15-859: Information Theory and Applications in TCS **CMU: Spring 2013**

Lecture 9-10: Concatenated codes; Polarization and polar codes

February 19 & 21, 2013

Lecturer: Venkatesan Guruswami *Scribe: Patrick Xia*

This is a conglomeration of two lectures (lecture 9 and lecture 10), with the subjects "Concatenated codes and Introduction to Polarization" and "Arıkan's recursive construction of a polarizing invertible transformation."

1 Previously...

Random linear codes allow for efficient codebook storage. We can specify an $n \times k$ matrix G such that our code is

$$\{Gu | u \in \{0,1\}^k\}$$

However, this technique is deficient in some regards. G is not explicit: we only proved that a random such G achieves the channel capacity with high probability. Decoding is also not efficient: the algorithm is cubic for the BEC case, and in general NP-hard, so it is unlikely that we can come up with an efficient algorithm to decode a random linear code.

So although linear codes are nice (they are structured; you can store the code compactly), the algorithmic problems are still very difficult. How can we do better?

2 Concatenated codes

We will discuss the BSC_p, $0 \leq p < 1/2$. Recall: BSC_p is the binary-input channel that flips bits with probability p and leaves them the same with probability $1 - p$.

The capacity is $1 - h(p)$:

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= 1 - h(p) \end{aligned}$$

Fact. *Fix $p, \epsilon > 0$. Shannon's theorem implies that $\forall n \geq \Omega(1/\epsilon^2)$, there exists a linear code given by a generator matrix $G \in \{0,1\}^{n \times k}$, $k = (1 - h(p) - \epsilon)n$ (so the rate is $1 - h(p) - \epsilon$), such that $\forall u$,*

$$\Pr_{\text{BSC}(p)}[Gu \text{ is closest to } \text{BSC}(Gu)] \geq 1 - 2^{-\Omega(\epsilon^2 n)}.$$

Proof. The proof is left as an exercise. Bits of it are on the homework. □

However, the aforementioned fact is a non-constructive result, as decoding this code is still very difficult.

Insight. *Use Shannon's existential code for small lengths q and combine it with explicit long codes. Decoding is therefore done in constant time (a very large constant, but a constant in terms of n).*

2.1 Construction

Take a linear code $C \subset \{0,1\}^{n_c}$. Now consider the following adversarial channel model: given an input string s of length n_0, the channel flips up to γn_0 bits (i.e., it outputs a string that is within Hamming distance γn_0 of the input), where $\gamma > 0$ is a small constant. We are expected to decode the original string from the output of this adversarial channel model.

Assume we know how to construct explicitly (i.e., in deterministic poly(n_0) time) a linear code $C \subseteq \{0,1\}^{n_0}$ of rate $1-\gamma^{1/3}$ that can correct worst-case γn_0 errors for a small $\gamma > 0$. (Such codes typically have an algebraic construction, so they are out of scope of an information theory discussion; e.g. the Reed-Solomon codes).

What we'll do is take our input string, encode via one of these codes, and split it into chunks of b bits, where $b \approx \mathcal{O}(1/\epsilon^2)\log(1/\gamma)$. Each of these chunks, we will independently encode using a Shannon code of rate $1-h(p)-\epsilon$. We can find such a Shannon code by brute-force search (the matrix G that we are brute-forcing over is constant-sized in terms of n). Each block will necessarily expand to b' bits.

What's our rate? It's $(1-\gamma^{1/3})(1-h(p)-\epsilon/2)$, which is $\geq (1-h(p)-\epsilon)$ for γ small enough compared to ϵ.

What we have made here is a concatenated code where the *inner code* is Shannon coding and the *outer code* is Reed-Solomon (or similar).

2.2 Decoding

For each block of b' bits decode it to the b bit chunk whose encoding is closest (this is brute-force search, which will take time $\gamma^{-1}2^{\mathcal{O}(1/\epsilon^2)}$, but this doesn't depend on n). Then decode the resulting string as per the outer code (which we assume to exist via the assumption of the outer code).

The runtime of the brute-force part is poly$(n_0, 2^b)$, the runtime of the outer code decoder is poly(n_0), and combining this, we can say the whole construction runs in time poly$(n_0 2^{1/\epsilon^2})$.

2.3 Correctness

If after the brute-force decoding step, the new string is within γn_0 of the original string, then the outer code will decode it successfully. So we need to write that

$$\Pr(\text{inner decodings produce a string with} < \gamma n_0 \text{ errors compared to } c) \to 1.$$

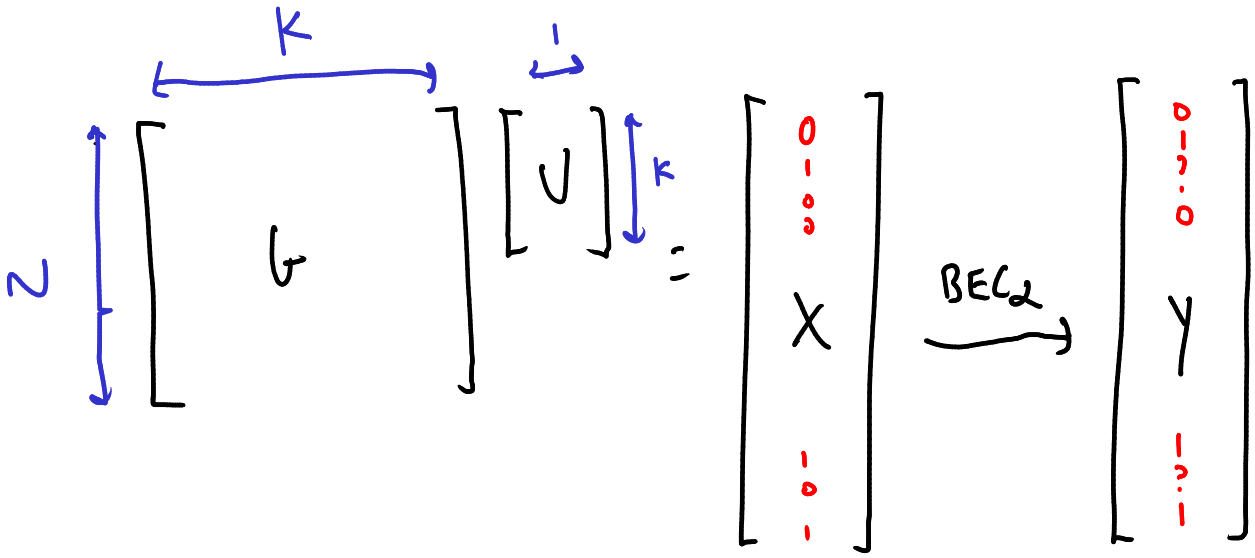
When we get a block right, we get all of the bits in the block right. So we want to say that the proportion of blocks that are decoded incorrectly is less than γ, and in turn, we want each block to be decoded with probability of error less than γ^2. Each block is decoded incorrectly with probability $2^{-\Omega(\epsilon^2 b)} \leq \gamma^2 \ll \gamma$ as proven by our choice of b.

3 Polarization

It'd be nice to have an explicit linear code that achieves capacity. Let's go back to BEC$_\alpha$ (recall: the capacity is $1-\alpha$).

Let's take a matrix G (of size $N \times K$) that generates a capacity-achieving linear code. Let's examine the properties such a G must have.

What does "G is capacity-achieving" mean? Consider the following world where we take G, encode a vector $(u_0, \ldots, u_{k-1})$ and obtain a codeword $(x_0, \ldots, x_{n-1})$. We send the codeword through a BEC$_\alpha$, and obtain $(y_0, \ldots, y_{n-1})$. The situation looks like the following:



Claim. *Take $x = Gu$ and send it through a BEC_α and obtain y. G is capacity-achieving if G allows recovery of u from y with high probability. Fano's inequality implies*

$$H(u_0^{k-1}|y_0^{N-1}) \to 0.$$

where u_i^j means $(u_i, u_{i+1}, \dots, u_j)$.

By the chain rule, we have

$$H(U_0|Y_0^{N-1}) + H(U_1|U_0, Y_0^{N-1}) + \cdots \to 0$$

which is just a statement that "if you get the full message, you can also get the ith bit."

Let's add some columns to G, call it $G_N = A \circ G$, such that the columns of A together with those of G form a basis of $\mathbb{F}_2^N$. (We can choose some canonical elements). This makes G_N an invertible matrix (as the rank of G is k and the rank of G_N is N).

Now let's say $X = G_N U$ and we send X through a BEC_α to form Y.

$$H(U_0^{N-1}|Y_0^{N-1}) = H(X_0^{N-1}|Y_0^{N-1}) = NH(X_0|Y_0) = N\alpha$$

where the first equality is because there's a bijection between U and X.

Expanding via the chain rule, obtain

$$H(U_0|Y_0^{N-1}) + H(Y_1|U_0, Y_0^{N-1}) + \cdots + H(U_{N-1}|U_0^{N-2}Y_0^{N-1}) = \alpha N.$$

The question is how are the terms on the left distributed. They could each be of value $\alpha \dots$ but the claim here is that α of them are near 1 and $1-\alpha$ of them are near 0. This is "polarization."

By virtue of the fact that G is capacity achieving,

$$H(U_{N-K}^{N-1}|Y_0^{N-1}, U_0^{N-K-1}) \to 0.$$

which means that the first $N-K$ guys are close to 1, and the last K guys are approximately 0.

Summarizing,

$$H(U_i|Y_0^{N-1}, U_0^{i-1}) \approx \begin{cases} 1 & i < (1-R)N \\ 0 & i > (1-R)N \end{cases}$$

whereas if we look at $H(X_i|Y_0^{N-1}, X_0^{i-1})$, this is a memoryless chanel, so the only information you get is from Y_i, so

$$H(X_i|Y_0^{N-1}, X_0^{i-1}) = H(X|Y).$$

So the way this works is that you are very unsure at the beginning, but then you become more and more sure. Keep in mind we have to preserve entropy or "surprise." The conclusion is that if we are building a capacity-achieving code, we have to have these properties, because we can take any capacity-achieving code and pad the matrix to obtain these results.

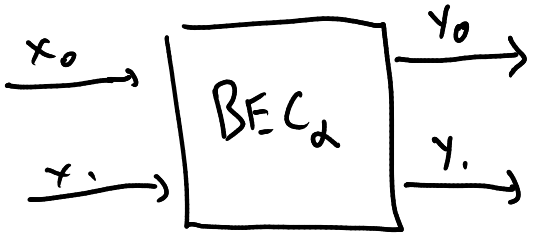
Arıkan's contribution is two-fold, and essentially shows us that a matrix with this property is sufficient for a capacity-achieving code. The insights in polar coding are therefore:

- Reversing the direction: i.e., getting codes from such invertible matrices. Arıkan will make a matrix that doesn't have such a nice property (it's not like the first x columns are nice and the remaining are bad), but we will build a matrix, and then freeze some of the input bits when the $H(U_i|\dots) \approx 1$.
- A recursive construction of such polarizing G_N. Clearly not all matrices G_N polarize: the identity matrix doesn't do any polarization at all (the entropy is still smeared over the input elements).

What is the "such" property for the matrix? Formally, we can write that there exists a set F of indices $F \subset [n-1]$, $|F| \approx H(X|Y)N$, such that if you take $H(U_i|Y_0^{N-1}, U_0^{i-1})$, this value is ≈ 1 if $i \in F$, and ≈ 0 if $i \in \overline{F}$.

Instead of telling you how to transform U to X, we will tell you how to transform X to U. (Since this is an invertible map, it doesn't really matter, and we are looking essentially at the decoding process) for the BEC_α.

$N = 2$ case:



We have $H(X_i|Y_i) = \alpha$, and our goal is to transform (X_0, X_1) into (U_0, U_1), $G_2 x = U$, such that U_0 is more uncertain and U_1 is less uncertain (and the total uncertainty needs to be the same because this is an invertible map). We notice that the matrix[1]

$$G_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

has this property: $U_0 = X_0 \oplus X_1$ and $U_1 = X_1$. Let's examine $H(U_0|Y_0, Y_1)$: this is the probability that either Y_0 and Y_1 are erased; because we have no idea what U_0 is. (Clearly we know what U_0 is if neither Y_0 nor Y_1 are erased; just xor them together), and this is $1 - (1-\alpha)^2 = 2\alpha - \alpha^2$.

Of course, $H(U_1|U_0, Y_0, Y_1)$ can be computed with conservation of entropy, but as a sanity check, let's compute it: we are only uncertain when both Y_0 and Y_1 are erased, because with a single one of those and U_0 we can recover U_1. So this is α^2, which works out, as the sum of the entropies is now 2α.

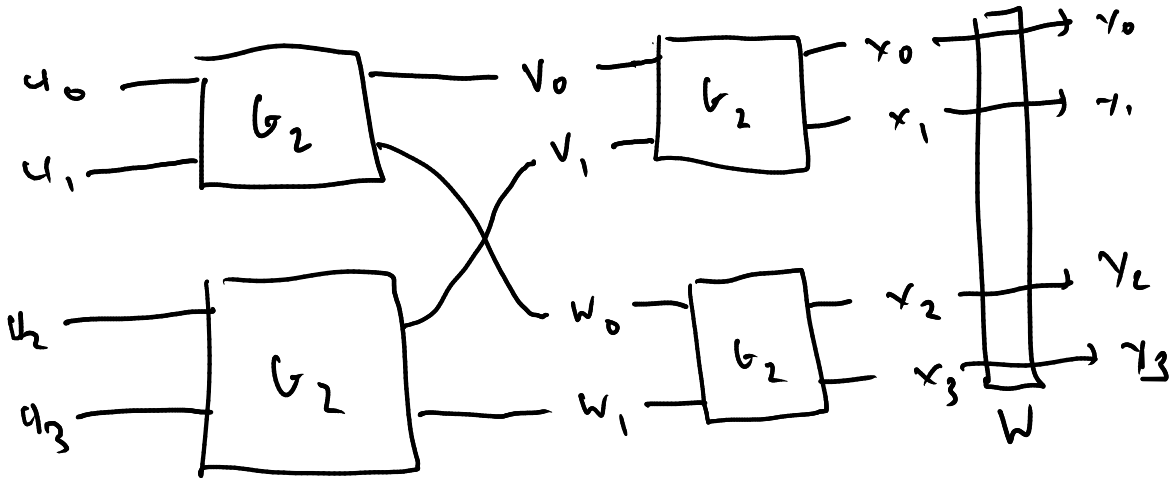
A interesting note is that G_2 is its own inverse, so we can easily map U to X and vice-versa.

Punchline. *If $\alpha \in (0,1)$, one channel becomes worse, and the other becomes better.*

The idea about getting more polarization is to iterate this. But this is tricky, so we're going to do one more example and then do the general case.

Here is what the 4×4 case looks like:

[1] If you look in the polar coding literature, you'll find the transpose of this matrix because multiplication with the input vector is typically done on the left, whereas in the CS community it is typically the other way around

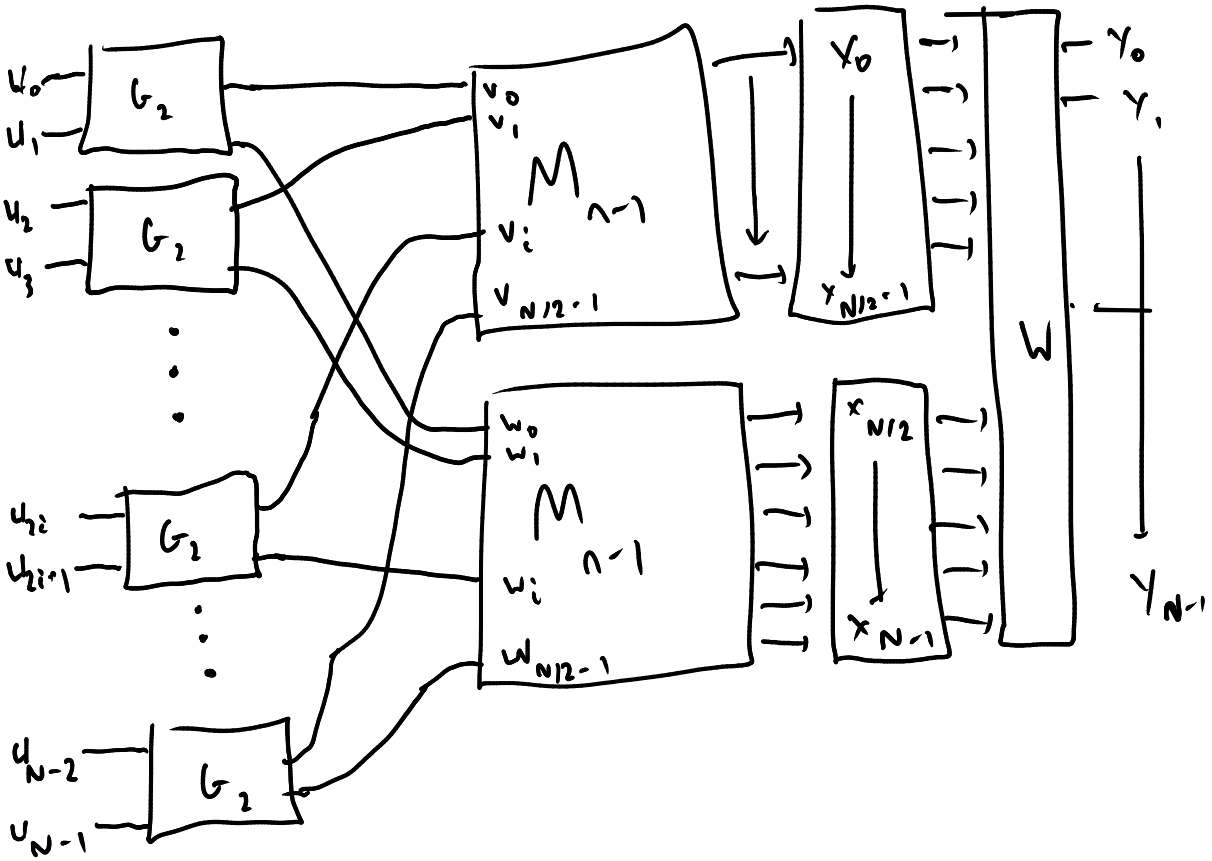


So at the end of the day, we are going to have to grunge through this 4x4 matrix of equations:

$$
\begin{aligned}
U_0 &= V_0 \oplus W_0 \\
U_1 &= W_0 \\
U_2 &= V_1 \oplus W_1 \\
U_3 &= W_1 \\
H(U_0|Y_0^3) &= \Pr(U_0 \text{ can't be determined}) \\
&= \Pr(\text{either } V_0 \text{ or } W_0 \text{ are not determined}) \\
&= (1-(1-(2\alpha-\alpha^2)))^2 \\
&= 1-(1-\alpha)^4 \\
H(U_1|Y_0^3, U_0) &= \Pr(V_0, W_0 \text{ are both unknown}) \\
&= (2\alpha-\alpha^2)^2 \\
H(U_2|U_0, U_1, Y_0^3) &= \Pr(\text{either } V_1 \text{ or } W_1 \text{ is unknown given } Y_0^3,\ V_0 \text{ and } W_0) \\
&= \Pr(V_1 \text{ is ? given } Y_0 \text{ and } V_0 \vee W_1 \text{ is ? given } Y_2, Y_3, W_0) \\
&= 1-(1-\alpha^2)^2 \\
H(U_3|U_0^2, Y_0^3) &= \Pr(V_1 \text{ is ? given } Y_0 \text{ and } V_0 \vee W_1 \text{ is ? given } Y_2^3, W_0) \\
&= \alpha^2 \cdot \alpha^2 = \alpha^4
\end{aligned}
$$

So if we keep recursing this, we will go to 0 and 1 because 0 and 1 are the only fixed points. Let's do this!

3.1 Recursive construction



Again, we will start analysis from the right.

And we can build this transformation matrix M_n out of a Kronecker product of 2×2 matrices... almost:

$$M_n = G_2^{\otimes n}$$

where the transformation M_n is given by

$$\begin{pmatrix} M_{n-1} & M_{n-1} \\ 0 & M_{n-1} \end{pmatrix} \begin{pmatrix} x_0^{N/2-1} \\ x_{N/2}^{N-1} \end{pmatrix}$$

This would suggest that $M_n = G_2 \otimes M_{n-1}$, leading to $M_n = G_2^{\otimes n}$. But actually in building M_n from M_{n-1}, we place the second half of the bits $M_n x_{N/2}^{N-1}$ at the odd indices of U_0^{N-1}. (This becomes crucial when we attempt to decode the bits in the order $U_0, U_1, \ldots, U_{N-1}$.)

It can be checked (we leave this as an exercise) that because of this the matrix M_n is given by $B_n G_2^{\otimes n}$, where B_n is a bit reversal permutation matrix:

$$(B_n \mathbf{s})_{b_1, b_2, \ldots, b_n} = \mathbf{s}_{b_n, b_{n-1}, \ldots, b_1}$$

where $\mathbf{s} \in \mathbb{R}^{2^n}$.

Note that the matrix transforming U_0^{N-1} to X_0^{N-1} (from left to right in the picture) is $M_n^{-1} = G_2^{\otimes n} B_n$. That is, $X_0^{N-1} = G_2^{\otimes n} B_n U_0^{N-1}$.

Recovering the bits U_i. We have

$$U_{2i} = V_i \oplus W_i$$
$$U_{2i+1} = W_i$$

If you know $(U_0, \ldots, U_{2i-1})$, then you know $(V_0, \ldots, V_{i-1})$ and $(W_i, \ldots, W_{i-1})$. And we have

$$H(U_{2i}|Y_0^{N-1}, U_0^{2i-1}) = \Pr[V_i \text{ is undetermined given } Y_0^{N-1},\ V_0^{i-1},\ W_0^{i-1} \text{ or } W_i \text{ is undetermined given } Y_0^{N-1} \text{ and } V_0^{i-1} \text{ and } W_0^{i-1}].$$

The two events are independent and we can split it, so we obtain

$$\begin{aligned}\alpha_n(2i) &= \Pr(V_i = ? \text{ given } Y_0^{N/2-1},\ V_0^{i-1} \text{ or } W_i = ? \text{ given } Y_{N/2}^{N-1},\ W_0^{i-1}) \\ &= 1 - (1 - \alpha_{n-1}(i)^2 \\ &= 2\alpha_{n-1}(i) - \alpha_{n-1}(i)^2\end{aligned}$$

Now let's do the other half. We need either V_i or W_i to get U_{2i+1}.

$$\begin{aligned}H(U_{2i+1}|Y_0^{N-1}, U_0^{2i}) &= \Pr(V_i = ? \text{ given } Y_0^{N/2-1},\ V_0^{i-1} \text{ and } W_i = ? \text{ given } Y_{N/2}^{N-1} \text{ and } W_0^{i-1}) \\ &= \alpha_{n-i}(i)\alpha_{n-1}(i)\end{aligned}$$