C), each corresponding to one row. We connect a vertex V_i with a vertex C_j iff there is a 1 in entry j, i of the matrix. We can thus think of the matrix as a non-traditional adjacency matrix for the graph. (Each column and row corresponds to a separate vertex, thus the matrix is not symmetrical.) Note that factor graphs corresponding to sparse parity check matrices have $O(n)$ edges in total.

The vertices of C are called check nodes for the following reason. Any codeword can be 'placed' on the nodes of V, assigning each vertex the number 1 or 0. The check nodes then enforce a parity check, because each column node has an edge to each row node that it is non-zero in.

If we can explicitly construct the graph described above, then we will be finished, because it is equivalent to, and can be used as, a parity check matrix. In fact, we can reconstruct the generator matrix from the parity check matrix in $O(n^3)$ time, by computing a basis kernel for the parity check matrix. This gives us both polynomial-time encoding and decoding.

4.2 Expander Graph Background

Definition 4.1. *An $(n, m, d, \gamma, \alpha)$-expander is a bipartite graph $G = (L, R, E)$ s.t. $|L| = n, |R| = m$, vertices in L have degree d, and $\forall S \subseteq L$ s.t. $|S| \leq \gamma n$, we have that $|N(S)| \geq \alpha d|S|$.*

The intuitive interpretation of an expander graph is that the neighborhood of any reasonably small subset of vertices should be somewhat large. It is impossible to achieve large expansion α for γ greater than 1, since $|N(S)| \leq d|S|$. Conversely, for $\gamma = \frac{1}{n}$, the maximal expansion of 1 is always achievable, since $|N(S)| = d$. However, we are really interested in the cases in between, in particular for constant $\frac{1}{n} < \gamma < 1$. If for some γ in this range we achieve $\alpha = 1 - \varepsilon$, then this graph is called a lossless expander, the best we can hope for.

For a long time, the best known explicit deterministic constructions of expanders achieved only $\alpha = \frac{1}{2}$. Then, less than five years ago, it was shown that expanders with $\alpha = 1 - \varepsilon$ could be constructed explicitly. In general, we find that $\forall\ \varepsilon > 0,\ \exists\ \rho > 1, d, \gamma > 0$ s.t. there exist explicit $(n, \rho n, d, \gamma, \alpha)$-expanders. Thus, given an arbitrarily small ε, one can pick parameters to construct expanders with $1 - \varepsilon$ expansion.

Next time we will state and prove the fact that for $\alpha > \frac{1}{2}$, an expander graph will be a factor graph corresponding to the parity check matrix for a good code. It will be a very simple and explicit construction. In particular, we will discuss the 1996 "Expander Codes" due to Sipser-Spielman, and 2002's "Randomness Conductors and Constant-Degree Lossless Expanders" by Capalbo-Reingold-Vadhan-Wigderson.