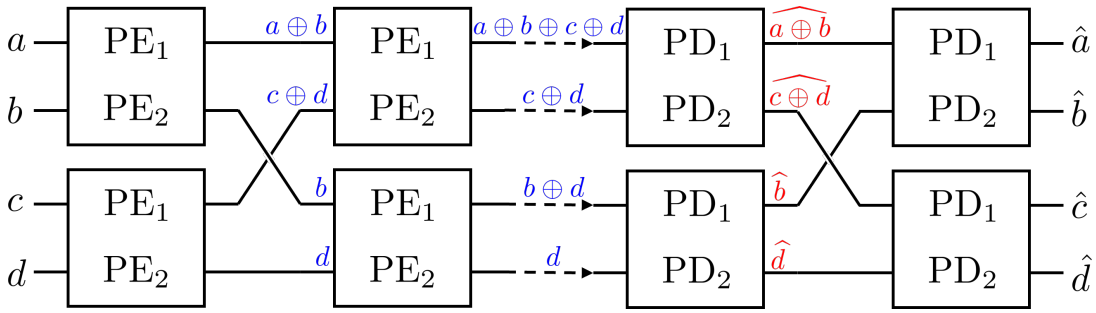
National Taiwan University, CommE5065

Modern Coding Theory & Technology Lecture Note

Instructor: Prof. Hsin-Po Wang



Wen Perng

2025 Spring

About the Course

This is the course on Modern Coding Theory and Technologies given by professor Hsin-Po Wang in the spring semester of 2025.

The lecture goes over the modern coding theory on polar codes and LDPC codes. Many of the properties and the required mathematical preliminaries such as concentration inequalities, martingale theory, and field theory are introduced along the way.

Besides on coding theory, professor Wang also has deep interest in topics such as group testing and matrix multiplication. Many such detours on diverse topics are given to broaden the students' horizon.

Here is a brief description of the items covered in each week:

- **Week 1 (2/18)**: an introduction to modern coding theory, covering the origin of polar coding and binary erasure channel.
- **Week 2 (2/25)**: basic knowledge on concentration bounds and Shannon's channel coding theorem, the wiring structure to polar code is written down.
- **Week 3 (3/04)**: special attention on decoding procedure, introduced martingale for applying on polar code over BEC, decomposition of BSC from analyzing BMSC.
- **Week 4 (3/11)**: large deviation theory and its relation with martingale theory, Reed-Muller code. Further discussion on source coding through Shannon's method, arithmetic encoding, and source polarization arguments.
- **Week 5 (3/18)**: channel parameters + martingale theory to give more characterization of polar transformation. Distributed source coding (Slepian-Wolf).
- **Week 6 (03/25)**: algebra of linear code, the functional equation regarding the speed of polarization, showing how polar code can do both source and channel coding.
- **Week 7 (04/01)**: a mock test on the midterm, covering many of the mentioned unproven results from previous lectures.
- **Week 8 (04/08)**: midterm.
- **Week 9 (04/15)**: the construction, existence and uniqueness of finite fields. Further applying the result to Reed–Solomon codes.
- **Week 10 (04/22)**: introduction to LDPC, also the evolution of erasure of LDPC over BEC. A detour into entropy complexity.
- **Week 11 (04/29)**: how to decode RS code, the quality evolution of LDPC code over BSC, solving group testing using RS code.
- **Week 12 (05/06)**: More on group testing! The binary search method and the subsequent discussions on progeny are also interesting.
- **Week 13 (05/13)**: Tradeoffs between different aspects of group testing. Distributed matrix multiplication.
- **Week 14 (05/20**: Quantum error correcting code, and a brief introduction to relational database.
- **Week 15 (05/27)**: a mock test on the final. Many new results related to the lecture are discussed.
- **Week 16 (06/03**: final exam. Some of the questions are interesting enough that I included them into the last chapter of the notes.

To be updated (2025/05/28):

- Appendix on Catalan numbers.
- Appendix on Majorization.
- w14: QECC & database
- W15: von Neumann trick and many more… (after end of 15-1)
- w16: cool stuffs

Contents

1. An Invitation to Polar Codes

18 Feb.

Our course is essentially centered around the modern coding theory of polar codes and LDPC codes. The first half will be about polar coding. Here we give an introduction to the motivation of how polar coding was come up with by Erdal Arıkan, some of the most non-trivial but stunning results are presented and proven later in the course. We will see how generalizations to polar codes and it related topics influence the agenda of our course.

1.1. The Origin of Polar Codes

Here we shall trace the paths walked by Arıkan when he first came up with the ideas of polar coding. For more details, one can refer to the paper "On the Origin of Polar Coding" written by Arıkan [Arı16].

Imaging sending two bits u_1 and u_2 over the following channel:

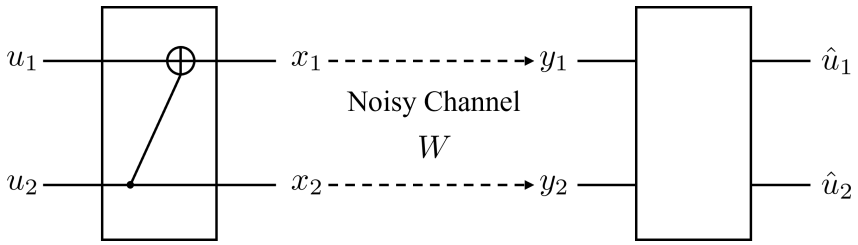


Figure 1.1. Single-layer polar code.

We have y_1 and y_2 being the channel outputs to the channel inputs $x_1 = u_1 \oplus u_2$ and $x_2 = u_2$, respectively, where $\oplus$ is the addition modular 2. If the context is clear, we may write $\oplus$ simply as $+$. If the channel W is faithful, then we can simply set the estimates as $\hat{u}_1 = y_1 \oplus y_2$ and $\hat{u}_2 = y_2$. However, since our channel is noisy, this is simply not the case. What Arıkan did to construct $\hat{u}_1$ and $\hat{u}_2$ from y_1 and y_2 was the following procedure:

(a) Guess u_1 by looking at y_1 and y_2, either via sufficient statistics, or by consensus.

(b) Assume that we did get u_1 right, guess u_2 by looking at y_1, y_2, and u_1.

What was Arıkan's motivation for considering the above channel coding scheme? How did he came up with the "magical" decoding procedure? First off, one needs to know that his considering of this transformed channel is motivated by the improving of the *cutoff rate* of a channel.

Definition 1.1: Cutoff Rate

Given a channel $W : x \to y$, the cutoff rate is defined as

$$\begin{aligned} R_0(W) &:= \max_{Q \in \Delta(\mathcal{X})} -\lg \sum_{y \in \mathcal{Y}} \left(\sum_{x \in \mathcal{X}} Q(x) \sqrt{W(y|x)} \right)^2 \\ &= \text{the rate at which we can communicate "easily."} \end{aligned} \tag{1.1}$$

Though the mathematical definition of the cutoff rate is fairly complicated, let us only focus on the text description of its purpose: one can immediately tell that $0 < R_0(W) \leq C(W)$, where $C(W) = \max_{P_X} I(X;Y)$ is the channel capacity, the theoretical maximum rate for communication over a noisy channel.

Let us denote the whole channel from the bits to the estimates as

$$\begin{aligned} W^- &: u_1 \to \hat{u}_1, \\ W^+ &: u_2 \to \hat{u}_2. \end{aligned} \tag{1.2}$$

Arıkan made the following critical observation on the two sub-channels above.

Observation: The cutoff rate of the channel increases:

$$R_0(W^-) + R_0(W^+) \geq 2 \cdot R_0(W), \tag{1.3}$$

where the inequality is almost always strict.

Remark 1.2

Note that the capacity remains unchanged:

$$C(W^-) + C(W^+) = 2 \cdot C(W). \tag{1.4}$$

The equality sign to the cutoff rate inequality Equation (1.3) holds when W is "extreme", what is its definition?

If there is a trick that can increase the cutoff rate *costlessly*, we can do it again and again: an example will be to do it twice, see the figure below.

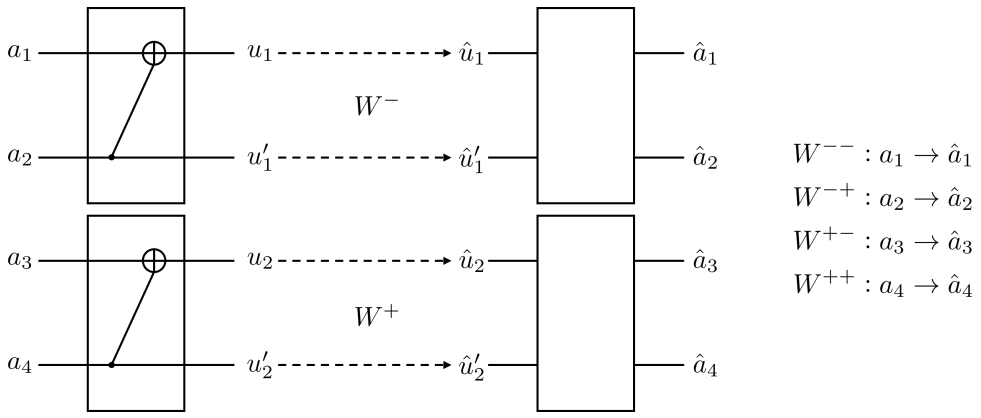


Figure 1.2. Splitting channels.

Now we have four new sub-channels indexed by the string $s \in \{+,-\}^2$, denoting whether they are from the $+$ or the $-$ branch. The average cutoff rate thus increases further:

$$R_0(W^{--}) + R_0(W^{-+}) + R_0(W^{+-}) + R_0(W^{++}) \geq 2 \cdot R_0(W^-) + 2 \cdot R_0(W^+) \geq 4 \cdot R_0(W). \quad (1.5)$$

Observation: if we repeat it indefinitely, then the average cutoff rate will continue to increase with an upper bound of $C(W)$. This bound *can*, in fact, be reached:

$$\frac{1}{2^n} \sum_{s \in \{+,-\}^n} R_0(W^s) \xrightarrow{n \to \infty} C(W) = \frac{1}{2^n} \sum_{s \in \{+,-\}^n} C(W^s). \quad (1.6)$$

Amazingly, we seem to have to accept that coding at any rate under the channel capacity is "easy." At this stage, Arıkan have almost invented polar code.

1.2. Why "Polar"? Asymptotic Behaviors of Polar Codes

This section is from the paper "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels" [Ari09]. The usage of the word "polar" comes from the following observation:

Observation: For most strings $s \in \{+,-\}^n$, the values $C(W^s)$ and $R_0(W^s)$ are "almost" close to 0 or 1. So that the sub-channels are "polarized."

To wrap up, let us combine our observations above:

(a) $\frac{1}{2^n} \sum_{s \in \{+,-\}^n} C(W^s) = C(W)$.

(b) $C(W^s)$ is either very close to 0 or 1.

(c) The number of strings s such that $C(W^s) = 1$ is about $2^n C(W)$; the number of strings s such that $C(W^s) = 0$ is about $2^n \big(1 - C(W)\big)$.

As a first example, let us see the easiest case of polar code over binary erasure channels. The cases for BSC, AWGN, or other general channels are harder and is considered later.

Observe that fixing an s such that $C(W^s) \approx 1$, W^s is such a good channel that we can transmit messages "*uncoded*," and most of the time it transmits messages faithfully. Funnily enough, we can forget cutoff rate as a whole. We will send uncoded messages over the good sub-channels W^s where $C(W^s) \approx 1$, and send deterministic bit values of 0 over the bad ones, we have the code rate as

$$\text{Code rate} = \frac{\#\text{ bits transmitted}}{\#\text{ channel uses} = 2^n} \approx C(W). \quad (1.7)$$

Hence, polar code is capacity-achieving. All that is left is to focus on what the "magic decoding procedures" are from y_i's to $\hat{u}_i$'s.

Remark 1.3

For $n = 1$, we denote W^+ as the good sub-channel and W^- as the bad sub-channel. Agreeing on sending 0's across the bad sub-channels ensures that we know that $\hat{u}_1$ is, and hence is good for obtaining the estimate $\hat{u}_2$ over W^+.

1.2.1. Binary Erasure Channel

A binary erasure channel is a tuple represented by $(\mathcal{X}, \mathcal{Y}, W)$, where $\mathcal{X} = \{0, 1\}$ is the input alphabet, $\mathcal{Y} = \{0, 1, \mathcal{E}\}$ is the output alphabet, $\mathcal{E}$ is the erasure symbol, and

$$W = \begin{bmatrix} 1-p & 0 & p \\ 0 & 1-p & 0 \end{bmatrix} \tag{1.8}$$

is the transition matrix (see diagram below). One should be careful that in literature, the misleading notational similarity between the transition probability $W(y|x)$ and posterior probability $W(x|y)$ exists, one should be careful of which one is used.

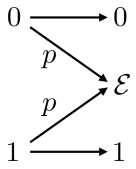


Figure 1.3. Illustration of a binary erasure channel.

A BEC is parameterized by its erasure probability $p \in [0, 1]$, i.e., a BEC channel can be simply denoted uniquely as $\mathrm{BEC}(p)$.

Why do we want to use the BEC as a first example? It is due to the following two properties:

Lemma 1.4: Properties of BEC

(a) The sub-channels BEC^+ and BEC^- are still BECs!

(b) We have

$$\mathrm{BEC}^+(x) = \mathrm{BEC}(x^2), \tag{1.9}$$
$$\mathrm{BEC}^-(x) = \mathrm{BEC}(2x - x^2). \tag{1.10}$$

Lemma 1.5: Channel Capacity of BEC

$$C\left(\mathrm{BEC}(x)\right) = 1 - x \tag{1.11}$$

Proof. Let $X \sim \mathrm{Ber}(\alpha)$, then the mutual information will be

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

The maximum is achieved at

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\alpha} I(X;Y) = 0 &= -(1-p)\left(\lg\alpha(1-p) + 1\right) + (1-p)\left(\lg(1-\alpha)(1-p) + 1\right) \\ &\to \lg\alpha = \lg(1-\alpha) \to \alpha = \frac{1}{2}. \end{aligned}$$

Hence the channel capacity is

$$C\left(\mathrm{BEC}(p)\right) = I(X;Y)\big|_{\alpha=\frac{1}{2}} = -(1-p)\lg\frac{1-p}{2} + (1-p)\lg(1-p) = (1-p).$$

Thus it is proven. ∎

From the above lemmas, we can check that BEC satisfies our previous observation:

$$C\left(\mathrm{BEC}(p)^{+}\right)+C\left(\mathrm{BEC}(p)^{-}\right)=(1-x^2)+(1-2x+x^2)=2-2x=2\cdot C\left(\mathrm{BEC}(p)\right).$$

Further, for the $+$ channels, the erasure probability reduces to x^2, while for the $-$ channels, the erasure probability increases to $1-(1-x)^2$. The $+$ channels quickly become faithful as their erasure probability $p\to 0$, while the $-$ channels quickly become useless as $p\to 1$. A quantitative description is as follows:

Theorem 1.6: Speed of Polarization

There exists a function $f:[0,1]\to[0,1]$ that has shape quite similar to $[x(1-x)]^{0.7}$ satisfying the following functional equation:

$$\frac{f(x^2)+f(2x-x^2)}{2f(x)}\approx 2^{-\frac{1}{3.627}}\approx 0.826. \tag{1.12}$$

Consequently,

$$\frac{1}{2^n f(x)}\sum_{s\in\{+,-\}^n} f\left(\text{erasure probability of }\mathrm{BEC}(x)^s\right)\approx 2^{-\frac{n}{3.627}}\approx 0.826^n. \tag{1.13}$$

Since the RHS of the above equation is small, most of "$f\left(\text{erasure probability of }\mathrm{BEC}(x)^s\right)$" is small as well. Looking at the graph of $f\approx[x(1-x)]^{0.7}$, we see that x must be close to 0 or 1. Henceforth, $\mathrm{BEC}^s(x)$ is mostly equal to either $\mathrm{BEC}(0)$ or $\mathrm{BEC}(1)$. What's more, the theorem above states that the quantitative speed of channel polarization is exponential in the order of $2^{-\frac{n}{3.627}}$. For general channels, we have the speed of polarization in the order of $2^{-\frac{n}{4}}$, with a mathematically rigorous bound currently at $2^{-\frac{n}{4.71\cdots}}$. Moreover, the derivation from Equation (1.12) to Equation (1.13) is as follows:

Proof. Assume such f exists, then plugging in x^2 and $2x-x^2$, we obtain

$$\begin{cases} f(x^4)+f(2x^2-x^4)=2^{-\frac{1}{3.627}}\cdot 2f(x^2) \\ f((2x-x^2)^2)+f(2(2x-x^2)-(2x-x^2)^2)=2^{-\frac{1}{3.627}}\cdot 2f(2x-x^2)\end{cases}.$$

Then, we have

$$\begin{aligned} &f(x^4)+f(2x^2-x^4)+f((2x-x^2)^2)+f(2(2x-x^2)-(2x-x^2)^2) \\ &\qquad =2^{-\frac{1}{3.627}}\cdot 2\left(f(x^2)+f(2x-x^2)\right)=2^{-\frac{2}{3.627}}\cdot 4f(x).\end{aligned}$$

Continue on this trend, we will obtain Equation (1.13) by induction. ∎

Equation (1.12) is sort of like an eigenvalue problem, with f as the eigenfunction and $2^{-\frac{1}{3.627}}$ as the eigenvalue. As of when this script was written, the only method for proving and showing speed of polarization is using the above "power-sum method" or some other similar variants. For more details, also refer to the paper "On the rate of channel polarization" [AT09].

If the erasure probability x is very small, say 2^{-20}, then $x^2=2^{-40}$ and $2x-x^2\approx 2^{-19}$. We can approximate the sub-channels as

$$\mathrm{BEC}(x)^s\approx \mathrm{BEC}(x^{\overbrace{2\times\cdots\times 2}^{\#\text{ of }+\text{'s in }s}}). \tag{1.14}$$

Then, for arbitrary $s \in \{+,-\}^n$, as a conservative bound we have about $n(\frac{1}{2}-\varepsilon)$ of $+$'s in s with high probability. Then if x is small, then most $\mathrm{BEC}(x)^s \approx \mathrm{BEC}(x^{2^{n/2}})$. Further recall that the amount of strings s such that $\mathrm{BEC}(x)^s$ has small erasure probability is $2^n C\left(\mathrm{BEC}(x)\right)$. Now we know that not only are a fraction (approximately $C(W)$) of W^s's are good, but they are also approximately $\mathrm{BEC}(x^{2^{n/2}})$-good, i.e., we have good channels that are "exponentially exponentially decaying!"

The agenda for this course will hence be to prove the *observations* and facts listed above. Further, we will also discuss generalizations of polar codes: BEC, BSC, AWGN, asymmetric channels, deletion channels, Markovian channels, non-stationary channels, source coding, rate distortion, wiretap, broadcast, multiple access channels, larger alphabet, larger kernel, ...

1.3. The Big Picture

There is a correspondence between probability theory and coding theory:

Probability Theory	Random Codes	Polar Codes
For i.i.d. $X_1, \ldots, X_N$, we have the *concentration inequality*: $\Pr\left\{\frac{X_1+\cdots+X_N}{N} - \mathbb{E}[X_1] \geq t\right\} \leq \exp(-ct^2N)$ for some constant c. This is also known as the *large deviation principle / behavior*, it can be derived from Hoeffding's inequality.	Over a channel W with fix rate $R < C(W) = C$, let the block length be N. Then there exists a good code of block error probability $\approx \exp(-c(C-R)^2N)$ for some constant C.	At block length N, code rate $R < C$, the bound for error probability is $\approx \exp(-\sqrt{N})$ for polar code, and can be improved to $\approx \exp(-N^{0.9})$ for other schemes.
The central limit theorem (CLT) states that $Z := \frac{X_1+\cdots+X_N}{\sqrt{N}} - \sqrt{N}\mu \sim \mathcal{N}(0,\sigma^2)$, where $\mu = \mathbb{E}[X_1]$ and $\sigma^2 = \mathrm{Var}(X_1)$. This is also known as the *small deviation principle*.	At block length N, and fix p as block error probability. The code rate is $R \approx C - O\left(\frac{1}{\sqrt{N}}\right)$.	At block length N, and fix p as block error probability. The code rate is $R \approx C - O(N^{-1/3.6})$ for polar code, and the exponent can be improved to $N^{-1/2.1}$.
The *moderate deviation principle* asks what is the distribution that the following R.V. follows? $Y := \frac{X_1+\cdots+X_N}{N^\alpha} - \frac{N\mu}{N^\alpha}$	There is a tradeoff between R and p.	There is a tradeoff between R and p.

As a side note, if the polar code is n-layer deep, then the block length $N = 2^n$.

This big picture is what one should keep in mind throughout this lecture note. Further note that the results for coding theory are often proved using random coding, which is not really practical. Hence, we also provided the bounds for practical codes such as the polar codes in the table above.

2. Polar Codes

25 Feb.

At the end of the last chapter, we characterize the behavior of codes in the asymptotic regime due to the fact that they are easier to compute and analyze. This chapter starts off by developing the essential tools in obtaining these concentration inequalities. Further, the Shannon's channel coding theorem on noisy channels is also introduced as an inspiration on how information is possible to be transmitted through erroneous channels.

Next, we delve into the theory and analysis of polar codes over noisy channels. Using tools such as the martingale theory, we see how the many observations mentioned from the last chapter are actually derived.

2.1. Concentration Inequalities

Given a collection of random variables, say X_1, X_2, ..., and X_n, that may or may not be independent or identically distributed, we would like to *control* their joint distribution. A way to control their joint distribution is to bound their moments, an example would be to assume that they are independent and identically distributed (i.i.d.), then we know that

$$\frac{X_1 + \cdots + X_n}{n} \approx \mathbb{E}[X_1].$$

The following theorem will be to characterize in what way does "$\approx$" hold.

Theorem 2.1: Hoeffding's Inequality

For i.i.d. R.V.s X_1, ..., and X_n, there exists a real number $c > 0$ dependent on the distribution such that

$$\Pr\left\{\frac{X_1 + \cdots + X_n}{n} - \mathbb{E}[X_1] \geq t\right\} \leq \exp\left(-cnt^2\right). \tag{2.1}$$

The exact value to c doesn't matter, and may vary from text to text. All that one has to know is that as long as $n \to \infty$, the bound approaches zero exponentially.

Let us consider a simple example on Bernoulli random variables and see that the Hoeffding's inequality indeed holds.

Example: Consider i.i.d. $X_1, \ldots, X_n \sim \mathrm{Ber}(0.5)$ and $t > 0$, we have

$$\begin{aligned}\Pr\left\{\frac{X_1 + \cdots + X_n}{n} - \frac{1}{2} \geq t\right\} &= \Pr\left\{X_1 + \cdots + X_n \geq n\left(t + 1/2\right)\right\} \\ &= \Pr\left\{\exp\left(s(X_1 + \cdots + X_n)\right) \geq \exp\left(s\left(nt + n/2\right)\right)\right\}.\end{aligned}$$

This holds for all $s < 0$, hence

$$\Pr\left\{\frac{X_1 + \cdots + X_n}{n} - \frac{1}{2} \geq t\right\} \leq \inf_{s>0} \frac{\mathbb{E}\left[\exp\left(s(X_1 + \cdots + X_n)\right)\right]}{\exp\left(s(nt + n/2)\right)}$$

$$= \inf_{s>0} \frac{\left[\frac{1}{2}(1+\mathrm{e}^s)\right]^n}{\exp\left(s(nt+n/2)\right)} \qquad \text{(consider the MGF)}$$
$$=: \exp(-cn),$$

where

$$c = \frac{1}{2}(2t+1)\ln\left(\frac{1+2t}{1-2t}\right) + \ln(1-2t) \gtrsim 2t^2 > 0.$$

Thus, it is shown. The right hand side to the first inequality written above is known as the Chernoff bound. □

Remark 2.2

There are two ways to generalize the Hoeffding's inequality:

(a) find the best c.

(b) use non-i.i.d. random variables.

Amazingly, if the random variables $X_i's$ are correlated, or even not identically distributed, we can still obtain similar results! But the details are more technical.

Consider the example on Bernoulli R.V.s above, replacements are required in calculating the expectation $\mathbb{E}\left[\exp(s(X_1+\cdots+X_n))\right]$. We need a different way to bound this expectation.

This introduces us to the theory of *sub-Gaussian random variables*. A random variable is called sub-Gaussian if $\mathbb{E}[\mathrm{e}^{tX}] < \mathrm{e}^{ct^2}$ for some constant c. This definition is motivated by the bounding of the tail probability: for $t > 0$,

$$\Pr\{X \geq t\} = \Pr\left\{\exp(sX) \geq \exp(st)\right\} \leq \mathbb{E}[\exp(sX)]\mathrm{e}^{-st} = \mathrm{e}^{cs^2-st} \overset{s=\frac{t}{2c}}{=} \mathrm{e}^{-\frac{t^2}{4c}}. \tag{2.2}$$

All bounded random variables are sub-Gaussian. For a weaker decaying tail, one may use the *sub-gamma random variables* to bound the tail probability.

Another nice property of sub-Gaussians is that their sum concentrates the mean, hence we have that their average having a smaller and smaller variance. The same thing sadly does not apply for sub-Gammas.

Besides the Chernoff bound and Hoeffding's inequality, the Chebyshev inequality and Markov inequality are also commonly used bounds. It is interesting to note that Chebyshev is actually the teacher of Markov and some may regard all the inequalities above as just special cases of Markov's inequality.

2.2. Shannon's Noisy-Channel Coding Theorem

A very well written paper which forms the onset of the information theory is C. E. Shannon's 1948 paper "A Mathematical Theory of Communication" [Sha48], in which the famous results such as source-channel separation, source coding, and channel coding are given.

2.2.1. Channels

In the first chapter, we have introduced terms such as capacity and rate, which are properties of a channel. So what exactly are channels? We shall illustrate them with examples.

The first three discrete channels shown below are, in order from left to right, *binary erasure channel* (*BEC*), *binary symmetric channel* (*BSC*), and Z-channel. The probability p in BEC is termed the erasure probability; the probability p in BSC is termed the cross-over probability.

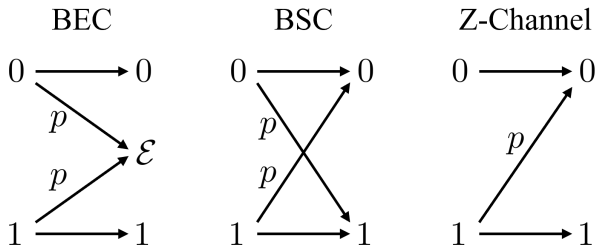


Figure 2.1. Some common channels: binary erasure channel, binary symmetric channel, Z channel.

Note that the Z-channel is a really bad channel. Though it seems to be a much simpler channel in comparison to BEC or BSC, it is bad since we cannot use linear code on it. The literature on linear code is so vast and widely used that if a channel cannot use linear code, the channel is deemed as a bad one.

Another example of a channel would be the continuous *additive white Gaussian noise* (AWGN) channel. See the figure below, when an input bit is sent, a white (unbiased) Gaussian noise of variance σ^2 is added, resulting in a continuous spectrum of possible received signal.

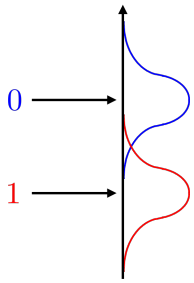


Figure 2.2. Additive white Gaussian noise channel.

In general, a discrete memoryless channel is characterized by the tuple $(\mathcal{X}, \mathcal{Y}, W)$, where the transition probability from $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ is $W(y|x)$.

2.2.2. Shannon's Noisy-Channel Coding Theorem

Here we give a more hand-wavy description of Shannon's theory, relating the capacity, rate, allowed error of a channel with its possibility to transmit information.

Theorem 2.3: Noisy-Channel Coding Theorem

For a channel W, there exists a number $C(W) > 0$, such that one can transmit $C(W) - \varepsilon$ bits per usage and guarantee the existence of a code with error probability arbitrarily small.

First off, this is a highly *non-trivial* result! How is it even possible for one to transmit information in the face of noises and errors?

Let us consider the following demonstration using BEC: select a subset $\mathcal{C} \subset \{0,1\}^n$ of *codewords* for transmission. For example, in the case of $n = 3$, the repetition code is $\mathcal{C} = \{000, 111\}$, sending 000 if the message 0 is to be sent, sending 111 if the message 1 is to be sent. The repetition code has a

code rate of $R = (\#\text{ of bits of information sent})/(\#\text{ of channel use}) = 1/3$. Can we do better? Yes, consider sending the message 1, 2, 3, and 4 with the codewords $\mathcal{C} = \{000, 011, 101, 110\}$. This code has a code rate of $R = 2/3$. And in the face of erasure, we can still determine the message most of the time: the decoder will be

$$\begin{aligned} \text{Dec} : 1\mathcal{E}0 &\mapsto 110 = \text{message 4} \\ 00\mathcal{E} &\mapsto 000 = \text{message 1} \\ 1\mathcal{E}\mathcal{E} &\mapsto ? \end{aligned}$$

Notice that if two or more erasures occur, the above code will not have perfect reconstruction. But this is alright, since we can just try again and again to decrease the failure probability.

A characteristic of the above demonstration is that when we increase the code rate, the error probability also increases. In practical designs, we care more about the increasing code rate instead of reducing the error probability, after all, who knows the difference between 10^{-6} and 10^{-9}?

Here we will give a more detailed example regarding the existence of a good coding scheme on BEC.

Example: Consider transmission with the channel $\text{BEC}(0.5)$. Suppose that n is big, and we can pick a $\mathcal{C} \subset \{0,1\}^n$. How do we choose good codewords $\mathcal{C}$? A common strategy for theoretical analysis is by *random coding*.

First, let us consider picking random codewords w^1 and $w^2 \in \{0,1\}^n$. Let us send w^1 over W to obtain, for example

$$W : w^1 = 0011011\ldots \mapsto 0\mathcal{E}110\mathcal{E}1\ldots = y,$$

a question will be to ask what is the error probability that y is *compatible* with w^2?

The received y will be compatible with w^2 if for all index 1 to n, either $y_i = \mathcal{E}$, or that $w_i^2 = y_i = w_i^1$, each happening with probability $\frac{1}{2}$ and $\frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4}$. So the total probability of a single bit being compatible will be $\frac{3}{4}$. So as $n \to \infty$, y is compatible with w^2 with a probability of $\left(\frac{3}{4}\right)^n \to 0$, meaning the two are incompatible. Moreover, we know that y is compatible with w^1 with a probability of $1 - \left(\frac{3}{4}\right)^n \to 1$, meaning we have a high probability of knowing y is indeed w^1.

□

However good the success probability may be, the rate of the above scheme is $R = 1/n \to 0$. Can we do any better?

Example: Consider the same as above, but this time, we allow for a larger set of codewords: $\mathcal{C} = \{w^i\}_{i=1}^m$. By including m codewords, we can ask the same question on the compatibility of y with w^2, w^3, . . ., and w^m, and obtain a *lower* bound on the success probability being

$$\Pr\{\text{success detecting } w^1 \text{ from } y\} = 1 - (m-1)\left(\frac{3}{4}\right)^n.$$

And by choosing $m \approx \frac{1}{100}\left(\frac{3}{4}\right)^n$, we have the success probability being approximately 99% with a non-zero code rate of

$$R = \frac{\#\text{ bits of information sent}}{\#\text{ channel use}} = \frac{\lg m}{n} = \frac{\lg\left(\frac{3}{4}\right)^n}{n} = 0.415,$$

where $\lg(\cdot) = \log_2(\cdot)$. This is a constant code rate scheme! □

Note that for the $\text{BEC}(0.5)$ channel, the theoretical limit is $R = 0.5$, which is the capacity of the channel. We can certainly improve further, but what is the problem with our analysis above that made

it not able to saturate the bound? The devils are in the details. The "bad events" have intersections, and the over-estimation of the error probability costs us some capacity. For example, the all erasure string $\mathcal{E}\mathcal{E}\dots\mathcal{E}$ is compatible with all $w^{i'}s$. A more careful analysis should be able to achieve the bound.

Remark 2.4

The above analysis utilizes random codes. But it is only good for theory, since the implementation of its encoder and decoder requires $\exp(\mathrm{O}(n))$ of calculations. In practice, we need *low complexity codes*. As we will see later, polar codes has the complexity of $\mathrm{O}(n \lg n)$.

As an end to this chapter on the noisy-channel theorem, let us also consider another example on BSC. What happens with BSC and random coding? Just on an intuitive level, one should be able to obserbe the following:

(a) The channel $\mathrm{BSC}(0.5)$ is a bad channel (though it is a really good random seed generator).

(b) Our analysis should use $\mathrm{BSC}\left(1/4\right)$ instead.

(c) The channel $\mathrm{BSC}\left(3/4\right) = \mathrm{BSC}\left(1/4\right)$ since one can simply flip the 0 and 1 after passing through the former to obtain the latter.

(d) Pick $w^i \in \{0,1\}^n$ randomly, send $w^1 = 00\dots0$.

(e) Obtain y, compute compatibility with w^1 by *Hamming distance*. One should obtain $d_{\mathrm{H}}(y, w^1) \approx \frac{n}{4}$.

(f) Compute the compatibility of y with other codewords, for example, $d_{\mathrm{H}}(y, w^2) \approx \frac{n}{2}$.

(g) We can set the decoding algorithm to be: If $d_{\mathrm{H}}(y, w^i) < \frac{n}{3}$, then we determine it to be w^i.

(h) The above decoding scheme fails when: (w.p. = with probability)

(1) $d_{\mathrm{H}}(y, w^1) > \frac{n}{3}$, w.p. $\leq \left(\frac{1}{4}\right)^{n/3}$ (define this error to be α), or

(2) $d_{\mathrm{H}}(y, w^{i\neq 1}) < \frac{n}{3}$, w.p. $\leq \left(\frac{1}{2}\right)^{2n/3}$ (define this error to be β).

(i) We can further bound the above via Chernoff bound: for $i = 1 \sim n$, let $F_i = 1$ if the ith bit of w^1 is flipped to obtain y, else it is 0, then

$$\alpha = \Pr\left\{F_1 + \dots + F_n \geq \frac{n}{3}\right\} = \Pr\left\{\frac{F_1 + \dots + F_n}{n} - \frac{1}{4} \geq \frac{1}{12}\right\} \leq \exp(-cn).$$

(j) Similarly, for $i = 1 \sim n$, let $X_i = 1$ if the ith bit of $w^{i\neq 1}$ is the same as y_i, else it is 0, then

$$\beta = \Pr\left\{X_1 + \dots + X_n \geq \frac{2n}{3}\right\} = \Pr\left\{\frac{X_1 + \dots + X_n}{n} - \frac{1}{2} \geq \frac{1}{6}\right\} \leq \exp(-c'n).$$

(k) The total error probability is hence upper bounded by $\exp(-cn) + (m-1)\exp(-c'n)$.

(l) Consider the number of codewords to be $m \approx \frac{1}{100}\mathrm{e}^{c'n}$, the code rate will hence be a constant $R \approx c' \lg e$, with success probability in decoding to be $\approx 99\%$. To obtain the exact value to the channel capacity (maximum possible code rate), one needs to solve a min-max program. Finally, with hard work, one should obtain that the channel capacity to be $C(p) = 1 + p\lg(p) + (1-p)\lg(1-p) = 1 - (\text{binary entropy function})$.

2.3. Polar Code over BEC

Consider again the binary erasure channel with a "magical decoder" as shown in the figure below.

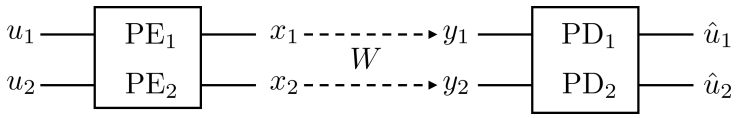


Figure 2.3. A single block for polar code encoding and decoding.

We have the polar encoders

$$\begin{aligned} x_1 &:= \mathrm{PE}_1(u_1, u_2) = u_1 \oplus u_2, \\ x_2 &:= \mathrm{PE}_2(u_1, u_2) = u_2. \end{aligned} \tag{2.3}$$

The channel is

$$W = \mathrm{BEC}(p),\ W(x) = \begin{cases} x & \text{w.p. } 1-p, \\ \mathcal{E} & \text{w.p. } p. \end{cases} \tag{2.4}$$

The polar decoders are

$$\begin{aligned} \hat{u}_1 := \mathrm{PD}_1(y_1, y_2) &= \begin{cases} y_1 \oplus y_2 & \text{if } y_1 \neq \mathcal{E} \text{ and } y_2 \neq \mathcal{E}, \\ \mathcal{E} & \text{otherwise;} \end{cases} \\ \hat{u}_2 := \mathrm{PD}_2(y_1, y_2, u_1) = \text{best guess of } u_2 &= \begin{cases} y_2 & \text{if } y_2 \neq \mathcal{E}, \\ y_1 \oplus u_1 & \text{if } y_1 \neq \mathcal{E} \text{ and } y_2 = \mathcal{E}, \\ \mathcal{E} & \text{otherwise.} \end{cases} \end{aligned} \tag{2.5}$$

Note that the u_1 used in PD_2 is *the real one*, and not an estimate. These are the magical decoders used.

2.3.1. Multi-Layered Polar Code

We can combine it to form a two-layer polar code as below:

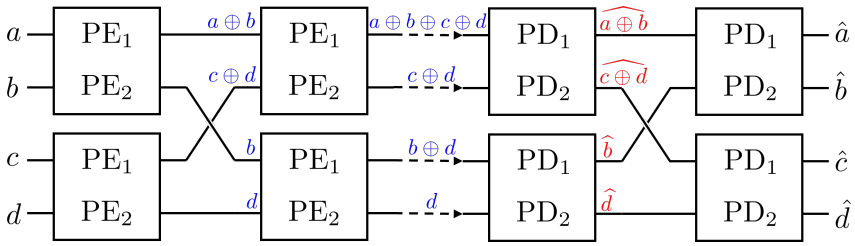


Figure 2.4. Two-layer polar code.

The variables with hats $\widehat{(\cdot)}$ once again represent estimates. On each transmission line, the variables being sent is also marked. We can further extend to a three-layer one, as shown in Figure 2.5. Amazingly, the structure of the encoder bears close resemblance to the *butterfly diagram* used in the circuitry realization of the fast Fourier transform (FFT)! For n bit inputs, the FFT algorithm has a computation complexity of $\mathrm{O}(n \lg n)$. Henceforth, the encoder structure of polar code has the operation complexity of $\mathrm{O}(n \lg n)$ as well. Random codes, on the other hand, though very useful for theoretical analysis, has a complexity of $\exp(O(n))$.

Finally, note that even though we draw the decoder also in the form of a butterfly diagram, it is not implemented so. Since PD_2 requires the *true value* to u_1 in calculating $\hat{u}_2$, the decoders are completed

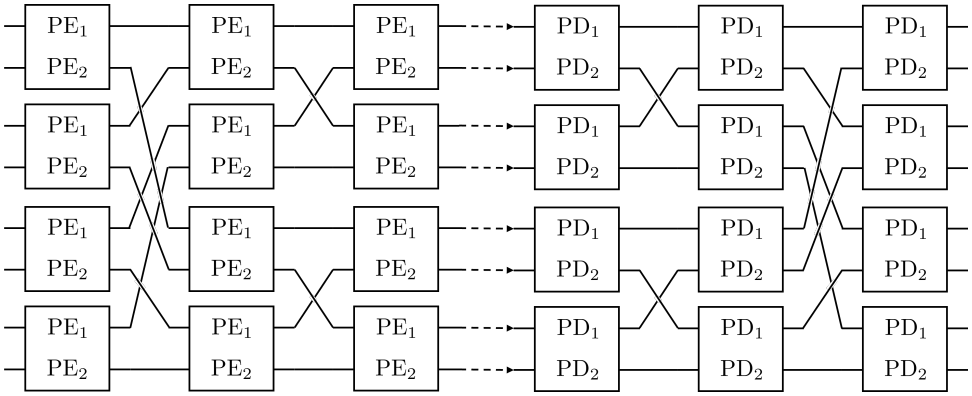


Figure 2.5. Three-layer polar code.

in an intertwined manner: The PD_1's are first calculated to obtain the required real values of u_2 for the PD_2's. As an exercise, one should check that for both the two-layer and three-layer polar code above, what is the order in which the PD_1's and PD_2's are performed? One can refer to the procedure shown in Figure 2.6 as a hint.

2.3.2. Polar Code on BSC

The same structure can be extended to $W = \mathrm{BSC}(p)$, the encoders can remain the same, however, the decoders need to be modified to a case-wise decision map. I.e., define the log-likelihood ratio function $\mathrm{LLR}(u|\theta) = \ln\big(\Pr\{u=0|\theta\}/\Pr\{u=1|\theta\}\big)$,

$$\begin{aligned} \mathrm{PD}_1(y_1, y_2) &= \left[\phi\left(u_1 |\, y_1, y_2\right) \underset{1}{\overset{0}{\gtrless}} 0\right], \\ \mathrm{PD}_2(y_1, y_2, u_1) &= \left[\phi\left(u_2 |\, y_1, y_2, u_1\right) \underset{1}{\overset{0}{\gtrless}} 0\right]. \end{aligned} \tag{2.6}$$

The value of ϕ needs to be determined case-wise. Even though it is more complicated than the BEC case, it still only requires finitely many steps, and hence the polar code works on BSC even in practical cases.

2.4. Polar Decoder

4 Mar.

As was mentioned at the end of the last section, the decoding procedure to the polar code is different from that of the encoder, and one has to go backwards and forwards between layers. The following discussion focuses on multi-layered polar codes with encoding and decoding boxes of size 2×2. However, the size of encoding and decoding boxes can be 2×2, 3×3, or even $\ell \times \ell$. Please refer to Fig. 2, 3, and 4 of the paper "Polar Codes' Simplicity, Random Codes' Durability" [WD21] for a more general and rigorous description of the below analysis.

See the decoding procedure Figure 2.6 for a 2-layer polar code. Going from (a) to (h), the decoders sequentially went through the PD_1's and PD_2's. Note that since PD_2's require the knowledge of u_1's, the box on their left will send the values by back propagating through PE_1's and PE_2's, here denoted by the teal arrows in the figure above. Moreover, we only treat the values obtained at the final column as the *true* u_1's.

Similar procedures can be drawn for 3-layer polar codes or more, and is left as an exercise for the readers.

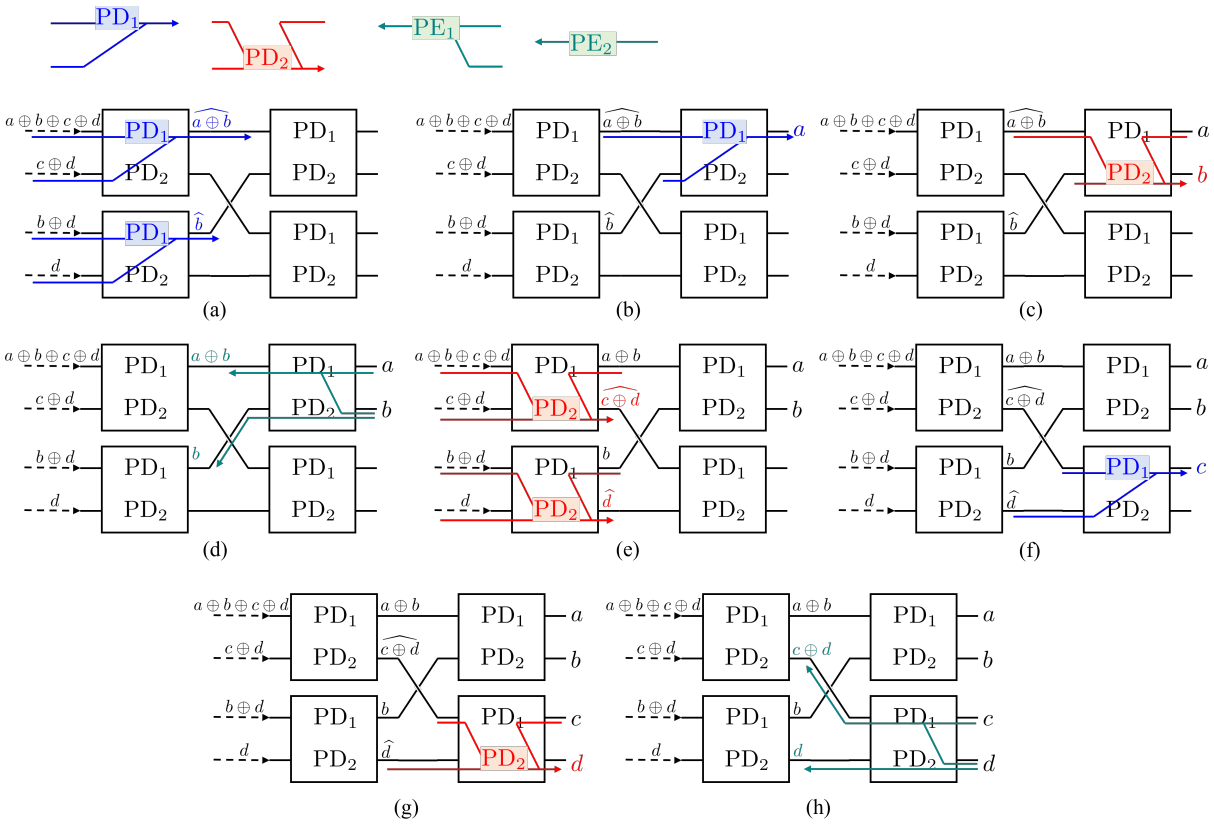


Figure 2.6. Decoding procedure of a two-layer polar code.

The decoding of PD_2 requires the back-propagation of true values via the polar encoders, however, what if the obtained true values are the error $\mathcal{E}$? Recall from Remark 1.3, it is stated that we can agree on sending 0 over *bad sub-channels* that are more erroneous. Consider having a genie[1] that tells us, for example, there are 4 good and 4 bad sub-channels in the 3-layer polar code as in Figure 2.7[2]. We can send information bits (IB) over the good sub-channels, and send frozen bits (FB) over the bad sub-channels. As mentioned, the frozen bits are chosen to be 0.

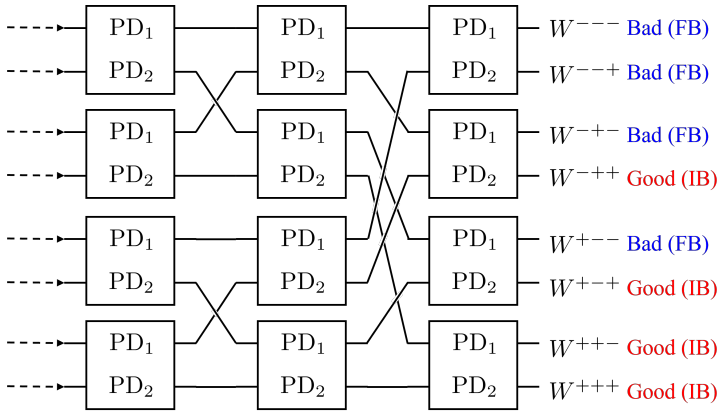


Figure 2.7. Genie-aided three-layer polar code.

During decoding, the genie will simply send back 0 for the frozen bits as their true values; the genie will ask the decoders to figure out the true values to the information bits. One should ask themselves:

[1] A genie is chosen since Arıkan is Turkish.

[2] What is the channel capacity implied by this distribution of good and bad channels? And no, this footnote is not an exponent.

why is the assignment of good and bad not *continuous*? How are they determined?

Let us analyze the error probability under this scenario over BEC. Since the frozen bits can never be wrong, everything that could go wrong is when the genie gets an information bit wrong, and we simply throw out the entire block of code. The error probability will hence be: denote the polarized channels by $W^4 = W^{-++}$, $W^6 = W^{+-+}$, $W^7 = W^{++-}$, and $W^8 = W^{+++}$,

$$\begin{aligned} P_e = &\Pr\{W^4 \text{ is wrong}\} + \Pr\{W^6 \text{ is wrong}\,;\, W^4 \text{ is right}\} \\ &+ \Pr\{W^7 \text{ is wrong}\,;\, W^4, W^6 \text{ are right}\} + \Pr\{W^8 \text{ is wrong}\,;\, W^4, W^6, W^7 \text{ are right}\}. \end{aligned}$$

If W^4 makes mistake, then everything else doesn't matter; if W^4 is correct but W^6 is wrong, then everything else doesn't matter; if W^4 and W^6 is correct but W^7 is wrong, then everything doesn't matter; if only W^8 is wrong, then the block is still discarded. This is also a result from the fact that the polar code is decoded sequentially. Note that from Equation (1.9) and Equation (1.10), we have

$$P_e = \left((2x - x^2)^2\right)^2 + (2x^2 - x^4)^2 + (2x^4 - x^8) + (x^8).$$

The above analysis evidently shows why we choose W^{-++} as a good sub-channel over W^{+--}.

Remark 2.5

We choose W^{-++} over W^{+--} due to the fact that

$$P_e(W^{-++}) = \left((2x - x^2)^2\right)^2 \leq 2(2x - x^2) - (2x - x^2)^2 = P_e(W^{+--}) \qquad \forall\, x \in [0, 1], \quad (2.7)$$

with equality satisfied only at the extremals 0 and 1. See the illustration below for a clearer idea of the inequality.

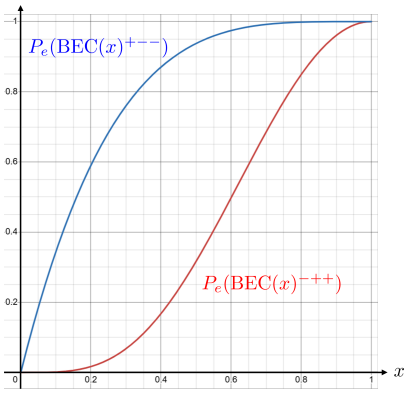


Figure 2.8. Ordering of the polarized error probabilities.

For 2-, 3-, and 4-layer polar codes over $\mathrm{BEC}(x)$, a total ordering of the sub-channels via their error probability exists.

However, for block length greater than or equal to 32, intersections start to appear on the different error probability curves between each sub-channels. Hence, in this case, only a *partial ordering* between the sub-channels exists.

Partial-order theory is useful in deciding which sub-channel is better than which, important for the allocation of information bits, especially in the cases of long block lengths.

2.5. Martingale

In this section, we take a detour and introduce the idea of a *martingale*. Martingale is a special kind of random sequence that provides many useful tools from stochastic calculus in aiding our analysis of the performance of polar codes. For a proof and a more rigorous description of all the theorems below, one can refer to the book "Probability: Theory and Examples" [Dur19] by Rick Durrett on graduate-level probability theory.

2.5.1. A Brief Overview of Martingale Theory

Definition 2.6: Martingale

A sequence of random variables $\{M_t\}_t$ is termed a martingale if

$$\mathbb{E}\left[|M_t|\right] < \infty, \qquad \mathbb{E}[M_{t+1}|M_t] = M_t. \tag{2.8}$$

Some examples of a martingale is shown below:

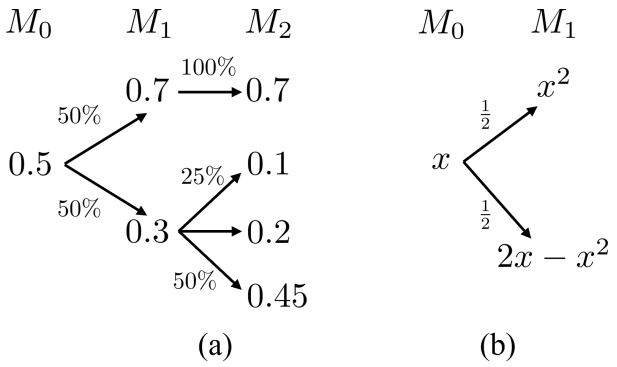


Figure 2.9. Illustration of martingales.

One should observe that the last example given above is exactly that of the division of a BEC!

Remark 2.7

The word "martingale" comes from J. Doob, the founding father of the theory of martingales. It is inspired a piece of horse gear: *a long leather belt that bifurcates, at one end, into two strips of the same length.* Excerpted from "The origin of the word martingale," by Roger Mansuy.

Let us now introduce the readers to a few useful theorems from the theory of martingales.

Theorem 2.8: Martingale Convergence Theorem

For a martingale $\{M_t\}$ that is bounded, i.e.,$|M_t| < \infty$[a], there exists a random variable M_∞ such that $M_t \xrightarrow{\text{a.s.}} M_\infty$ as $t \to \infty$.

[a]The absolutely boundedness is a really strong constraint, which can be relaxed to bounded absolute expectation.

Here we shall give a more illustrative example showing that the theorem above is reasonable and should instinctively hold:

Example: Define $\{M_t\}$ as the *polarization process* of polar code over BEC. Let $M_0 = x$, with

$$M_{t+1} = \begin{cases} M_t^2 & \text{w.p. } \frac{1}{2}, \\ 2M_t - M_t^2 & \text{w.p. } \frac{1}{2}. \end{cases} \tag{2.9}$$

By the theorem above, M_∞ should exist, and we further claim that $M_\infty \in \{0, 1\}$.

If $\Pr\left\{M_\infty = x \neq \{0, 1\}\right\} \geq 0$, then for all $t \geq T$, there exists a $T \gg 0$ such that $M_t \in [x - \varepsilon, x + \varepsilon]$ for some fixed $\varepsilon > 0$. However, M_{T+1}, there are 50% chances that either of the following happens:

$$\begin{aligned} M_T^2 &\leq (x + \varepsilon)^2 < x - \varepsilon, \\ 2M_T - M_T^2 &\geq 2(x - \varepsilon) - (x - \varepsilon)^2 > x + \varepsilon, \end{aligned}$$

both leading to a contradiction by choosing an ε small enough. The claim that $M_\infty \in \{0, 1\}$ is thus proven by contradiction. □

Example: Again consider $\{M_t\}$ as the polarization process of polar code over $\mathrm{BEC}(x)$. We have

$$\begin{aligned} \Pr\{M_\infty = 1\} = \mathbb{E}[M_\infty] &= \mathbb{E}\left[\lim_{t\to\infty} M_t\right] \\ &= \lim_{t\to\infty} \mathbb{E}[M_t] \\ &= \lim_{t\to\infty} \mathbb{E}[M_{t-1}] = \cdots = \lim_{t\to\infty} \mathbb{E}[M_0] = x. \end{aligned} \tag{2.10}$$

The validity of the exchange of the limit and expectation in Equation (2.10) is guaranteed by the convergence of the martingale in the limit.

Similarly, we have $\Pr\{M_\infty = 0\} = 1 - x$.

Further consider a random $s \in \{+, -\}^n$, we have

$$\begin{aligned} &\Pr\{\mathrm{BEC}(x)^s \text{ is close to } \mathrm{BEC}(1)\} = x, \\ &\Pr\{\mathrm{BEC}(x)^s \text{ is close to } \mathrm{BEC}(0)\} = 1 - x. \end{aligned} \tag{2.11}$$

The number of $s \in \{+, -\}^n$ such that $\mathrm{BEC}(x)^s$ is good is $2^n(1 - x - \varepsilon)$, corresponding to the number of information bits; the rest of $2^n(x + \varepsilon)$ are bad, corresponding to the frozen bits. □

Definition 2.9: Stopping Time

An integer random variable σ is called a *stopping time* if whether or not $\sigma \leq t$ depends only on $M_1, \ldots, M_t$, and not on $M_{t+1}, M_{t+1}, \ldots$.

Theorem 2.10: Optional Stopping Time

For a martingale $\{M_t\}$ with stopping time σ, we have

$$\mathbb{E}[M_\sigma] = \mathbb{E}[M_0]. \tag{2.12}$$

The proof can be directly seen from the example above.

Example: Let us consider a martingale $\{M_t\}$ with $M_t \geq 0$. Let $M_0 = 0.001$ be deterministic. Define its stopping time as $\sigma := \min\{\, t \mid M_t \geq 1 \,\} \cup \{N\}$ where N is very large. In the context of a stock market, this definition of stopping time corresponds to ones strategy of selling out when reaching a

gain of profit larger than 1 with a small initial investment. Let us bound the success probability of such strategy.

$$\begin{aligned} M_0 \equiv \mathbb{E}[M_\sigma] &= \mathbb{E}\left[M_\sigma \cdot \mathbb{1}\{M_\sigma \geq 1\}\right] + \mathbb{E}\left[M_N \cdot \mathbb{1}\{M_\sigma \text{ is never larger than } 1\}\right] \\ &\geq \mathbb{E}\left[M_\sigma \cdot \mathbb{1}\{M_\sigma \geq 1\}\right] \\ &\geq \Pr\{M_\sigma \geq 1\}. \end{aligned}$$

Henceforth, we have $\Pr\{M_\sigma \geq 1\} \leq M_0 = 0.1\%$. The probability of gaining huge profit from small investment in this model is small. □

This result above can be generalized via the Markov inequality: given a martingale $M_t \geq 0$, we have $\Pr\{M_\sigma \geq a\} \leq \mathbb{E}[M_\sigma]/a = \mathbb{E}[M_0]/a$. This is known as Doobs's maximum inequality.

Theorem 2.11: Doob's Maximum Inequality

Let the optional stopping time for a martingale $\{M_t\}$ be

$$\sigma : \text{ minimum } t \text{ that } M_t \text{ reaches } A. \tag{2.13}$$

Then since

$$M_\sigma \geq A \Leftrightarrow \max_t M_t \geq A, \tag{2.14}$$

we have

$$\Pr\{\max_t M_t \geq A\} = \Pr\{M_\sigma \geq A\} \leq \frac{M_0}{A}. \tag{2.15}$$

Remark 2.12

Doob single-handedly translated the theory of martingale from a competing theory of description of probability to the now widely-used measure-theoretical probability theory.

Remark 2.13

The applications to martingale often includes stochastic calculus and stock market analysis. The concept of stopping time can be, as an example, interpreted as the time one sells out their stocks.

Moreover, two other kinds of martingales exist:

(a) Supermartingale: $\mathbb{E}[M_{t+1}|M_t] \leq M_t$;

(b) Submartingale: $\mathbb{E}[M_{t+1}|M_t] \geq M_t$.

We can proof the (Doob's) martingale convergence theorem from the Doob's maximum inequality:

Proof. Assume that we are considering a bounded martingale $0 \leq M_t \leq 1$. We claim that if M_t does not converge, then there exists $0 < a < b < 1$ such that M_t oscillates infinitely many times between a and b. See the figure below:

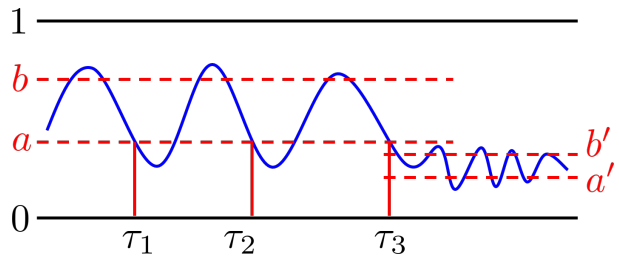


Figure 2.10. Illustration of the martingale convergence theorem.

If the martingale oscillates outside $[a, b]$, one can find another interval $[a', b']$ in which the martingale will finally reside in. Denote the time when $M_t = a$ as τ's, we have, from the maximum inequality:

$$\Pr\{M_{\tau+t} \geq b | M_\tau = a\} \leq \frac{a}{b} < 1. \tag{2.16}$$

Since the oscillation happens infinitely many times, we have the total probability $(\frac{a}{b})^n \to 0$. Thus, if the sequence doesn't bound between 2 numbers, it must be *almost surely* convergent. ■

2.5.2. McDiarmid's Inequality

Recall from large deviation theory that we have, for i.i.d. and bounded random variables $X_1, \ldots, X_n$,

$$\Pr\left\{\frac{X_1 + \cdots + X_n}{n} - \mathbb{E}[X_1] \geq \varepsilon\right\} \leq \exp(-c\varepsilon^2 n) \tag{2.17}$$

for some constant c. This is known as the Chernoff bound or Hoeffding's inequality.

A similar version for martingale exists:

Theorem 2.14: Azuma's Inequality

For a martingale $\{M_t\}$ with bounded variation $|M_{t+1} - M_t| \leq C$, we have

$$\Pr\left\{\frac{M_n}{n} - M_0 > \varepsilon\right\} < \exp(-c\varepsilon^2 n) \tag{2.18}$$

for some constant c.

This inequality can be used to show the capacity-achieving property of polar codes. Moreover, by defining M_t as

$$M_t := X_1 + \cdots + X_t - t \cdot \mathbb{E}[X_1], \tag{2.19}$$

we can see that it is indeed a martingale:

$$\mathbb{E}[M_{t+1} | M_t] = M_t + \mathbb{E}\left[X_{t+1} - \mathbb{E}[X_1]\right] = M_t.$$

We readily obtained the Hoeffding's inequality from Azuma's inequality.

This result can, obviously, be further generalized. Let us define some similar notions needed for the description of the following theorem.

Definition 2.15: Bounded Variation

A function $f : [0, 1]^n \to \mathbb{R}$ is of bounded variation if there exists a C_i such that

$$\left|f(a_{1:i-1}, b_i, a_{i+1:n}) - f(a_{1:i-1}, c_i, a_{i+1:n})\right| < C_i \tag{2.20}$$

is satisfied for all possible i, b_i, and c_i.

Definition 2.16: Doob's Martingale

For an n-variable function f, the following sequence is a martingale:

$$M_0 = \mathbb{E}_Y[f(Y_{1:n})], \tag{2.21}$$
$$M_t = \mathbb{E}_{Y_{t+1:n}}[f(X_{1:t}, Y_{t+1:n})], \tag{2.22}$$
$$M_n = f(X_1, \ldots, X_n). \tag{2.23}$$

Theorem 2.17: McDiarmid's Inequality

For f of bounded variation, there exists L such that

$$\Pr\{f(X_{1:n}) - L > \varepsilon\} < \exp(-c\varepsilon^2 n). \tag{2.24}$$

The McDiarmid's inequality is often used in statistical learning, where $X_{1:n}$ can be seen as a random data set obtained. All in all, both Azuma's and McDiarmid's inequality show the concentration of bounded-variational martingales or functions, respectively.

2.5.3. Analyzing Polar Code over BEC via Martingale

This subsection utilizes our newly obtained skills from the martingale theory to apply to an example of polar code over BEC.

Example: Let us construct a $2n$-layer polar code (with block length being $2^{2n} = 4^n$) over $\mathrm{BEC}(x)$ with x small. Let us define the set

$$\mathcal{A} := \left\{ s \in \{+,-\}^n \,\middle|\, P_e\left(\mathrm{BEC}(x)^s\right) < 2^{-10} \right\}.$$

For a fixed sub-channel $s \in \mathcal{A}$, we further pick a random $t \in \{+,-\}^n$. One should be immediately be curious of the bound to

$$\Pr\left\{ P_e\left(\mathrm{BEC}(x)^{st_{1:j}}\right) \geq 2^{-6} \right\},$$

where $st_{1:j}$ is the concatenation of the string s with $t_{1:j}$, the first to jth element of t.

By the optional stopping time theorem in the previous subsection, we have that

$$\Pr\left\{ P_e\left(\mathrm{BEC}(x)^{st_{1:j}}\right) \geq 2^{-6} \right\} \leq 2^{-10} \cdot 2^6 = 2^{-4} \quad \text{for some stopping time } j.$$

Next, let

$$\mathcal{B} := \left\{ st \in \{+,-\}^{2n} \,\middle|\, s \in \mathcal{A},\ P_e\left(\mathrm{BEC}(x)^{st} \leq 2^{-6}\right) \text{ for fixed } s \right\}.$$

What will $|\mathcal{B}| \,/\, \left(2^n|\mathcal{A}|\right)$ be? It is easy to see that

$$\frac{|\mathcal{B}|}{2^n|\mathcal{A}|} \geq 1 - 2^{-4}.$$

For an $st \in \mathcal{B}$, every new $+$ sign sends x to x^2; and every new $-$ sign send x to $2x$ (since x is assumed to be small, the term $-x^2$ is omitted). One further asks that, for example, if t contains at least $n/3$ $+$'s and at most $2n/3$ $-$'s, what is the error probability $P_e\left(\mathrm{BEC}(x)^{st}\right)$?

We can bound it by considering the variables

$$\begin{aligned} a_0 &:= \lg\left(-\lg P_e\left(\mathrm{BEC}(x)^s\right)\right), \\ a_j &:= \lg\left(-\lg P_e\left(\mathrm{BEC}(x)^{st_{1:j}}\right)\right), \end{aligned} \tag{2.25}$$

satisfying the following recursion relation:

$$a_{j+1} = \begin{cases} a_j + 1 & \text{if } t_{j+1} = +, \\ a_j - \varepsilon_k & \text{if } t_{j+1} = -. \end{cases} \tag{2.26}$$

Then we have that $a_n \geq \frac{n}{3} - (\max_k \varepsilon_k)\frac{2n}{3} = rn$ for a constant r. Thus, for an $st \in \mathcal{B}$, the error probability has a bound

$$P_e\left(\mathrm{BEC}(x)^{st}\right) < 2^{-2^{rn}} \text{ w.p. } \Pr\left\{\text{a random } t \text{ has } \geq \frac{n}{3} \text{ of } +\text{'s}\right\} \geq 1 - \mathrm{e}^{-cn},$$

for some constant c derived from the Chernoff bound. The Chernoff bound is applicable since $\frac{n}{3} \leq \frac{n}{2} = \mathbb{E}[\#\text{ of } +\text{'s in a random } t]$. Lastly, the number of $st \in \{+,-\}^{2n}$ subject to $P_e\left(\mathrm{BEC}(x)^{st}\right) < 2^{-2^{rn}}$ is greater than $2^n|\mathcal{A}| - 2^{2n}\left(2^{-4} - \exp(-cn)\right)$. □

2.6. Polar Code over BSC and BMSC

2.6.1. Binary Memoryless Symmetric Channel

Binary memoryless symmetric channel (BMSC) is a generalization to BSC.

Definition 2.18: BMSC

A channel $W : \{0,1\} \to \mathcal{Y}$ with an involution $\iota : \mathcal{Y} \to \mathcal{Y}$ (an involution is a permutation that satisfies $\iota^2 = \mathrm{identity}$) is called a BMSC if

$$W(y|0) = W(\iota(y)|1). \tag{2.27}$$

Some prime examples of a BMSC are BSC and BEC. The figure below shows another example of a BMSC and its involution.

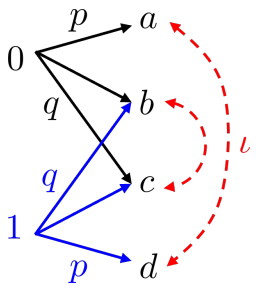


Figure 2.11. Involution of a BMSC.

A more general construction of a BMSC channel is shown as follow:

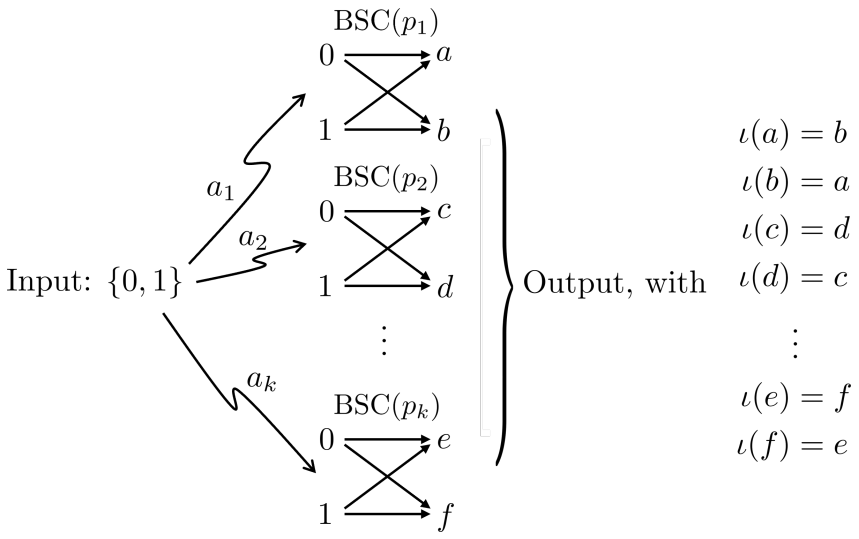


Figure 2.12. Illustration of a general BMSC.

By probabilistically assigning a channel $\mathrm{BSC}(p_i)$ with probability a_i, we can create seemingly any possible BMSC using BSCs. Note that the involution relations are needed.

In general, we have the following theorem.

Theorem 2.19: BMSC as a Compound of BSCs

Given $0 < p_1, \dots, p_k \leq \frac{1}{2}$ and $0 < a_1, \dots, a_k < 1$ satisfying $a_1 + \cdots + a_k = 1$, the compound channel

$$\sum_{i=1}^{k} a_i \cdot \mathrm{BSC}(p_i) \tag{2.28}$$

is a BMSC. Similarly, any BMSC is equivalent to a decomposition $\sum_{i=1}^{k} a_i \cdot \mathrm{BSC}(p_i)$.

The convex combination of channels corresponds to a probabilistic mixture of the channels.

Remark 2.20

The decomposition of a BMSC into BSCs holds true even for continuous channels such as an AWGN, where we can replace the summation by an integral:

$$\mathrm{AWGN}(\sigma) = \int A_\sigma(p)\ \mathrm{BSC}(p)\ \mathrm{d}p.$$

For a more general construction and detailed descriptions on BMSC, please refer to the book "Modern Coding Theory" [RU07] and the paper "Channel Polarization through the Lens of Blackwell Measures" [GR20].

2.6.2. The $+$ and $-$ Sub-Channels

From the above, we know that we can obtain analytical results of any BMSC by analyzing BSCs. In this subsection, we aim to understand the meaning to $\mathrm{BSC}(p)^+$, $\sum a_i\,\mathrm{BSC}(p_i)^+$, $\mathrm{BSC}(p)^-$, and $\sum a_i\,\mathrm{BSC}(p_i)^-$. The paper "Channel Polarization through the Lens of Blackwell Measures" [GR20] again provided a detailed and expansive discussion of the following topic.

The most general language that we can use the discussion over the *parallel* and *serial combination* of BSCs, which are denoted by $\mathrm{BSC}(p) \circledast \mathrm{BSC}(q)$ and $\mathrm{BSC}(p) \boxtimes \mathrm{BSC}(q)$, respectively.

Parallel Combination: The parallel combination channel $\mathrm{BSC}(p) \circledast \mathrm{BSC}(q)$ describes what we know about $x \in \{0, 1\}$ given the outputs $\mathrm{BSC}(p)(x)$ and $\mathrm{BSC}(q)(x)$. The diagram of the channel is as shown below:

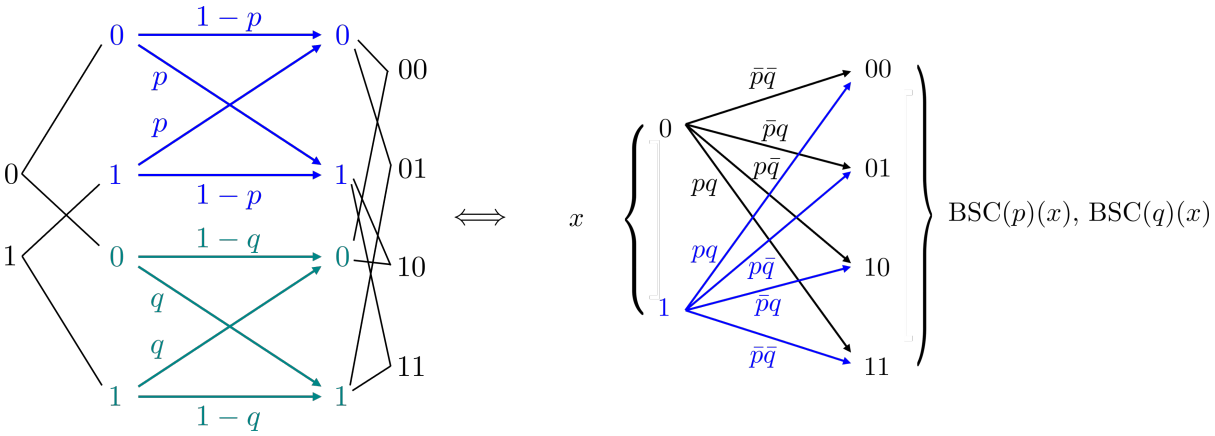


Figure 2.13. Parallel combination of $\mathrm{BSC}(p) \circledast \mathrm{BSC}(q)$. An input is sent through two parallel channels, with the output compared in parallel to obtain information about the input.

By introducing the notation $\bar{x} = 1 - x$, we have the involution $\iota(x) = \bar{x}$. Furthermore, by an abuse of notation, we also have $\bar{p} = 1 - p$ and $\bar{q} = 1 - q$. For an input of x, we have the outputs

$$\begin{cases} xx & \text{w.p. } \bar{p}\bar{q}, \\ \bar{x}\bar{x} & \text{w.p. } pq, \\ x\bar{x} & \text{w.p. } \bar{p}q, \\ \bar{x}x & \text{w.p. } p\bar{q}. \end{cases}$$

Henceforth, by considering the involute pairs of $(xx, \bar{x}\bar{x})$ and $(\bar{x}x, x\bar{x})$ as the individual BSC components (since $\iota(xx) = \bar{x}\bar{x}$ and $\iota(\bar{x}x) = x\bar{x}$), we can view the combined channel as

$$\mathrm{BSC}(p) \circledast \mathrm{BSC}(q) = (pq + \bar{p}\bar{q}) \cdot \mathrm{BSC}\left(\frac{pq}{pq + \bar{p}\bar{q}}\right) + (p\bar{q} + \bar{p}q) \cdot \mathrm{BSC}\left(\frac{p\bar{q}}{p\bar{q} + \bar{p}q}\right). \tag{2.29}$$

The result can be further extended by linearity to

$$\left(\sum_i a_i \cdot \mathrm{BSC}(p_i)\right) \circledast \left(\sum_i b_i \cdot \mathrm{BSC}(q_i)\right) = \sum_{i,j} a_i b_j \cdot \mathrm{BSC}(p_i) \circledast \mathrm{BSC}(q_j). \tag{2.30}$$

Serial Combination: The serial combination channel $\mathrm{BSC}(p) \boxtimes \mathrm{BSC}(q)$ describes what we know about $x + y$ (mod 2 addition) given the outputs $\mathrm{BSC}(p)(x)$ and $\mathrm{BSC}(q)(y)$. The diagram of the channel is as shown below:

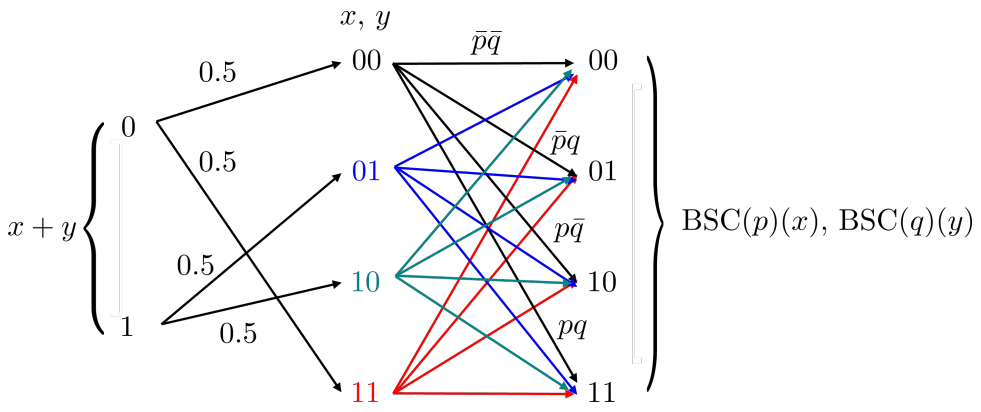


Figure 2.14. Serial combination of channels. An input is sent through two channels with the output summed together. For the two channels both being BSC, the total effect is equivalent to serial combining the two channels where the output of the first becomes the input of the second.

The combined channel law is

$$W(00|0) = W(11|0) = W(01|1) = W(10|1) = \frac{1}{2}(\bar{p}\bar{q} + pq),$$
$$W(01|0) = W(10|0) = W(00|1) = W(11|1) = \frac{1}{2}(\bar{p}q + p\bar{q}).$$

By considering the super-symbols of involute pairs $A = (00, 11)$ and $B = (01, 10)$ since the symbols in each super-symbol provides the same information over what $x + y$ is, one can check that the above channel is equivalent to the channel below.

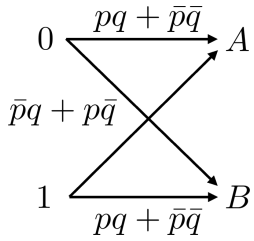


Figure 2.15. The serial combination of BSC's is still a BSC.

Henceforth, we can view the combined channel as

$$\mathrm{BSC}(p) \boxtimes \mathrm{BSC}(q) = \mathrm{BSC}(pq + \bar{p}\bar{q}). \tag{2.31}$$

The result can be further extended by linearity to

$$\left(\sum_i a_i \cdot \mathrm{BSC}(p_i)\right) \boxtimes \left(\sum_i b_i \cdot \mathrm{BSC}(q_i)\right) = \sum_{i,j} a_i b_j \cdot \mathrm{BSC}(p_i) \boxtimes \mathrm{BSC}(q_j). \tag{2.32}$$

Theorem 2.21: Parallel and Serial Combination of BSC

For probabilities p and q, we have

$$\mathrm{BSC}(p) \circledast \mathrm{BSC}(q) = (pq + \bar{p}\bar{q}) \cdot \mathrm{BSC}\left(\frac{pq}{pq + \bar{p}\bar{q}}\right) + (p\bar{q} + \bar{p}q) \cdot \mathrm{BSC}\left(\frac{p\bar{q}}{p\bar{q} + \bar{p}q}\right) \tag{2.33}$$

$$\mathrm{BSC}(p) \boxtimes \mathrm{BSC}(q) = \mathrm{BSC}(pq + \bar{p}\bar{q}). \tag{2.34}$$

Note that we can clarify the terms mentioned at the start of this subsection by the following theorem:

Theorem 2.22: The $+$ and $-$ Sub-Channels for BSC

For sub-channels of a BSC, we have

$$\mathrm{BSC}(p)^{+} = \mathrm{BSC}(p) \circledast \mathrm{BSC}(p), \tag{2.35}$$
$$\mathrm{BSC}(p)^{-} = \mathrm{BSC}(p) \boxtimes \mathrm{BSC}(p). \tag{2.36}$$

Lastly, note that for the special case where we are only considering the serial combination of BSCs, and with super-symbols in mind, Figure 2.14 can be transformed into Figure 2.16.

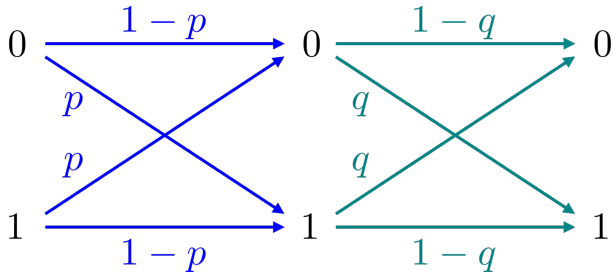


Figure 2.16. Serial combination of $\mathrm{BSC}(p) \boxtimes BSC(q)$.

By the referenced provided (using the tools derived from the *Blackwell measures*), we have the following general result.

Theorem 2.23: Polar Transform

Given two BMSCs $(W_1, \mathcal{Y}_1)$ and $(W_2, \mathcal{Y}_2)$, the *polar transform* maps them into another pair of BMSCs $W_1 \circledast W_2$ and $W_1 \boxtimes W_2$ defined as

$$(W_1 \circledast W_2)(y_1, y_2, u|x) := \frac{1}{2} W_1(y_1|x+u) W_2(y_2|x), \tag{2.37}$$
$$(W_1 \boxtimes W_2)(y_1, y_2|x) := \frac{1}{2} \sum_{u \in \{0,1\}} W_1(y_1|x+u) W_2(y_2|u) \tag{2.38}$$

for all $y_1 \in \mathcal{Y}_1$, $y_2 \in \mathcal{Y}_2$, and $x, u \in \{0, 1\}$. The former sub-channel is an improved channel, while the latter is a weaker channel. Especially when $W_1 = W_2 = W$, we have

$$W^{+} = W \circledast W \text{ and } W^{-} = W \boxtimes W, \tag{2.39}$$

where the equality, if written rigorously, should be an *equivalence* by the *Blackwell measure*.

It can be readily seen that under the polar coding scheme: by setting $u_1 = u$ and $u_2 = x$, with u uniformly distributed, Equation (2.37) represents the probability of inputting (u, x) and having a channel output of (y_1, y_2) – if $W_1 = W_2 = W$, and we group certain symbols into a super-symbol by considering how the polar decoder decodes, this is indeed the channel W^{+}.

Similarly, by setting $u_1 = x$ and $u_2 = u$, with u uniformly distributed, Equation (2.38) represents the probability of inputting (x, u) and having a channel output of (y_1, y_2) – if $W_1 = W_2 = W$, and

we group certain symbols into a super-symbol by considering how the polar decoder decodes, this is indeed the channel W^-.

Let us demonstrate the theorem above by the following two examples:

Example: It can be readily checked that the following equation holds:

$$\mathrm{BEC}(x) = \bar{x} \cdot \mathrm{BSC}(1) + x \cdot \mathrm{BSC}(1/2). \tag{2.40}$$

Henceforth, we have the two sub-channels calculated to be

$$\begin{aligned}
\mathrm{BEC}(x)^+ &= \big(\bar{x} \cdot \mathrm{BSC}(1) + x \cdot \mathrm{BSC}(1/2)\big) \circledast \big(\bar{x} \cdot \mathrm{BSC}(1) + x \cdot \mathrm{BSC}(1/2)\big) \\
&= \bar{x}^2 \cdot \mathrm{BSC}(1)^+ + x^2 \cdot \mathrm{BSC}(1/2)^+ + 2\bar{x}x \cdot \mathrm{BSC}(1) \circledast \mathrm{BSC}(1/2) \\
&= (1 - 2x + x^2) \cdot \mathrm{BSC}(1) + x^2 \cdot \mathrm{BSC}(1/2) + (2x - 2x^2) \cdot \mathrm{BSC}(1) \\
&= (1 - x^2) \cdot \mathrm{BSC}(1) + x^2 \cdot \mathrm{BSC}(1/2) \\
&= \mathrm{BEC}(x^2),
\end{aligned}$$

$$\begin{aligned}
\mathrm{BEC}(x)^- &= \big(\bar{x} \cdot \mathrm{BSC}(1) + x \cdot \mathrm{BSC}(1/2)\big) \boxtimes \big(\bar{x} \cdot \mathrm{BSC}(1) + x \cdot \mathrm{BSC}(1/2)\big) \\
&= \bar{x}^2 \cdot \mathrm{BSC}(1)^- + x^2 \cdot \mathrm{BSC}(1/2)^- + 2\bar{x}x \cdot \mathrm{BSC}(1) \boxtimes \mathrm{BSC}(1/2) \\
&= (1 - 2x + x^2) \cdot \mathrm{BSC}(1) + x^2 \cdot \mathrm{BSC}(1/2) + (2x - 2x^2) \cdot \mathrm{BSC}(1/2) \\
&= (1 - 2x + x^2) \cdot \mathrm{BSC}(1) + (2x - x^2) \cdot \mathrm{BSC}(1/2) \\
&= \mathrm{BEC}(2x - x^2).
\end{aligned}$$

Thus, we have proven the sub-channel laws of polar code on BEC observed in the first chapter, where $\circledast$ results in a better sub-channel, and $\boxtimes$ results in a worse sub-channel. □

Example: The sub-channels to BEC can also be directly calculated from the polar transformation. Note that $W_1 = W_2 = \mathrm{BEC}(x)$ and $\mathcal{Y}_1 = \mathcal{Y}_2 = \{0, 1, \mathcal{E}\}$. Hence, by abbreviating $W \circledast W$ as W_p, we have for parallel combination

$$\begin{aligned}
\frac{1}{2}\bar{x}^2 &= W_p(0,0,0|0) = W_p(1,1,0|1) = W_p(1,0,1|0) = W_p(0,1,1|1), \\
\frac{1}{2}\bar{x}x &= W_p(\mathcal{E},0,0|0) = W_p(\mathcal{E},1,0|1) = W_p(\mathcal{E},0,1|0) = W_p(\mathcal{E},1,1|1) \\
&= W_p(0,\mathcal{E},0|0) = W_p(1,\mathcal{E},0|1) = W_p(1,\mathcal{E},1|0) = W_p(0,\mathcal{E},1|1), \\
\frac{1}{2}x^2 &= W_p(\mathcal{E},\mathcal{E},0|0) = W_p(\mathcal{E},\mathcal{E},0|1) = W_p(\mathcal{E},\mathcal{E},1|0) = W_p(\mathcal{E},\mathcal{E},1|1),
\end{aligned}$$

all the other cases have a value of 0. Consider the following disjoint sets of possible outputs,

$$\begin{aligned}
\mathcal{S}_0 &= \{(0,0,0),(1,0,1),(\mathcal{E},0,0),(0,\mathcal{E},0),(\mathcal{E},0,1),(1,\mathcal{E},1)\}, \\
\mathcal{S}_1 &= \{(1,1,0),(0,1,1),(\mathcal{E},1,0),(1,\mathcal{E},0),(\mathcal{E},1,1),(0,\mathcal{E},1)\}, \\
\mathcal{S}_\mathcal{E} &= \{(\mathcal{E},\mathcal{E},0),(\mathcal{E},\mathcal{E},1)\},
\end{aligned}$$

they can be viewed as super-symbols, where the polar decoder will decode $\mathcal{S}_y$ into y. Then we can calculate the probability of each set happening:

$$\begin{aligned}
1 - x^2 &= \Pr\{\mathcal{S}_0|0\} = \Pr\{\mathcal{S}_1|1\}, \\
0 &= \Pr\{\mathcal{S}_1|0\} = \Pr\{\mathcal{S}_0|1\}, \\
x^2 &= \Pr\{\mathcal{S}_\mathcal{E}|0\} = \Pr\{\mathcal{S}_\mathcal{E}|1\}.
\end{aligned}$$

Hence, we have that $\mathrm{BEC}(x)^+ = \mathrm{BEC}(x) \circledast \mathrm{BEC}(x) \equiv \mathrm{BEC}(x^2)$.

Next, let us analyze the serial combination. By abbreviating $W \boxtimes W$ as W_s, we have

$$\begin{aligned}
\frac{1}{2}\bar{x}^2 &= W_s(0,0|0) = W_s(1,1|0) = W_s(0,1|1) = W_s(1,0|1),\\
\frac{1}{2}\bar{x}x &= W_s(\mathcal{E},0|0) = W_s(\mathcal{E},1|0) = W_s(\mathcal{E},1|1) = W_s(\mathcal{E},0|1)\\
&= W_s(0,\mathcal{E}|0) = W_s(1,\mathcal{E}|0) = W_s(0,\mathcal{E}|1) = W_s(1,\mathcal{E}|1),\\
x^2 &= W_s(\mathcal{E},\mathcal{E}|0) = W_s(\mathcal{E},\mathcal{E}|1).
\end{aligned}$$

all the other cases have a value of 0. Consider the following disjoint sets of possible outputs,

$$\begin{aligned}
\mathcal{S}_0 &= \{(0,0),(1,1)\},\\
\mathcal{S}_1 &= \{(0,1),(1,0)\},\\
\mathcal{S}_\mathcal{E} &= \{(\mathcal{E},0),(\mathcal{E},1),(0,\mathcal{E}),(1,\mathcal{E}),(\mathcal{E},\mathcal{E})\},
\end{aligned}$$

they are again viewed as super-symbols, representing the decoding scheme. Then we can calculate the probability of each set happening:

$$\begin{aligned}
\bar{x}^2 &= \Pr\{\mathcal{S}_0|0\} = \Pr\{\mathcal{S}_1|1\},\\
0 &= \Pr\{\mathcal{S}_1|0\} = \Pr\{\mathcal{S}_0|1\},\\
2x - x^2 &= \Pr\{\mathcal{S}_\mathcal{E}|0\} = \Pr\{\mathcal{S}_\mathcal{E}|1\}.
\end{aligned}$$

Hence, we have that $\mathrm{BEC}(x)^- = \mathrm{BEC}(x) \boxtimes \mathrm{BEC}(x) \equiv \mathrm{BEC}(2x - x^2)$. □

By the tedious work above, we have successfully proven the two roads to be equivalent. It should be noted that the grouping of the super-symbols are achieved by considering the polar decoder, which is a decoding scheme based on maximum likelihood test.

Remark 2.24

Channels can have arbitrary output alphabets, and need not be integers that can be added. For both serial and parallel combinations, we talked about *aggregating* the output symbols in certain ways (for example, adding the symbols as accounting for parity or creating super-symbols), but these interpretations are actually based on the fact that we treat *symbols posing the same coding challenge as the same.*

We cal a BMSC channel $W : \{0,1\} \to \mathcal{Y}_W$ a *symbol aggregation* of another BMSC $V : \{0,1\} \to \mathcal{Y}_V$ if there exists a map $\pi : \mathcal{Y}_V \to \mathcal{Y}_W$ satisfying

$$\begin{aligned}
V(y \mid 0) : V(y \mid 1) &= W(\pi(y) \mid 0) : W(\pi(y) \mid 1),\\
\sum_{v \in \pi^{-1}(y)} V(v \mid 0) + V(v \mid 1) &= W(y \mid 0) : W(y \mid 1).
\end{aligned} \tag{2.41}$$

2.7. Reed-Muller Code

11 Mar.

As a closing-off to this chapter, let us look at the Reed-Muller code and its similarities with the polar code. This section is highly related to Section 4.3.

Consider the encoding of a 2-layer polar code:

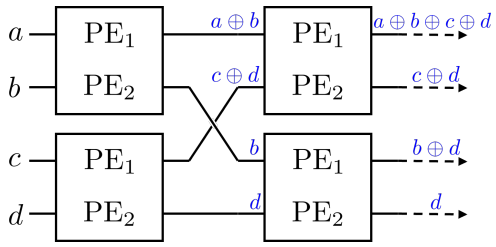


Figure 2.17. Codewords of the two-layer polar code.

We can rewrite the encoding as a matrix product

$$\begin{bmatrix} a & c & b & d \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a & c & b & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes 2}, \tag{2.42}$$

where $\otimes$ is the Kronecker product of matrices.

Definition 2.25: Kronecker Product

The Kronecker product of matrices $A = [a_{ij}]$, B is

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots \\ a_{21}B & a_{22}B & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{2.43}$$

This is also known as the "tensor product" of matrices. Note that we also have $A^{\otimes n} = \underbrace{A \otimes \cdots \otimes A}_{n \text{ times}}$.

In general, under a permutation of the channel orders, we can describe the encoding of an n-layer polar code via the matrix

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n}, \tag{2.44}$$

this matrix is known as the Arıkan matrix or the 1011-matrix, which is the *generator matrix* of polar code. By writing out the elements to the Arıkan matrix explicitly, it beautifully represents the famous fractal *Sierpinski's triangle*. Each row to the Arıkan's matrix represents a channel W^s, where $s \in \{0,1\}^{2^n}$.

Now, how will this help us with the analysis of the code? Recall our scheme for polar code encoding with genie, and compare with the following scheme for Reed-Muller code proposed.

- **Polar Code Encoding:** choose rows of the Arıkan's matrix with lower $P_e(\mathrm{BEC}(x)^s)$ by applying $f(x) = x^2$ and $g(x) = 2x - x^2$ recursively. A partial ordering is created by ordering the error probability.
- **Reed-Muller Code:** choose rows of the Arıkan's matrix with a lot of 1's. Explicitly, choose those with the higher Hamming weights.

Example: Consider the 3-layer polar code, we have the Arıkan's matrix with the weights listed as

Arıkan's matrix	Hamming weight
1	1
1 1	2
1 0 1	2
1 1 1 1	4
1 0 0 0 1	2
1 1 0 0 1 1	4
1 0 1 0 1 0 1	4
1 1 1 1 1 1 1 1	8

See how those with higher Hamming weights are the ones chosen by the genie! They correspond to those with a lower error probability. Moreover, unlike the analysis of partial ordering over the error probability polynomials. The Hamming weights to n-layer polar codes can be easily generalized by doubling the weights to the $(n-1)$-layer polar codes. This is a much simpler ordering scheme when compared to polar code. □

The exact amount of rows chosen by the genie to be the good channels under the Reed-Muller code depends on the code rate desired, but the ordering is simply based on the Hamming weights.

Definition 2.26: Reed-Muller Code

We define the Reed-Muller code $\mathrm{RM}(r,m) \subseteq \{0,1\}^{2^m}$ as the rows of channels of the Arıkan's matrix chosen such that their Hamming weights $\geq 2^{m-r}$.

The higher r is for a given Reed-Muller code $\mathrm{RM}(r,m)$, the higher rate the coding scheme is; on the contrary, for a lower r, one pursues a code with high standards.

Remark 2.27

Some examples to Reed-Muller code listed on Wikipedia are:

(a) $\mathrm{RM}(0,m)$: repetition code, 1-dimensional code generate by all-1 vector,

(b) $\mathrm{RM}(m,m)$: the entire space,

(c) $\mathrm{RM}(1,m)$: Hadamard code,

(d) $\mathrm{RM}(m-1,m)$: single-parity-check code,

(e) $\mathrm{RM}(m-2,m)$: extended Hamming code.

3. Polar Code in Source Coding

3.1. Source Coding

11 Mar.

3.1.1. Shannon's Source Coding Theorem

In Shannon's 1948 paper, he introduced source coding as a mean of compression for a source, i.e., a sequence of symbols X_i. The information theorist Blahut further defines *compression* as *lossy compression*, and *compaction* as *lossless compression*.

The following information quantity is useful in describing source coding:

Definition 3.1: Entropy

For an i.i.d. source X following a probability density function (can be easily extended to the continuous case) $p(x)$, we define its entropy to be

$$H(X) = -\sum_x p(x) \lg p(x) \text{ bits.} \tag{3.1}$$

If $\ln$ is used in place of $\lg$, the units changes from "bits" to "nats".

Theorem 3.2: Shannon's Source Coding Theorem

Given an i.i.d. source $X_1, X_2, \ldots$, a total of $H(X)+\varepsilon$ bits are needed per observation to describe the source.

Example: Consider the following *Huffman coding*, which is an easier version of Shannon's source coding theorem. Let a random source of nucleobases X be equal to A w.p. $\frac{1}{2}$, C w.p. $\frac{1}{4}$, G w.p. $\frac{1}{8}$, or T w.p. $\frac{1}{8}$. Since $-\lg p(A) = 1$, we should use 1 bit to describe A; since $-\lg p(C) = 2$, we should use 2 bits to describe C; and similarly, we should use 3 bits to describe both G and T. For example, we encode the nucleobases by $A : 0$, $C : 10$, $G : 110$, and $T : 111$. If one tries to encode n nucleobases, we have the total expected bits used as

$$\frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 3 + \frac{n}{8} \cdot 3 = \frac{7}{4} n \text{ bits} = n \cdot H(X).$$

□

Let us define a few more terms:

Definition 3.3: Information Density

We define the information density / surprisal of a realization x of the random variable X as

$$h(x) = -\lg p(x). \tag{3.2}$$

Entropy is equal to the expected surprisal, i.e., $H(X) = \mathbb{E}[h(X)]$.

Definition 3.4: Asymptotic Equipartition Property, AEP

This describes the log version of the law of large number. Notice that

$$h(x_1 \dots x_n) = -\lg\big(p(x_1)\cdots p(x_n)\big) = \sum_{i=1}^{n} -\lg p(x_i) = \sum_{i=1}^{n} h(x_i). \tag{3.3}$$

This is an i.i.d. sum of R.V.s! We can use the concentration inequality:

$$\Pr\left\{\frac{\sum_{i=1}^{n} h(X_i)}{n} - H(X) \geq \varepsilon\right\} \leq \exp(-c\varepsilon^2 n).$$

The AEP property simply states the concentration of the average of information density converges to the entropy.

Let us now prove the theorem. We will further use polar code to achieve it.

Proof. (Shannon's Source Coding Theorem, fixed length version). Let us try to encode as many length n strings as possible using only $n\big(H(X)+2\varepsilon\big)$ bits. Let us define a *typical set*

$$\mathcal{A}(\varepsilon) := \left\{ x_1 \dots x_n \in \mathcal{X}^n \,\middle|\, h(x_1 \dots x_n) \leq n\big(H(X)+\varepsilon\big) \right\}. \tag{3.4}$$

For a string $x_1 \dots x_n \in \mathcal{A}(\varepsilon)$, we have

$$p(x_1 \dots x_n) \geq 2^{-n\left(H(X)+\varepsilon\right)} \Rightarrow \big|\mathcal{A}(\varepsilon)\big| \leq 2^{n\left(H(X)+\varepsilon\right)}. \tag{3.5}$$

Thus, the set $\mathcal{A}(\varepsilon)$ can be mapped injectively onto bit string of length $n\big(H(X)+\varepsilon\big)$. What about the sequences outside of $\mathcal{A}(\varepsilon)$? Luckily, by the concentration inequality, their total probability of appearing is exponentially small and can be neglected. ■

Proof. (Shannon's Source Coding Theorem, variable length version). If fix length encoding fails to describe a source, we can use the suboptimal "literal" encoding for some strings. An encoding of a string either looks to have around $nH(X)$ bits for those in $\mathcal{A}(\varepsilon)$, or that they are described by Cn bits, where C is a constant. Then the expected value of the code length will be

$$\mathbb{E}[\text{length}] = \Big(1-\exp(-c\varepsilon^2 n)\Big)\cdot n\big(H(X)+\varepsilon\big) + \exp(-c\varepsilon^2 n)\cdot Cn = n\big(H(X)+2\varepsilon\big).$$

■

In fact, most of our modern day coding uses variable length coding, as one can easily check that photos saved in your phone album has varying file size.

3.1.2. Arithmetic Encoding

Example: Consider a source X being equal to A w.p. $\frac{2}{3}$, equal to C w.p. $\frac{1}{3}$. Then given a string, for example $ACAACAC$, we can encode it into a binary string by the following procedure:

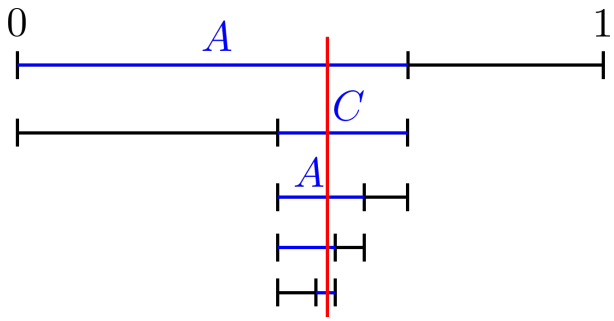


Figure 3.1. Illustration of arithmetic encoding.

By continuing partitioning the interval with respect to the probability one can find a shortest binary string in the last interval and use it to represent the original string of symbols. In our case above, the last interval is $[124/243, 44/81] \approx [0.510288, 0.543209]$, the shortest bit string within is 0.10001 in binary. The number of bits needed is approximately

$$\begin{aligned}\lg \frac{1}{\text{smallest interval size}} &= \lg \left(\frac{2}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}\cdot\frac{2}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}\cdot\frac{1}{3}\right)^{-1} \\ &= -\lg\left(\prod p(x_i)\right) \\ &\approx -\lg 2^{nH(X)}\Big|_{n=7} = nH(X) \approx 6.4.\end{aligned}$$

□

Note that arithmetic encoding is in fact capacity-achieving as seen in the example above. However, due to patent reasons, this code is not widely used.

3.1.3. Source Coding with Incorrect Code

Suppose that we have a coding scheme designed specifically for $\mathrm{Ber}(q)$. However, if the true underlying distribution is $\mathrm{Ber}(p)$ where $q \neq p$, what will happen?

Obviously, the compression to the code would not be as good as initially expected. In fact, we can characterize the extra length to the code paid by our ignorance to be

$$\begin{aligned}\mathbb{E}_{\mathrm{Ber}(p)}\left[-\lg q(X)\right] - \mathbb{E}_{\mathrm{Ber}(p)}\left[-\lg p(X)\right] &= \left[-p\lg q - (1-p)\lg(1-q)\right] \\ &\quad - \left[-p\lg p - (1-p)\lg(1-p)\right] \\ &= p\lg\frac{p}{q} + (1-p)\lg\frac{1-p}{1-q} \\ &=: D_{\mathrm{KL}}(P \,||\, Q).\end{aligned}$$

The term D_{KL} that we obtained here is another important information theoretical quantity:

Definition 3.5: Kullback-Leibler Divergence, KL Divergence

For two distributions P and Q, we can define their KL divergence to be

$$D_{\mathrm{KL}}(P \,||\, Q) := \mathbb{E}_P\left[\lg\frac{P}{Q}\right] = \sum p_i \lg\frac{p_i}{q_i}. \tag{3.6}$$

It is easy to see that the KL divergence must be greater than or equal to zero. Moreover, given $D_{\mathrm{KL}}(P \,||\, Q)$, it describes the expected surprise from modeling as Q given the true underlying distribution being P.

3.1.4. Source Polarization

For source coding, there are three different methods: Shannon's source coding theorem, arithmetic encoding, and Arıkan's source polarization. We have talked about the previous two already, let us discuss how the polarization trick we have seen earlier from noisy-channel coding can also be used in source coding. Note that the following trick we introduce has great advantage when we want to apply it to distribution shaping in the future.

In essence, source polarization is equivalent to channel polarization but without looking at the channel outputs. Given two symbols U_1, U_2 in a field F. Then, let us consider a similar scheme as in channel polarization:

(a) What is the distribution of $X_1 := U_1 + U_2$?

(b) Given $U_1 + U_2$, what is the distribution of $X_2 := U_2$?

(c) Do the same polarization recursion: given two copies of U from the same distribution, construct X_1 and X_2 as U^+ and U^-. Given two copies of U^+, construct U^{++} and U^{+-}; given two copies of U^-, construct U^{-+} and U^{--}. And so on.

One should be able to observe that $H(U^{\pm\pm\cdots})$ polarizes to 0 or $\lg|F|$, which corresponds to a constant random variable or a uniform distribution over F, respectively.

The number of $s \in \{+,-\}^n$ such that $H(U^s) \approx 0$ is about $2^n(1 - \frac{H(U)}{\lg|F|})$; the number of $s \in \{+,-\}^n$ such that $H(U^s) \approx \lg|F|$ is about $2^n(\frac{H(U)}{\lg|F|})$. By a similar argument using martingale, the quantity going up or down remains balanced so as to have the expectation unchanged. We do not provide a detailed proof here.

The source coding scheme under this setup is: remember those U^s that satisfy $H(U^s) \approx \lg|F|$, i.e., remember those that are random. This scheme is, in fact, capacity-achieving.

Amazingly, polar code can do both source coding and noisy-channel coding as defined by Shannon's 1948 paper.

3.1.5. Source Coding with Side Information

Given a source that generates a pair of random variables (X_i, Y_i) where X_i is what one tries to remember, and Y_i is the *side information* available at both the encoder and decoder.

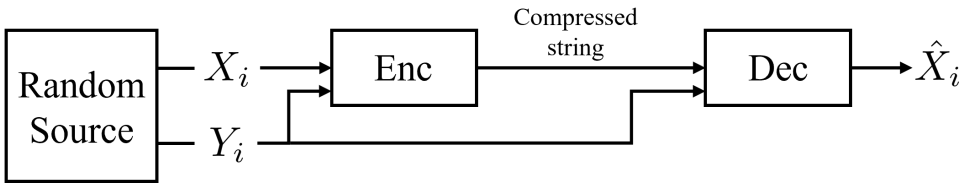


Figure 3.2. Illustration of source coding with side information.

The code rate is the *conditional entropy* $H(X|Y) = H(X,Y) = H(Y)$, which satisfies $H(X|Y) \leq H(X)$. This is the Slepian-Wolf coding, developed after WWII to figure out the amount of information needed to be sent from spy planes with overlapping scanning areas.

The proof to the Slepian-Wolf coding as a valid coding method is similar to that as Shannon's source coding theorem:

(a) Define conditional information density $h(x_i|y_i) = -\lg p(x_i|y_i)$.

(b) $h(x_1 \dots x_n | y_1 \dots y_n) =$ sum of i.i.d. R.V.s $= \sum_i h(x_i|y_i)$.

(c) $\mathcal{A}(\varepsilon) = \left\{ \text{string} \,\middle|\, h \le n(H(X|Y) + \varepsilon) \right\}$. (AEP and typical set)

(d) Find injection between $\mathcal{A}(\varepsilon)$ and strings of length $n(H(X|Y) + \varepsilon)$.

(e) Bound error probability by concentration inequality.

Similarly, one can realize the capacity-achieving property using polar code. The proof is also of similar fashion:

(a) Define polarized random variables $(X|Y)^{\pm}$.

(b) Check for polarization: $H((X|Y)^s) = 0$ or $\lg|F|$.

(c) Find number of s such that H is high is about $2^n(H(X|Y) + \varepsilon)$.

(d) Remember these random variables that are almost uniformly random.

(e) Check for capacity-achieving property.

3.2. Channel Parameters

18 Mar.

For a channel $W : X \mapsto Y$, we have the usual parameters such as entropy $H(W) := H(X|Y)$ and mutual information $I(W) := I(X;Y)$. Let us define a few more channel parameters that will aid our analysis of channels latter on.

We especially consider those that respect the structure of the decomposition of a compound channel into BSCs. Consider an arbitrary BMSC $W = \sum_i a_i \mathrm{BSC}(p_i)$, let us define

Definition 3.6: Channel Parameters

- **Conditional Entropy**:

$$H(W) := \sum_i a_i H\left(\mathrm{BSC}(p_i)\right), \tag{3.7}$$

$$H\left(\mathrm{BSC}(p)\right) := h_b(p) = -p \lg p - \bar{p} \lg \bar{p}. \tag{3.8}$$

  The function h_b is the binary entropy, and $\bar{p} = 1 - p$. Note that the capacity of a BSC is $C(\mathrm{BSC}(p)) := 1 - H(\mathrm{BSC}(p))$.

- **Bhattacharyya Parameter**:

$$Z(W) := \sum_i a_i Z\left(\mathrm{BSC}(p_i)\right), \tag{3.9}$$

$$Z\left(\mathrm{BSC}(p)\right) := 2\sqrt{p\bar{p}}. \tag{3.10}$$

  Note that when $p \to 0$ or 1, $Z \to 0$; when $p \to \frac{1}{2}$, $Z \to 1$.

- **Total Variation Distance**:

$$T(W) := \sum_i a_i T\left(\mathrm{BSC}(p_i)\right), \tag{3.11}$$

$$T\left(\mathrm{BSC}(p)\right) := |1 - 2p| = \left|\frac{1}{2} - p\right| + \left|\frac{1}{2} - \bar{p}\right| = |p - \bar{p}|. \tag{3.12}$$

This parameter is also a symmetric one. Furthermore, it measures how far away our distribution is from the uniform distribution.

- **Bit Error Probability**:

$$P_e(W) := \sum_i a_i P_e\left(\mathrm{BSC}(p_i)\right), \tag{3.13}$$

$$P_e\left(\mathrm{BSC}(p)\right) := 2\min\{p, \bar{p}\}. \tag{3.14}$$

Where does the mysterious 2 come from? Observe the following example for an explanation.

Example: Consider a channel $\mathrm{BEC}(x) = \bar{x}\ \mathrm{BSC}(0) + x\ \mathrm{BSC}(1/2)$. The error probability actually depends on how we want to decode it. If one simply gives up when seeing an erasure output, the error probability will be $P_e(\mathrm{BSC}(x)) = x$. However, if we try really hard and simply guess when receiving an erasure, the error probability will be, under a uniform prior, $P_e(\mathrm{BSC}(x)) = x/2$.

It is of a notational agreement that when considering the channel parameter P_e, we let it be characterizing the error probability when we simply give up. Henceforth, we have

$$P_e\left(\mathrm{BEC}(x)\right) = \bar{x}\ P_e\left(\mathrm{BSC}(0)\right) + x\ P_e\left(\mathrm{BSC}(1/2)\right) = \bar{x}\cdot 0 + x\cdot 2\cdot\frac{1}{2} = x.$$

The benefit of this notational agreement will be the fact that $P_e\left(\mathrm{BSC}(p)\right) \in [0,1]$ for all p, and so is their convex combination. This is the same for all other parameters we just introduced, they all lie in the interval $[0,1]$. □

These parameters will be shown to be really useful in characterizing how good a channel is. And by converting between different parameters, many of the properties of a channel can be characterized as well.

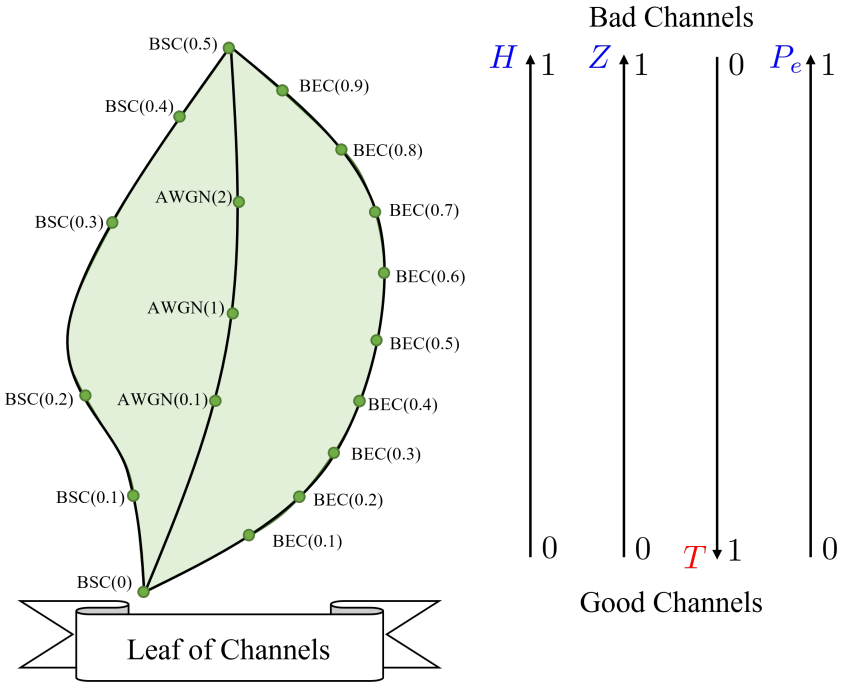


Figure 3.3. Leaf of channels representing the trend of channels from bad to good. The measures of the trend of how good a channel is are the channel parameters, coinciding on the value of 0 and 1 on $\mathrm{BSC}(0)$ and $\mathrm{BSC}(0.5)$ for the parameters H, Z, P_e (measuring how bad a channel is), and vice versa for T.

3.2.1. Inequalities of Channel Parameters

For a more exhaustive list of results and proofs of the topic in this subsection, please refer to the papers "A note on some inequalities used in channel polarization and polar coding" [JA18] and "Binary Polar Codes Based on Bit Error Probability" [Mur22].

Theorem 3.7

For all BMSC W, we have

$$Z(W)^2 \leq H(W) \leq Z(W). \tag{3.15}$$

Proof. By plotting the respective values for the simple case $W = \mathrm{BSC}(p)$, we can see the above inequality holds true. Furthermore, the two inequalities still holds true under convex combination, but the first needs to further utilize the fact that $y = x^2$ is a convex functions. ■

Theorem 3.8

For a BMSC W,

$$1 - T(W) \leq H(W) \leq h_b\left(\frac{1 - T(W)}{2}\right) \tag{3.16}$$

Proof. The proof is similar to the previous one, where one first show the inequality holds for BSCs, which is trivial. Then the first inequality is easily extended under convex combination. The second inequality still holds true since $h_b(p)$ is concave. ■

Remark 3.9

BEC is really special in that its $H = Z = P_e = 1 - T$ for all possible erasure probabilities.

This is easy to show since that BECs are convex combination of the extremal BSCs, i.e., $\mathrm{BSC}(0)$ and $\mathrm{BSC}(1/2)$. The channel parameters coincide on these extremals.

With the two inequalities above, we can actually bound regions of possible (H, Z) and (H, T) pairs as shown in Figure 3.4 below. Note that the upper curve in the H-Z plot is not $Z^2 = X$, it is not optimal.

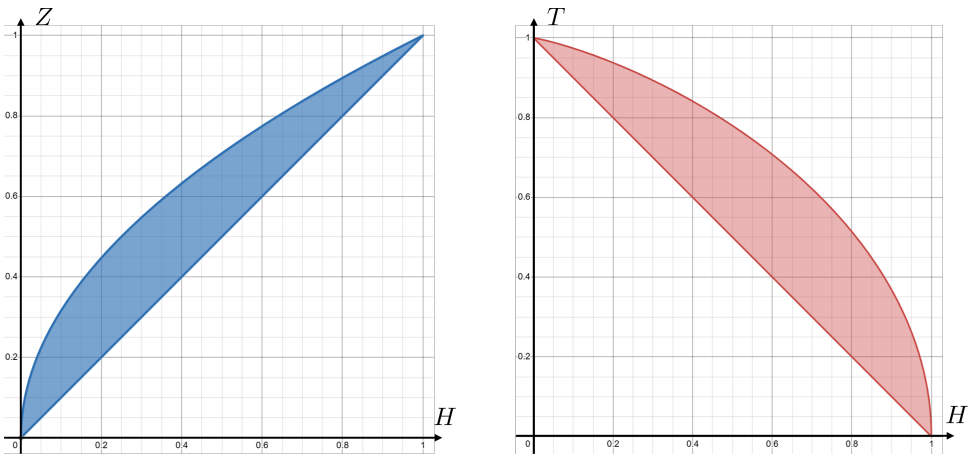


Figure 3.4. Possible combinations of channel parameters.

3.2.2. Channel Parameters as Martingales

Why do we study all these parameters? Because each one of them is good at one or two aspects, and only when combined together do we learn the full picture.

Theorem 3.10: Channel Parameters of Polarized Sub-Channels

The following relations hold for the polarized sub-channels W^+ and W^- of the BMSC W:

$$H(W^+) + H(W^-) = 2H(W), \tag{3.17}$$
$$Z(W^+) = Z(W)^2, \tag{3.18}$$
$$Z(W^-) \leq 2Z(W) - Z(W)^2, \tag{3.19}$$
$$T(W^-) = T(W)^2, \tag{3.20}$$
$$T(W^+) \leq 2T(W) - T(W)^2. \tag{3.21}$$

Hence, we know that for $s_{1:k} \in \{+,-\}^k$ $H(W^{s_{1:k}})$ is a martingale, and both $Z(W^{s_{1:k}})$ and $T(W^{s_{1:k}})$ are supermartingales. Furthermore, we have the more general results of

$$Z(W_1 \circledast W_2) = Z(W_1) \cdot Z(W_2), \tag{3.22}$$
$$T(W_1 \boxtimes W_2) = T(W_1) \cdot T(W_2) \tag{3.23}$$

for channels W_1 and W_2.

Remark 3.11: Helliger Affinity

The Helliger affinity of a channel W is defined as

$$Z_\alpha(W) = 2p^\alpha \bar{p}^{1-\alpha}. \tag{3.24}$$

It is a (Rényi-like) generalization to the Bhatacharyya parameter (at $\alpha = 1/2$), also satisfying the multiplicative property of

$$Z_\alpha(W_1 \circledast W_2) = Z_\alpha(W_1) \cdot Z_\alpha(W_2). \tag{3.25}$$

The proof to parts of Theorem Theorem 3.10 will be shown later. But for now, let us analyze the consequences the martingales generated by the polarization process brought.

Theorem 3.12: Martingale Convergence Theorem

If $\{M_t\}$ is a bounded[a] supermartingale / submartingale, then its limit exists: $M_t \xrightarrow{\text{a.s / } L^1} M_\infty$.

[a] For supermartingales M_t, the boundedness is in the sense of that $\sup_t \mathbb{E}[-\min(M_t, 0)] < \infty$.

Theorem 3.13: Optional Stopping Time

For supermartingale $\{M_t\}$, we have $\mathbb{E}[M_\sigma] \leq M_0$.

For submartingale $\{M_t\}$, we have $\mathbb{E}[M_\sigma] \geq M_0$.

Theorem 3.14: Doob's Maximum Inequality

If $M_t \geq 0$ is a supermartingale, then $\Pr\{\max_t M_t \geq B\} \leq M_0/B$.

If $M_t \leq 0$ is a submartingale, then $\Pr\{\max_t M_t \leq -B\} \leq -M_0/B$.

When applied to the channel parameters, one can have the following analyses:

Example: Take the Bhattacharyya parameter as an example, define the Bhattacharyya martingale as

$$Z_0 := Z(W),\ Z_k := Z(W^{s_{1:k}}) \quad (s^k \in \{+,-\}^k \text{ uniformly and random}). \tag{3.26}$$

By the martingale convergence theorem, a limit should exist. It can be shown that $Z_\infty \in \{0,1\}$. Then we have

$$\Pr\{Z_\infty = 1\} = H_0 = H(W) \leq Z_0.$$

Weirdly enough, one would expect the left hand side be equal to Z_0 by mirroring the proof from martingale. However, since now Z_k is a supermartingale, the analysis should be: since $Z(W) \approx 1$ if and only if $H(W) \approx 1$, we have

$$\Pr\{Z_\infty = 1\} = \Pr\{H_\infty = 1\} = H_0. \tag{3.27}$$

□

In fact, we have

$$H_\infty = Z_\infty = 1 - T_\infty = P_{e,\infty}. \tag{3.28}$$

Similarly, we have that as $k \to \infty$, T_k converges to $T_\infty \in \{0,1\}$, with $\Pr\{T_\infty = 1\} = 1 - H_0$.

3.2.3. Polar Decoder under Parallel and Serial Combination

The general decoding scheme for a polar decoder is abstractly illustrated as below:

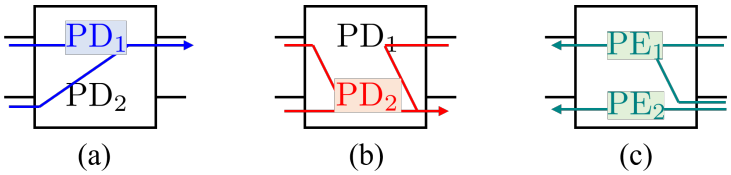


Figure 3.5. The three steps for decoding of polar code.

The decoding steps are implemented in real life by analyzing the likelihood ratio. To derive the likelihood ratio of the input symbols, let us introduce a new notion.

Definition 3.15: Posterior Probability Representation of Uncertain Symbols

If one were to guess the input $x \in \mathcal{X} = \{0,1\}$ given the received $y \in \mathcal{Y}$, the only crucial information are the posterior probabilities. Hence, one should represent y by the pair of posterior probabilities

$$\big(W(0|y), W(1|y)\big) \tag{3.29}$$

with a uniform prior, where the channel is $W : \mathcal{X} \to \mathcal{Y}$.

Example: Consider the channel below:

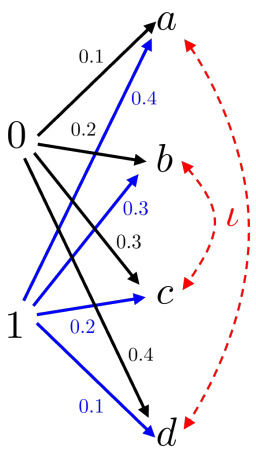


Figure 3.6. Example of a BMSC for posterior representation.

In stead of "a", use the representation

$$\big(W(0|a, W(1|a)\big) = \left(\frac{0.1}{0.1+0.4}, \frac{0.4}{0.1+0.4}\right) = \left(\frac{1}{5}, \frac{4}{5}\right).$$

Similarly, instead of "b", use $(2/5, 3/5)$; instead of "c", use $(3/5, 2/5)$; instead of "d", use $(4/5, 1/5)$. □

Example: Consider the channel below

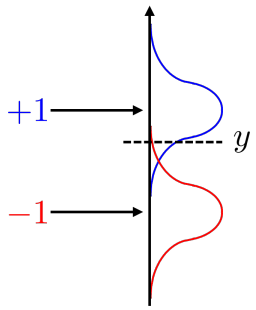


Figure 3.7. Example of posterior representation on continuous random variables.

Since probabilities of outputs no longer make sense, we utilize the probability densities instead. Let the $\varphi(y)$ be the probability density function of the standard normal distribution. Then since $W(y|1) = \varphi(y-1)$ and $W(y|-1) = \varphi(y+1)$, we have the posterior representation of the output y being

$$\big(W(1|y), W(-1|y)\big) = \left(\frac{\varphi(y-1)}{\varphi(y-1)+\varphi(y+1)}, \frac{\varphi(y+1)}{\varphi(y-1)+\varphi(y+1)}\right).$$

□

Now let utilize both the posterior representation and the channel parameters introduced in the analysis of parallel and serial combination of channels.

Posterior Representation & Parallel Combination: If one sends $X = x$ through W and W', and obtained posterior representations $(\alpha, \bar{\alpha})$ and $(\beta, \bar{\beta})$, respectively. Then what is the posterior

probability of x now? This can be readily solved by pretending W and W' as BSCs, and consider the figure below:

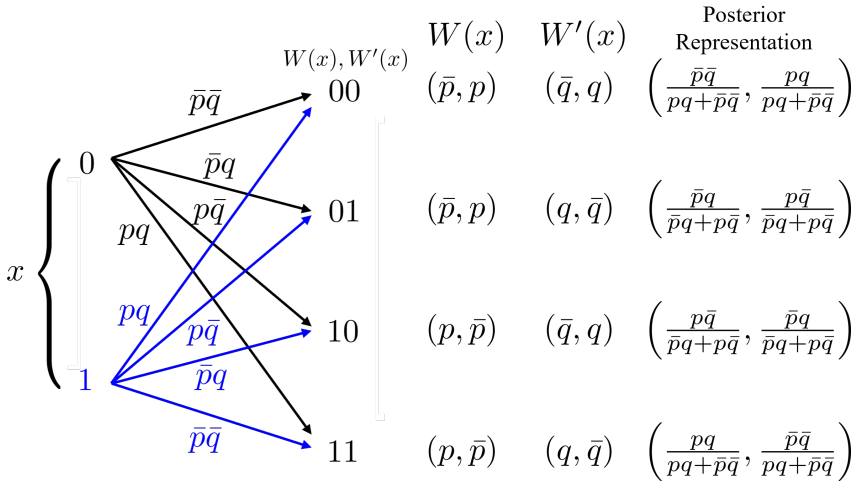


Figure 3.8. Posterior representation for parallel combination of BSC's.

It is easily seen that if the received symbols are $(\alpha, \bar{\alpha})$ and $(\beta, \bar{\beta})$, they amount to a posterior representation of

$$(\gamma, \bar{\gamma}) := \big(W(x=0|W(x), W'(x)), W(x=1|W(x), W'(x))\big) = \left(\frac{\alpha\beta}{\alpha\beta + \bar{\alpha}\bar{\beta}}, \frac{\bar{\alpha}\bar{\beta}}{\alpha\beta + \bar{\alpha}\bar{\beta}}\right). \tag{3.30}$$

Consider the likelihood ratios, especially, we can easily see that they are multiplicative!

$$\frac{\gamma}{\bar{\gamma}} = \frac{\alpha}{\bar{\alpha}} \cdot \frac{\beta}{\bar{\beta}}. \tag{3.31}$$

This serves as a mnemonic for one to remember the channel combination rule.

We hence have the following result:

Theorem 3.16: Bhattacharyya Parameter under Parallel Combination

For channels W_1 and W_2, we have

$$Z(W_1 \circledast W_2) = Z(W_1) \cdot Z(W_2). \tag{3.32}$$

Especially when $W_1 = W_2 = W$, we have

$$Z(W^+) = Z(W)^2. \tag{3.33}$$

Proof. Since any BMSC can be decomposed into a convex combination of BSCs, we only need to proof the theorem above for parallel combination of BSCs. This can be directly proven by plugging in Equation (2.33).

However, also observe that for a general channel $W = \sum_i a_i \mathrm{BSC}(p_i)$. By treating the output symbols y_i in their posterior representations $(\alpha(y_i), \bar{\alpha}(y_i)) = (p_i, \bar{p}_i)$ as a random variable with distribution derived from sending only 1 as the input, we can construct the following expectation:

$$Z(W) = \sum_i a_i \cdot 2\sqrt{p_i \bar{p}_i} = \sum_i a_i \left(p_i \sqrt{\bar{p}_i / p_i} + \bar{p}_i \sqrt{p_i / \bar{p}_i}\right) = \mathbb{E}\left[\sqrt{\bar{\alpha}/\alpha}\right].$$

Then the result above can be simply derived from

$$Z(W_1 \circledast W_2) = \mathbb{E}\left[\sqrt{\gamma/\bar{\gamma}}\right] = \mathbb{E}\left[\sqrt{\alpha/\bar{\alpha}}\right] \cdot \mathbb{E}\left[\sqrt{\beta/\bar{\beta}}\right] = Z(W_1) \cdot Z(W_2).$$

■

Posterior Representation & Serial Combination: If one sent x through W obtaining $(\alpha, \bar{\alpha})$ and sent y through W' obtaining $(\beta, \bar{\beta})$. Then what is the posterior representation to $x + y$? Similarly, pretend W and W' to be BSCs and consider the figure below:

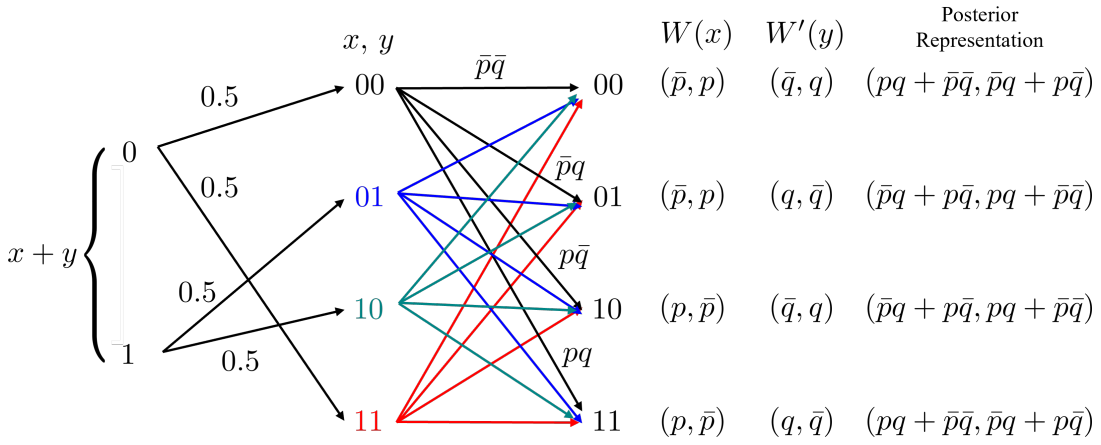


Figure 3.9. Posterior representation for serial combination of BSC's.

One can see that if the received symbols are $(\alpha, \bar{\alpha})$ and $(\beta, \bar{\beta})$, they amount to a posterior representation of

$$(\gamma, \bar{\gamma}) := \left(W(x+y=0|W(x), W'(y)), W(x+y=1|W(x), W'(y))\right) = \left(\alpha\beta + \bar{\alpha}\bar{\beta}, \bar{\alpha}\beta + \alpha\bar{\beta}\right). \tag{3.34}$$

One can also observe that the signed variation distance is multiplied:

$$(\gamma - \bar{\gamma}) = (\alpha - \bar{\alpha}) \cdot (\beta - \bar{\beta}). \tag{3.35}$$

This serves as yet another mnemonic.

The following result follows suit:

Theorem 3.17

Total Variation Distance under Serial Combination For channels W_1 and W_2, we have

$$T(W_1 \boxtimes W_2) = T(W_1) \cdot T(W_2). \tag{3.36}$$

Especially when $W_1 = W_2 = W$, we have

$$T(W^-) = T(W)^2. \tag{3.37}$$

Proof. Again, we only need to proof the theorem above for serial combination of BSCs, and can be directly proven by plugging in Equation (2.34). However, the proof can also be done by constructing

a random variable. Treating $(\alpha, \bar{\alpha})$ as a random variable following the distribution as sending only 1 through as input, then

$$\begin{aligned}\rightarrow T(W) &= \sum_i a_i |p_i - \bar{p}_i| = \sum_i a_i \left(\bar{\alpha}(y_i)\left|\alpha(y_i) - \bar{\alpha}(y_i)\right| + \alpha(y_i)\left|\bar{\alpha}(y_i) - \alpha(y_i)\right|\right) \\ &= \mathbb{E}\left[\left|\alpha - \bar{\alpha}\right|\right].\end{aligned}$$

Hence,

$$T(W_1 \boxtimes W_2) = \mathbb{E}\left[\left|\gamma - \bar{\gamma}\right|\right] = \mathbb{E}\left[\left|\alpha - \bar{\alpha}\right|\right] \cdot \mathbb{E}\left[\left|\beta - \bar{\beta}\right|\right] = T(W_1) \cdot T(W_2).$$

■

In real applications, which representation is used? $(\alpha, \bar{\alpha})$, $\alpha/\bar{\alpha}$, or $\alpha - \bar{\alpha}$? The answer is that the log likelihood ratio (LLR) is used.

It is easy to see that under parallel combination, we have

$$\mathrm{LLR}_\gamma = \mathrm{LLR}_\alpha + \mathrm{LLR}_\beta, \tag{3.38}$$

where $\mathrm{LLR}_\alpha = \ln(\alpha/\bar{\alpha})$. The additive property makes the calculation of channel parallel combination very easy. How about serial combination?

Under serial combination, we have that

$$\mathrm{LLR}_\gamma = 2\,\mathrm{arctanh}\left(\tanh\left(\mathrm{LLR}_\alpha/2\right) + \tanh\left(\mathrm{LLR}_\beta/2\right)\right). \tag{3.39}$$

This can be easily seen by following the calculations below:

$$\begin{aligned}&\because \alpha/\bar{\alpha} = \exp(\mathrm{LLR}_\alpha) \Rightarrow \bar{\alpha} = \frac{1}{1+\exp(\mathrm{LLR}_\alpha)} \\ &\therefore \gamma - \bar{\gamma} = \frac{\exp(\mathrm{LLR}_\gamma) - 1}{\exp(\mathrm{LLR}_\gamma) + 1} = \frac{\exp(\mathrm{LLR}_\alpha) - 1}{\exp(\mathrm{LLR}_\alpha) + 1} \cdot \frac{\exp(\mathrm{LLR}_\beta) - 1}{\exp(\mathrm{LLR}_\beta) + 1} = (\alpha - \bar{\alpha}) \cdot (\beta - \bar{\beta}) \\ &\Rightarrow \tanh(\mathrm{LLR}_\gamma/2) = \tanh(\mathrm{LLR}_\alpha/2) \cdot \tanh(\mathrm{LLR}_\beta/2).\end{aligned}$$

Thus the relation is shown.

The two rules above constitutes the famous "product-sum rule." Moreover, since hyperbolic-(arc)tangent is costly to compute, as simpler approximation formula exists:

$$\mathrm{LLR}_\gamma \approx \mathrm{sgn}(\mathrm{LLR}_\alpha \cdot \mathrm{LLR}_\beta) \cdot \min\left\{\left|\mathrm{LLR}_\alpha\right|, \left|\mathrm{LLR}_\beta\right|\right\}. \tag{3.40}$$

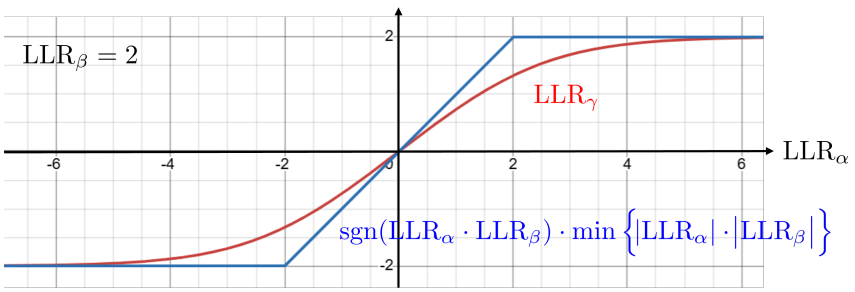


Figure 3.10. Approximation for the LLR of serial combination.

Example: If one flips a biased coin with probability of landing on heads being p a total of n times, what is the probability that the number of heads obtained is even?

This problem can be solved by using techniques from difference equations. However, we can recast this problem into a channel: Suppose one transmits the bit string $00\ldots0$ (length n) through a $W = \mathrm{BSC}(p)$, then the probability of receiving obtaining an even amount of 1's is the same as the probability in question. This then also equals to the probability that summing (mod 2 summation) the received bits gives you 0. The summation of received bits is equivalent to the serial combination of channels! Since the signed total variation distance (temporarily denoted by $\tilde{T}$) is multiplicative under serial combination, let the probability of obtaining a sum of 0 be denoted by q, then

$$\tilde{T}(W^{\boxplus n}) = q - (1-q) = \big((1-p)-p\big)^n = \tilde{T}(W)^n \Rightarrow q = \frac{1}{2}\left((1-2p)^n + 1\right).$$

□

3.3. Distributed Source Coding

This section discusses the possibility of distributed source coding by the method of Slepian-Wolf.

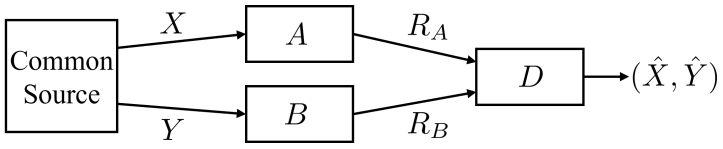


Figure 3.11. Distributed source coding of a common source by two agents A and B.

Consider the figure above: agent Alice (A) and Bob (B) has a common source and plan to send bit streams with rate R_A and R_B to decoder David (D) to learn about the source. What are the possible (R_A, R_B) pairs?

From Section 3.1.5, we have learned that if Alice or Bob shares information with the other (say, Alice knows about the random variable Y that Bob obtains), then Alice can send at a rate of $H(X|Y)$ while Bob sends at a rate of $H(Y)$. But now our question is restricted so that there are no information exchange between Alice and Bob, is there a way for the them to compress their information while still enabling David to decode the information losslessly? Our task then will be to find a way to describe the *rate region*.

Definition 3.18: Rate Region

The rate region is a subset $\mathcal{R} \subset \mathbb{R}^2$ such that $(R_A, R_B) \in \mathcal{R}$ is feasible.

The characterization to the rate region is the figure below:

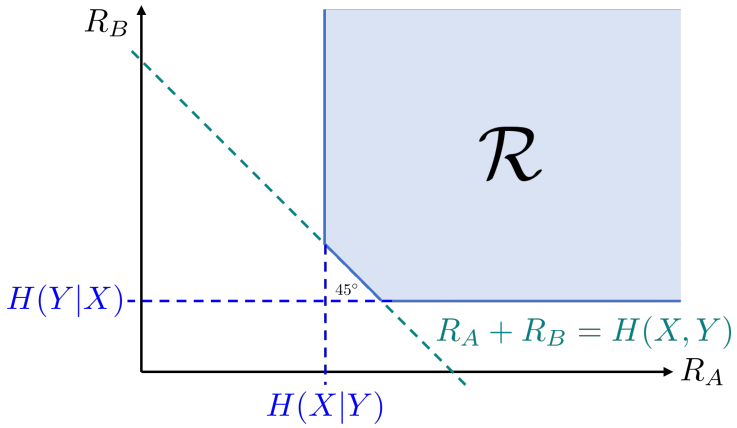


Figure 3.12. Rate region of two agent distributed source coding.

Remark 3.19

The more bits used to describe a single symbol, the more possible it is do decode. Hence, the region $\mathcal{R}$ is unbounded on the top right corner.

If X and Y are independent, $H(Y|X) = H(Y)$ and $H(X|Y) = H(X)$, hence $H(X) + H(Y) = H(X,Y)$, the rate region simply becomes an unbounded rectangle.

The strategy of the coding scheme for Alice and Bob is simple: ask Alice to send bit streams with a rate of $H(X|Y)$ and ask Bob to send with a rate of $H(Y)$. An immediate problem arises, however, without explicitly knowing the Y Bob obtains, how can Alice compress the information? Luckily, this task is actually possible, and the technique used is called *random binning*:

(a) Prepare $2^{n(H(X)+\varepsilon)}$ small balls for Alice. This corresponds to the typical set $\mathcal{A}(n,\varepsilon)$ of information in X.

(b) Prepare $2^{n(H(X|Y)+\varepsilon)}$ large bins.

(c) Alice throws the balls into the bins randomly. (Note that if there is no side information, Shannon says that Alice should in fact send David the index of the balls which costs $n(H(X)+\varepsilon)$ bits.)

(d) Alice send the index of the bin a ball is in to David, costing $n(H(X|Y)+\varepsilon)$ bits.

(e) In order to show that this trick actually works, David should be able to find the correct ball that is in the bin which has an index Alice sent and David knows.

(f) Each ball is a different random stream of variable $X_{1:n}$. David will select the ball with the largest probability $\Pr\{X_{1:n}|Y_{1:n}\}$.

(g) Control and bound the error probabilities:

- The wrong ball has a higher probability.
- $\cdots$ (to be left as an exercise for the readers)

By bounding the error probabilities, we can show that the strategy of random binning is feasible, and the error probabilities reaches zero as n becomes asymptotically large.

3.4. Speed of Polarization

25 Mar.

Referring back to Theorem 1.6, where we established the speed at which BECs polarize. Now let us delve deeper into the analysis and show some of the missing details while connecting with what we have learned. The papers "Unified Scaling of Polar Codes: Error Exponent, Scaling Exponent, Moderate Deviations, and Error Floors" [MHU16] and "Sub-4.7 Scaling Exponent of Polar Codes" [Wan+23] provide more recent advances in finding the scaling exponent and its relationship with other coding theoretical parameters.

Let us assume that there exists a special concave function $f(x)$ (an approximation would be $\approx [x(1-x)]^{2/3}$, with another commonly used exponent is 0.7). By concavity, we have that

$$\begin{aligned} &f(x^2) + f(2x - x^2) \le 2f(x) \\ &\Rightarrow f\left(P_e(\mathrm{BEC}(x)^+)\right) + f\left(P_e(\mathrm{BEC}(x)^-)\right) \le 2f\left(P_e(\mathrm{BEC}(x))\right). \end{aligned}$$

This is how this special function is linked with BECs. Furthermore, we are interested in the following quotient (see also the figure below) that is between 0 and 1:

$$\lambda := \sup_{0\le x\le 1} \frac{f(x^2) + f(2x - x^2)}{2f(x)}. \tag{3.41}$$

We have already shown that

$$\frac{1}{2^n} \sum_{s\in\{+,-\}^n} f\left(P_e(\mathrm{BEC}(x)^s)\right) \le \lambda^n f(x). \tag{3.42}$$

Hence as $n \to \infty$, we have that $f\left(P_e(\mathrm{BEC}(x)^s)\right)$ approaches 0 for all s, meaning that $P_e(\mathrm{BEC}(x)^s) =$ 0 or 1 for all s. This is our characterization of the speed of polarization.

For all possible convex f, which produces the smallest λ? It is obvious that for $[x(1-x)]^{2/3}$, the λ is not optimal. How do we find the optimal f? Simply by the power method.

Remark 3.20

Let us take a small detour into the "power method" from linear algebra, which is very similar to the question proposed above.

Given a matrix A, we would like to find the eigenvalue with the largest absolute value, i.e.

$$|\lambda_1| = \max_{\boldsymbol{x}} \frac{\|Ax\|}{\|x\|},$$

where the norm is the Euclidean norm. The maximum is achieved when $\boldsymbol{x} = \boldsymbol{x}_1$, the eigenvector corresponding to the eigenvalue λ_1. A useful method in finding $\boldsymbol{x}$, and hence λ_1, is the power method, in which one simply iterates the map

$$\boldsymbol{x} \leftarrow \varphi(\boldsymbol{x}) = \frac{A\boldsymbol{x}}{\|\boldsymbol{x}\|}.$$

Then we have $\varphi^n(\boldsymbol{x}) \to \boldsymbol{x}_1$ as $n \to \infty$.

For BECs, let us define the following operator

$$T_{\mathrm{BEC}}(f) = \frac{f(x^2) + f(2x - x^2)}{2}. \tag{3.43}$$

Then Equation (3.41) will be equivalent to finding the largest eigenvalue to the eigenvalue problem

$$T_{\mathrm{BEC}}(f) = \lambda f(x). \tag{3.44}$$

However, we always have the trivial largest eigenvalue of $\lambda = 1$ for $f(x) = 1$ and $f(x) = x$. Hence, we are to find the largest eigenvalue λ^* other than 1.

By the power method, we define the following iteration scheme:

$$f \leftarrow \frac{f(x^2) + f(2x - x^2)}{2 \sup_{x\in[0,1]} f(x)}. \tag{3.45}$$

In the end when this iteration converges, the supremum is $\lambda^* = 2^{-1/\mu^*}$, the value μ^* is coined the scaling exponent.

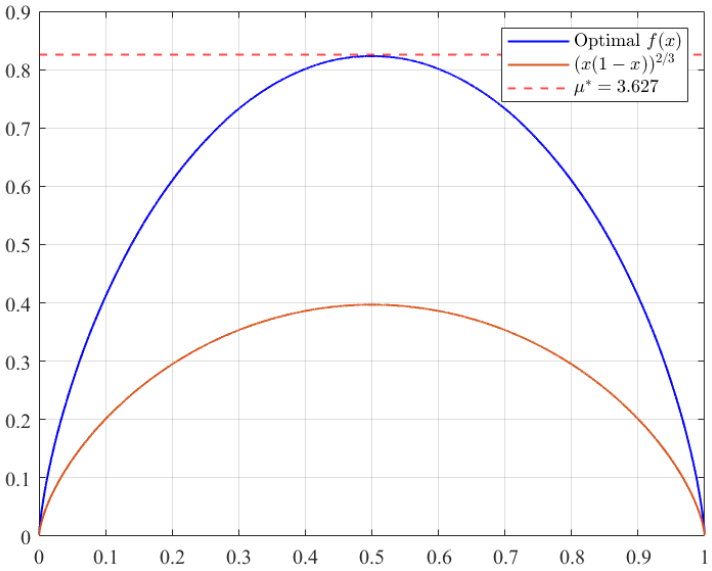


Figure 3.13. Eigenfunction to T_{BEC}.

Using $(x(1-x))^{2/3}$ as the initial condition, the algorithm quickly converges to Figure 3.13, with the optimal scaling exponent equal to $\mu^* = 3.627$.

Remark 3.21

Why does Equation (3.45) converge?

Previously in Equation (2.11), we considered M_n as the polarization process of $\mathrm{BEC}(x)$, and by arguments of martingale convergence theorem, we have seen that

$$\Pr\{M_n \in (\varepsilon, 1-\varepsilon)\} < \delta$$

is a constant bound on the converging probability. We can do better than this by introducing the optimal f (optimal as in the arguments above) to have

$$\begin{aligned}\Pr\{M_n \in (\varepsilon, 1-\varepsilon)\} &= \Pr\{f(M_n) > f(\varepsilon)\} \\ &\leq \frac{\mathbb{E}[f(M_n)]}{f(\varepsilon)} \\ &\approx 2^{-\frac{n}{3.627}} f(M_0)/f(\varepsilon).\end{aligned}$$

We can see that the speed of polarization is exponentially fast.

Remark 3.22

Besides plugging in x as the error probability, we have also learned that the Bhattacharyya parameter Z as an equivalent description of channel quality! For BECs, the Bhattacharrya parameter Z_n follows the same polarization process as P_e, and hence has the same polarization speed.

However, for more general BMSCs, the Bhattacharyya parameter becomes a supermartingale. By denoting $z = Z(W)$,

$$z\sqrt{2-z^2} \leq Z(W^-) \leq 2z - z^2, \tag{3.46}$$

the iteration Equation (3.45) hence needs to be updated as

$$f \leftarrow \sup_{x\sqrt{2-x^2}\leq y\leq 2x-x^2} \frac{f(x^2)+f(y)}{2\sup_{x\in[0,1]} f(x)}. \tag{3.47}$$

In this case, the scaling exponent obtained will be $\mu^* = 4.717$, just as described in Theorem 1.6.

3.5. Large Kernel of Polar Code

Previously in polar code, we have only been consider the 2×2 kernel $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ as proposed by Arıkan, where each encoding and decoding block has two input and two outputs. However, this can be modified to allow for a larger kernel.

If we consider

$$A = \begin{bmatrix} a_{11} & \cdots & a_{\ell 1} \\ \vdots & \ddots & \vdots \\ a_{1\ell} & \cdots & a_{\ell\ell} \end{bmatrix} \in \mathbb{F}_2^{\ell\times\ell}$$

as our kernel for polar code, then given the input bits $[u_1, \ldots, u_\ell]$, the encoded bits will be $[x_1, \cdots, x_\ell] = [u_1, \ldots, u_\ell]A$. With the encoded bits as channel inputs, the channel outputs will be $[y_1, \cdots, y_\ell]$. The decoder is again operated by maximum likelihood estimation. The ℓ polarized channels will be:

- $W^{(1)}$ is guessing u_1 given $y_1, \ldots, y_\ell$.
- $W^{(2)}$ is guessing u_2 given $y_1, \ldots, y_\ell$ and u_1.
- $W^{(3)}$ is guessing u_3 given $y_1, \ldots, y_\ell$ and u_1, u_2.
- $\vdots$
- $W^{(\ell)}$ is guessing u_ℓ given $y_1, \ldots, y_\ell$ and $u_1, \ldots, u_{\ell-1}$.

The matrix A is termed the *generator matrix* of the code. For more details, and also for the definitions of the terms used in the theorem below, please refer to Section 4.1.

Theorem 3.23

Consider a code with generator matrix A for a BMSC W. Let A_i be the ith row to A, and $\mathcal{C}_i$ as the code generated by $A_{i+1}, A_{i+1}, \dots, A_\ell$. We have the following inequality for the Bhattacharyya parameter:

$$Z(W^{(i)}) \leq \sum_{w \in \mathcal{C}_i} Z(W)^{|w+A_i|}. \tag{3.48}$$

The function $|w| = d_{\mathrm{H}}(w, 0)$ is the Hamming weight of the codeword w. The theorem is stated without a proof. But as a sanity check, let us consider the example below.

Example: Consider $W = \mathrm{BEC}(x)$, with $Z(W) = x$. Then let the generator matrix of the code be the Arıkan matrix

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \in \mathbb{F}_2^{2\times 2}, \qquad A_1 = [1 \quad 0], \qquad A_2 = [1 \quad 1],$$

with the code generated being

$$\mathcal{C}_1 = \{00, 11\}, \qquad \mathcal{C}_2 = \{00\}.$$

Then we have

$$\begin{aligned} Z(W^{(1)}) = Z(W^-) = 2x - x^2 \leq \sum_{w \in \mathcal{C}_1} x^{|w+A_1|} = 2x, \\ Z(W^{(2)}) = Z(W^+) = x^2 = \sum_{w \in \mathcal{C}_2} x^{|w+A_2|} = x^2. \end{aligned}$$

The theorem indeed holds true. □

Moreover, consider the uniformly distributed indices $s_1, s_2, \dots \in \{1, 2, \dots, \ell\}$. We can define a random process $W_{i+1} = (W_i)^{(s_{i+1})}$ with $W_0 = W$. Then the Bhattacharyya parameter $Z_i = Z(W_i)$ is also a random process. However,

$$\mathbb{E}[Z_{i+1}|Z_i] = \frac{1}{2}(2x) + \frac{1}{2}(x^2) \not\leq x,$$

it is no longer a supermartingale, nor a martingale. This is different from Theorem 3.10. Nevertheless, the following result holds true:

Theorem 3.24

Define $a \wedge b = \min(a, b)$, then

$$X_n = (Z_n \wedge \delta)^\varepsilon \tag{3.49}$$

is a supermartingale for δ and ε small enough.

The theorem is stated without a proof.

3.6. Polar Code with Joint Source-Channel Coding

We have seen that polar code doing source coding. We have also seen that polar code doing channel coding. Now let us see polar code doing both at the same time. The description of this section follows

the paper " Polar Coding without Alphabet Extension for Asymmetric Models" [HY13] by J. Honda and H. Yamamoto.

As a first example, let us consider $W = (X|Y)$ as the Z channel with the optimal input distribution that achieves the channel capacity. Define the channel $V = (X|\sim)$ that sends nothing. We can define both $W^{\pm\pm\cdots}$ and $V^{\pm\pm\cdots}$, and they will both be polarized: $H(W^s) \approx 0$ or 1 and $H(V^s) \approx 0$ or 1 for all $s \in \{+,-\}^n$. There are a total of four possible combinations (see table below), let us analyze their properties.

$H(V^s)$ \ $H(W^s)$	0	1
0	(A)	(B)
1	(C)	(D)

Table 3.1. Classification of bits: A is for source coding. B is impossible. C is for channel coding. And D is the frozen bits.

- (B): One can show that $\Pr\{\mathrm{B}\} = 0$, hence this combination is not possible.
- (D): $\Pr\{\mathrm{D}\} = H(W_0) = H(W)$.
- (C): Since $\Pr\{\mathrm{C}\cup\mathrm{D}\} = H(V_0) = H(X)$, hence $\Pr\{\mathrm{C}\} = H(X) - H(W) = H(X) - H(X|Y) = I(W)$. If we let C be the information bits, then we see that it is in fact capacity-achieving. This is the channel coding part.
- (A): For $H(V^s) \approx 0$, it implies that u_s is almost predictable as it only depend on the previous bits $u_{1:s-1}$. This is the source coding part, to shape the input distribution to be no longer uniformly distributed.

4. Linear Codes

In this section we trace back to the more classical coding theory of linear codes. Special emphasis are put on the Reed–Solomon code. We will first give some more basic results on the prime fields of $\mathrm{GF}(p)$, these includes the not-so-trivial weight enumerator polynomial and the MacWilliams' identity. Then, after diving into the vast theory of abstract algebra and having the knowledge of a general field $\mathrm{GF}(p^n)$, we will give a general description to Reed–Solomon code and its properties.

4.1. Binary Linear Code

25 Mar.

For half of the lecture, we have talked about polar codes and its non-trivial properties. Next, we will dive into the theory of LDPC codes. However, before that, we should first review the results of the classical linear codes.

Definition 4.1: Binary Linear Code

A (binary) linear code $\mathcal{C} \in \mathbb{F}_2^n$ is a subspace of the finite field $\mathbb{F}_2^n$ (also written as the Galois field $\mathrm{GF}(2)^n$), where

(a) n is the block length, and

(b) $k = \dim \mathcal{C}$ is the dimension of the code.

We also say that $\mathcal{C}$ is an $[n, k]$-code.

Besides the notation $[n, k]$-code, one might also see an $[n, k, d]$-code, where the variable d is defined as the minimum Hamming distance between two codewords

$$d := \min_{x,y \in \mathcal{C}} d_{\mathrm{H}}(x, y). \tag{4.1}$$

But note that since we are working with linear code, $\mathcal{C}$ is a subspace and $x - y \in \mathcal{C}$, so we also have that, in this case,

$$d = \min_{x \in \mathcal{C}} d_{\mathrm{H}}(x, 0) = \min_{x \in \mathcal{C}} |x| \,, \tag{4.2}$$

where $|x|$ again denotes the Hamming weight of x. We call the Hamming distance of a codeword from 0 the *Hamming weight* of that codeword.

And this is the whole point of using linear code! By the properties inherited from its vector space definition, many of the equations can be simplified. But sometimes it simplifies too much, making the result too trivial to be useful.

Remark 4.2

In comparison to linear code, we will see how polar code can be used to construct "non-linear" codes that can be better than linear code under certain circumstances later.

Definition 4.3: Corrupted Codeword

For a codeword $x \in \mathcal{C}$, the output of a channel W given input x is called a *corrupted (code)word.*

Definition 4.4: Weight Enumerator Polynomial

The weight enumerator of a linear code $\mathcal{C}$ is defined as the generating function of the form

$$A_{\mathcal{C}}(z) := \sum_{w \in \mathcal{C}} z^{|w|} = \sum_{i=0}^{n} A_i z^i, \tag{4.3}$$

where z is an arbitrary abstract variable, $|\cdot|$ denotes the weight of a codeword, and A_i is the number of codewords in $\mathcal{C}$ that has weight i. The weight enumerator is a polynomial in z. The sequence $(A_0, \dots, A_n)$ is called the weight distribution of the code.

Definition 4.5: Homogeneous Weight Enumerator

We can homogenize the weight enumerator of a linear code into

$$A_{\mathcal{C}}(x, y) := \sum_{w \in \mathcal{C}} x^{|w|} y^{n-|w|} = \sum_{i=0}^{n} A_i x^i y^{n-i}. \tag{4.4}$$

This is a homogeneous polynomial in the abstract variables x and y.

It is obvious to see that for d being the minimum weight of a linear code $\mathcal{C}$, we have, as $z \to 0$,

$$A_{\mathcal{C}}(z) \in O(z^d).$$

Further, the two forms of the weight enumerator can be interchanged,

$$A_{\mathcal{C}}(x, y) = y^n A_{\mathcal{C}}(x/y), \qquad A_{\mathcal{C}}(z) = A_{\mathcal{C}}(z, 1). \tag{4.5}$$

I.e., the two forms contain the same amount of information.

Remark 4.6

When $\mathcal{C}$ is used over $\mathrm{BSC}(p)$, there is a probability of $p^{|w|}\bar{p}^{n-|w|}$ that $0 \in \mathcal{C}$ is flipped into w.

For decoding, flipping $d/2$ bits is confusing enough, and will result in a block error probability $\leq O(p^{d/2})$ as $p \to 0$.

Since the weight enumerator polynomial carries information about the error, it is important.

Since $\mathcal{C}$ is a vector space, we can define a kind of scalar product on it:

Definition 4.7: Pairing

To put it in brief, a pairing is a bilinear map that satifies:

(a) $\langle x, y + z \rangle = \langle x, y \rangle + \langle x, z \rangle$,

(b) $a\langle x, y \rangle = \langle ax, y \rangle = \langle x, ay \rangle$,

where a, x, y reside in a suitable ring / module. As for our case where $a \in \mathbb{F}_2$ and $x, y \in \mathbb{F}_2^{n\times 1}$,

$$\langle x, y\rangle = x_1y_1 + \cdots + x_ny_n. \tag{4.6}$$

Note that a pairing is NOT an inner product.

Given a pairing, we can define the *dual* of a code.

Definition 4.8: Dual Code

The dual code to $\mathcal{C}$ is defined as

$$\mathcal{C}^\perp := \left\{ y \in \mathbb{F}_2^{n\times 1} \,\middle|\, y \perp \mathcal{C} \right\}. \tag{4.7}$$

The phrase "$y \perp \mathcal{C}$" is equivalent to $y \perp x$ for all $x \in \mathcal{C}$, or that $\langle y, x\rangle = 0$ for all $x \in \mathcal{C}$.

Theorem 4.9

For any code $\mathcal{C}$,

$$(\mathcal{C}^\perp)^\perp = \mathcal{C}. \tag{4.8}$$

Let us now introduce a few more terms that proves to be really useful later on.

Definition 4.10: Generation Matrix

A generation matrix of a code $\mathcal{C}$ is a matrix $G \in \mathbb{F}_2^{k'\times n}$ whose row space is $\mathcal{C}$. Given a message u, the corresponding code word would then be

$$x = uG. \tag{4.9}$$

The generation matrix can be full rank or not. Generally, it is not, and $k' \geq k$. Some standard forms are of most use:

(a) row reduced echelon form (RREF):

$$G = \begin{bmatrix} 1 & * & * & * & * & * \\ & & 1 & * & * & * \\ & & & 1 & * & * \end{bmatrix}.$$

(b) systematic form:

$$G = \underbrace{\begin{bmatrix} 1 & & & * & * & * \\ & 1 & & * & * & * \\ & & 1 & * & * & * \end{bmatrix}}_{\text{systematic part}\quad\text{check sums}}.$$

In early days of coding theory, it is often designed to have systematic part storing the information that needs to be protected, and the check sum bits are there to protect.

Definition 4.11: Parity-Check Matrix

The parity-check matrix of a code $\mathcal{C}$ is denoted by $H \in \mathbb{F}_2^{h'\times n}$ if H is a generator matrix of the

dual code $\mathcal{C}^\perp$. Or equivalently, the left null space of H^T is $\mathcal{C}$:

$$\mathcal{C} = \left\{ x \in \mathbb{F}_2^{1\times n} \,\middle|\, xH^\mathsf{T} = 0 \right\}. \tag{4.10}$$

Again, H need not be full rank ($h' \geq n-k$), but it will be nice if it is. Moreover, the generator matrix of $\mathcal{C}$ is a parity-check matrix of $\mathcal{C}^\perp$; a parity-check matrix of $\mathcal{C}$ is a generator matrix of $\mathcal{C}^\perp$.

Definition 4.12: Low Density

A matrix with a low number of 1's in its rows is called *low density*. Especially, if the parity matrix H of a code is of low density, this is a low-density parity-check (LDPC) code. A typical LDPC code has only three 1's in each row.

Example: A $(3, 6)$ LDPC code has weight 6 columns and weight 3 rows. □

Remark 4.13

The reason why the number three is chosen as the density of a LDPC code is as demonstrated below:

(a) If $\#1$'s in $H = 0$, then we have that H is the zero matrix, making the parity-check matrix not useful at all.

(b) If $\#1$'s in $H = 1$, then the rows of H must be of the form $y_i = \delta_{ij}$ where j is fixed. Then when used as parity check, this only tells us that $\langle y, x\rangle = 0 \Rightarrow x_j = 0$, which in fact forces there to be zeros in the code.

(c) If $\#1$'s in $H = 2$, then we have that, for example, $y_2 = y_5 = 1$. Hence, $\langle y, x\rangle = 0 \Rightarrow x_2 = x_5$, meaning that x_5 is simply a copy of x_2. Not a particularly good parity check method either.

(d) If $\#1$'s in $H = 3$, then we have that, for exampling a row satisfying $y_2 = y_3 = y_5 = 1$. Hence, we have $\langle y, x\rangle = 0 \Rightarrow x_2 + x_3 + x_5 = 0$. This is a non-trivial equation, and may be really useful in checking for errors and fixing them.

Definition 4.14: Syndrome

For a parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ which is full rank, we can define the map $\psi : \mathbb{F}_2^{1\times n} \to \mathbb{F}_2^{1\times n-k}$

$$\psi(v) = vH^\mathsf{T}. \tag{4.11}$$

The value vH^T is called the *syndrome* of the word v. If $vH^\mathsf{T} \neq 0$, then $v \notin \mathcal{C}$, meaning there is something wrong with v.

Theorem 4.15

An equivalence of the rank-nullity theorem states that

$$\dim \mathcal{C} + \dim \mathcal{C}^\perp = n. \tag{4.12}$$

Proof. Consider the map $\psi(v) = vH^\mathsf{T}$ for $v \in \mathbb{F}_2^{1\times n}$ and H the full rank parity check matrix of the

code $\mathcal{C}$. We then have that

$$\ker \psi = \left\{ x \in \mathbb{F}_2^{1\times n} \,\middle|\, \langle x, y\rangle = 0 \text{ for all row vectors } y \text{ of } H \right\} = \mathcal{C}.$$

Since the H is full rank, we also have that $\operatorname{rank} \psi = n - k$. Hence, by the rank-nullity theorem:

$$n = \operatorname{rank} \psi + \operatorname{null} \psi = (n-k) + k.$$

■

Remark 4.16

It should be noted that although $\mathcal{C} \cup \mathcal{C}^\perp \supseteq \{0\}$, the equality sign does not necessarily holds.

A prime example is the so called *self-dual codes*. An example is the following $[4, 2]$-code:

$$G = H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Another prime example of a self-dual code is some special Reed–Muller codes, which were introduced back in Section 2.7. The proof is given in Section 4.3.

4.2. Punctured and Shortened Codes

A powerful theorem relating the weight enumerator function of a code and its dual is the MacWilliams' identity. The following two operations are introduced so as to prove the MacWilliams' identity stated later, they are operations that reduce a code into smaller subcodes.

Definition 4.17: Punturing

The puncturing of an $[n, k]$-code $\mathcal{C}$ at position i is defined as the following code of block length $n - 1$:

$$\operatorname{Pun}_i(\mathcal{C}) := \{\, x_1 \dots x_{i-1} x_{i+1} \dots x_n \,|\, x = x_1 \dots x_n \in \mathcal{C} \,\}. \tag{4.13}$$

Definition 4.18: Shortening

The shortening of an $[n, k]$-code $\mathcal{C}$ at position i is defined as the following code of block length $n - 1$:

$$\operatorname{Sho}_i(\mathcal{C}) := \{\, x_1 \dots x_{i-1} x_{i+1} \dots x_n \,|\, x = x_1 \dots x_n \in \mathcal{C} \text{ and } x_i = 0 \,\}. \tag{4.14}$$

It is obvious to see that $\operatorname{Sho}_i(\mathcal{C})$ is a subspace of $\operatorname{Pun}_i(\mathcal{C})$. The fact that the two sets defined above are a vector space can be readily checked by the readers.

Theorem 4.19: Duality between Puncturing and Shortening

Given a linear code $\mathcal{C}$, the following two dualities hold:

$$\operatorname{Pun}_i(\mathcal{C}^\perp) = \operatorname{Sho}_i(\mathcal{C})^\perp, \tag{4.15}$$

$$\operatorname{Sho}_i(\mathcal{C}^\perp) = \operatorname{Pun}_i(\mathcal{C})^\perp. \tag{4.16}$$

Since the dual code of a dual code is the original code, we will only be proving Equation (4.16).

Proof. Consider any $x' \in \mathrm{Pun}_i(\mathcal{C})$ and $y' \in \mathrm{Sho}_i(\mathcal{C}^\perp)$. Also define $x \in \mathcal{C}$ and $y \in \mathcal{C}^\perp$ to be the code words that maps to x and y when their ith letter is removed, respectively. Then we have

$$\langle x', y' \rangle = \langle x, y \rangle - x'_i y'_i = -x'_i \cdot 0 = 0.$$

Hence $\mathrm{Pun}_i(\mathcal{C})^\perp \supseteq \mathrm{Sho}_i(\mathcal{C}^\perp)$.

Furthermore, let us count the dimension:

(A1) If $x_i = 0$ for all $x \in \mathcal{C}$, then $(0, \ldots, 0, 1) \in \mathcal{C}^\perp$, and

$$\mathrm{Pun}_i(\mathcal{C}) = \mathrm{Sho}_i(\mathcal{C}) = \mathcal{C} \Rightarrow \dim \mathrm{Pun}_i(\mathcal{C}) = \dim \mathrm{Sho}_i(\mathcal{C}) = \dim \mathcal{C}.$$

(A2) If $(0, \ldots, 0, 1, 0, \ldots, 0) \in \mathcal{C}$ (1 at ith position), then $x_i^\perp = 0$ for all $x^\perp \in \mathcal{C}^\perp$, and

$$\mathrm{Pun}_i(\mathcal{C}) = \mathrm{Sho}_i(\mathcal{C}) \Rightarrow \left|\mathrm{Pun}_i(\mathcal{C})\right| = \left|\mathrm{Sho}_i(\mathcal{C})\right| = \frac{1}{2}|\mathcal{C}|\,.$$

(A3) If some $x_i = 1$ but $(0, \ldots, 0, 1, 0, \ldots, 0) \notin \mathcal{C}$, then some $x_i^\perp = 1$ but $(0, \ldots, 0, 1, 0, \ldots, 0) \notin \mathcal{C}^\perp$, and

$$\left|\mathrm{Sho}_i(\mathcal{C})\right| = \frac{1}{2}|\mathcal{C}|\,, \left|\mathrm{Pun}_i(\mathcal{C})\right| = |\mathcal{C}|\,.$$

The notation $|\mathcal{C}|$ denotes the number of codewords in the code $\mathcal{C}$. For all three cases above, we have that

$$\dim \mathrm{Pun}_i(\mathcal{C}) + \dim \mathrm{Sho}_i(\mathcal{C}^\perp) = n - 1.$$

Since the dimension of $\mathrm{Pun}_i(\mathcal{C})^\perp$ and $\mathrm{Sho}_i(\mathcal{C}^\perp)$ are the same, as well as the former includes the latter, they are the same subspace. ∎

The technique of dividing the problem on a length n code into its puncturing and shortening of length $n-1$ is really useful. Combined with the duality between puncturing and shortening, we here present a very non-trivial result on the relationship between the weight enumerator of a code and its dual.

Theorem 4.20: MacWilliams' Identity

Denote the number of codewords in a code $\mathcal{C}$ as $|\mathcal{C}|$, then the following identity holds:

$$A_{\mathcal{C}^\perp}(x, y) = \frac{1}{|\mathcal{C}|} A_{\mathcal{C}}(y - x, y + x). \tag{4.17}$$

Proof. Let us abbreviate $A_{\mathrm{Pun}_i(\mathcal{C})}$ by $A_{\mathrm{P}_i(\mathcal{C})}$ and A_{Sho_i} by $A_{\mathrm{S}_i(\mathcal{C})}$. Continue on with (A1) to (A3), for a $w = (w_1, w_2, \ldots, w_n) \in \mathcal{C}$,

(B) If $w_n = 0$, w contributes

(B1) $x^{|w|}y^{n-|w|}$ to $A_{\mathcal{C}}$,

(B2) $x^{|w|}y^{n-1-|w|}$ to $A_{\mathrm{P}_n(\mathcal{C})}$, and

(B3) $x^{|w|}y^{n-1-|w|}$ to $A_{\mathrm{S}_n(\mathcal{C})}$.

(C) If $w_n = 1$,

(C1) w contributes $x^{|w|}y^{n-|w|}$ to $A_{\mathcal{C}}$, and

(C2) further if (A3), w contributes $x^{|w|-1}y^{n-1-|w|-1} = x^{|w|-1}y^{n-|w|}$ to $A_{\mathrm{P}_n(\mathcal{C})}$.

For the cases below, we can represent $A_{\mathcal{C}}$ simply using $A_{\mathrm{P}_n(\mathcal{C})}$ and $A_{\mathrm{S}_n(\mathcal{C})}$:

- If (A1), then

$$\begin{aligned} A_{\mathcal{C}} = \Bigg(\overbrace{\sum_{w_n=1}}^{\text{(C1)}} + \overbrace{\sum_{w_n=0}}^{\text{(B1)}}\Bigg) x^{|w|}y^{n-|w|} &= 0 + y \cdot \text{(B2)} = yA_{\mathrm{P}_n(\mathcal{C})} \\ &= 0 + y \cdot \text{(B3)} = yA_{\mathrm{S}_n(\mathcal{C})}. \end{aligned}$$

- If (A2), then since $\mathrm{Pun}_n(\mathcal{C}) = \mathrm{Sho}_n(\mathcal{C})$ and $\mathcal{C} = \big(\mathrm{Pun}_n(\mathcal{C}), 1\big) \cup \big(\mathrm{Pun}_n(\mathcal{C}), 0\big)$,

$$\begin{aligned} A_{\mathcal{C}} = \text{(C1)} + \text{(B1)} = (x+y) \cdot \text{(B2)} &= (x+y)A_{\mathrm{P}_n(\mathcal{C})} \\ = (x+y) \cdot \text{(B2)} &= (x+y)A_{\mathrm{S}_n(\mathcal{C})}. \end{aligned}$$

- If (A3), then

$$\begin{aligned} A_{\mathcal{C}} = \text{(C1)} + \text{(B1)} = x \cdot \text{(C2)} + y \cdot \text{(B3)} &= xA_{\mathrm{P}_n(\mathcal{C}),w_n=1} + yA_{\mathrm{S}_n(\mathcal{C})} \\ &= x\left(A_{\mathrm{P}_n(\mathcal{C})} - A_{\mathrm{S}_n(\mathcal{C})}\right) + yA_{\mathrm{S}_n(\mathcal{C})} = (y-x)A_{\mathrm{S}_n(\mathcal{C})} + xA_{\mathrm{P}_n(\mathcal{C})}. \end{aligned}$$

Now, let us combine everything. In the base case where the code has length 1, $\mathcal{C} = \{0\}$ or $\{0,1\}$, with $\mathcal{C}^\perp = \{0,1\}$ or $\{0\}$, respectively. Without loss of generality, let us consider $\mathcal{C} = \{0\}$ and $\mathcal{C}^\perp = \{0,1\}$, with $A_{\mathcal{C}}(x,y) = y, |\mathcal{C}| = 1$ and $A_{\mathcal{C}^\perp}(x,y) = x+y$. The statement obviously holds. Now let us suppose that codes up to length $n-1$ satisfy the statement. For a code $\mathcal{C}$ of length n, three cases can be considered:

- If (A1),

$$\begin{aligned} A_{\mathcal{C}^\perp}(x,y) = (x+y)A_{\mathrm{P}_n(\mathcal{C}^\perp)}(x,y) &= (x+y)A_{\mathrm{S}_n(\mathcal{C})^\perp}(x,y) \\ \overset{\text{(i)}}{=} \frac{x+y}{\big|\mathrm{S}_n(\mathcal{C})\big|}A_{\mathrm{S}_n(\mathcal{C})}(y-x, y+x) &= \frac{1}{|\mathcal{C}|}A_{\mathcal{C}}(y-x, y+x). \end{aligned}$$

- If (A2),

$$\begin{aligned} A_{\mathcal{C}^\perp}(x,y) = yA_{\mathrm{P}_n(\mathcal{C}^\perp)}(x,y) &= yA_{\mathrm{S}_n(\mathcal{C})^\perp}(x,y) \\ \overset{\text{(i)}}{=} \frac{y}{\big|\mathrm{S}_n(\mathcal{C})\big|}A_{\mathrm{S}_n(\mathcal{C})}(y-x, y+x) &= \frac{(y-x)+(y+x)}{2\big|\mathrm{S}_n(\mathcal{C})\big|}A_{\mathrm{S}_n(\mathcal{C})}(y-x, y+x) \\ = \frac{1}{|\mathcal{C}|}A_{\mathcal{C}}(y-x, y+x). \end{aligned}$$

- If (A3),

$$\begin{aligned} A_{\mathcal{C}^\perp}(x,y) = (y-x)A_{\mathrm{S}_n(\mathcal{C}^\perp)}(x,y) + xA_{\mathrm{P}_n(\mathcal{C}^\perp)}(x,y) &= (y-x)A_{\mathrm{P}_n(\mathcal{C})^\perp}(x,y) + xA_{\mathrm{S}_n(\mathcal{C})^\perp}(x,y) \\ \overset{\text{(i)}}{=} \frac{y-x}{\big|\mathrm{P}_n(\mathcal{C})\big|}A_{\mathrm{P}_n(\mathcal{C})}(y-x, y+x) + \frac{x}{\big|\mathrm{S}_n(\mathcal{C})\big|}A_{\mathrm{S}_n(\mathcal{C})}(y-x, y+x) \\ = \frac{1}{|\mathcal{C}|}\Big((y-x)A_{\mathrm{P}_n(\mathcal{C})}(y-x, y+x) + ((y+x)-(y-x))A_{\mathrm{S}_n(\mathcal{C})}(y-x, y+x)\Big) \\ = \frac{1}{|\mathcal{C}|}A_{\mathcal{C}}(y-x, y+x). \end{aligned}$$

The $\overset{\text{(i)}}{=}$ implies by induction. Henceforth, the statement holds for all cases, and the theorem is proven. ■

We see from the above proof a repeating scheme for proving these statements. First we reduce a code into a combination of the subcodes of puncturing and shortening, and three cases should be considered. Then by induction, the statement can be proven. We will see this technique being used yet again for the upcoming two lemmas. 1 Apr.

Definition 4.21: Corank-Nullity Generating Function

Let $G \in \mathbb{F}_2^{k\times n}$ be a generator matrix of an $[n,k]$-linear code $\mathcal{C}$. The *corank-nullity generating polynomial* of $\mathcal{C}$ is

$$W_{\mathcal{C}}(x,y) := \sum_{S\subseteq E} x^{k-r(S)} y^{|S|-r(S)}. \tag{4.18}$$

The notation W is after Whitney, as $W_{\mathcal{C}}$ is also known as the Whitney rank generating function. The sum enumerates over all subsets S of the columns of G where $E = \{1,2,\dots,n\}$, $|S|$ is the number of columns in S, and $r(S)$ is the rank of G restricted to the columns S. Usually, $k-r(S)$ is called the *corank*, and $|S|-r(S)$ is called the *nullity*.

Definition 4.22: Tutte Polynomial

The *Tutte polynomial* of $\mathcal{C}$ is

$$T_{\mathcal{C}}(x,y) := \sum_{S\subseteq E} (x-1)^{k-r(S)} (y-1)^{|S|-r(S)}. \tag{4.19}$$

Note that Tutte and Whitney polynomials are the same thing with a small change of variables.

Lemma 4.23

The corank-nullity generating function of a code $\mathcal{C}$ is independent of the generator matrix G used. Hence it is well defined.

The above lemma can easily be shown by observing that adding redundant rows or applying linear combinations over the rows of a generating matrix do not affect the k, $r(S)$ and $|S|$ of a given subset of columns.

The corank-nullity generating function is introduced as a tool for us to proof the MacWilliams' identity. By proving the following two lemmas over its symmetry and its relation with the weight enumerator, we will finally obtain the MacWilliams' identity.

Lemma 4.24: Corank-Nullity Generating Function of the Dual Code

For $\mathcal{C}^\perp$ the dual code of $\mathcal{C}$,

$$W_{\mathcal{C}^\perp}(x,y) = W_{\mathcal{C}}(y,x). \tag{4.20}$$

Proof. In this proof, we will try to relate $\mathcal{C}$ with $\mathrm{Pun}_n(\mathcal{C})$ and $\mathrm{Sho}_n(\mathcal{C})$, finally obtaining the result via induction. Three different cases can be considered:

- If (A1), then we separate S's into those that include the nth column and those that doesn't. Since $k_{\mathrm{P}_n(\mathcal{C})} = k_{\mathcal{C}}$, $r(S)_{\mathrm{P}_n(\mathcal{C})} = r(S)_{\mathcal{C},S \text{ without col. } n} = r(S)_{\mathcal{C},S \text{ with col. } n}$, $|S|_{\mathrm{P}_n(\mathcal{C})} = |S|_{\mathcal{C},S \text{ without col.n}}$, and $|S|_{\mathrm{P}_n(\mathcal{C})} = |S|_{\mathcal{C},S \text{ with col.n}} - 1$,

$$W_{\mathcal{C}}(x,y) = W_{\mathcal{C},S \text{ without col. } n}(x,y) + W_{\mathcal{C},S \text{ with col. } n}(x,y) = (1+y)W_{\mathrm{P}_n(\mathcal{C})}(x,y).$$

- If (A2), since $k_{\mathrm{S}_n(\mathcal{C})} = k_{\mathcal{C}} - 1$, $r(S)_{\mathrm{S}_n(\mathcal{C})} = r(S)_{\mathcal{C},S \text{ without col. } n} = r(S)_{\mathcal{C},S \text{ with col. } n} - 1$, $|S|_{\mathrm{S}_n(\mathcal{C})} = |S|_{\mathcal{C},S \text{ without col.n}}$, and $|S|_{\mathrm{S}_n(\mathcal{C})} = |S|_{\mathcal{C},S \text{ with col.n}} - 1$,

$$W_{\mathcal{C}}(x,y) = (1+x)W_{\mathrm{S}_n(\mathcal{C})}(x,y).$$

- If (A3), then

$$W_{\mathcal{C}}(x,y) = W_{\mathrm{P}_n(\mathcal{C})}(x,y) + W_{\mathrm{S}_n(\mathcal{C})}(x,y).$$

Then the lemma can easily be proven via induction. ■

Lemma 4.25: (Greene's)

The weight enumerator polynomial and the corank-nullity generating function can be related by

$$A_{\mathcal{C}}(x,y) = (y-x)^k x^{n-k} W_{\mathcal{C}}\left(\frac{2x}{y-x}, \frac{y-x}{x}\right). \tag{4.21}$$

Proof. Again consider the three cases (A1), (A2), and (A3). The lemma can be proven by utilizing induction on the decomposition of a code $\mathcal{C}$ into its subcodes $\mathrm{Pun}_n(\mathcal{C})$ and $\mathrm{Sho}_n(\mathcal{C})$ and the subsequent decomposition of $A_{\mathcal{C}}$ and $W_{\mathcal{C}}$ seen in the proofs of Theorem 4.20 and Lemma 4.24, respectively. ■

For a more elegant proof of the two lemmas above, see Appendix A.1.1 and Appendix A.1.2. These two lemmas can be used to give a cleaner derivation of the MacWilliams' identity, which is presented in Appendix A.1.3.

4.3. Dual of Reed–Muller and Reed–Solomon Code

This section proves the statement regarding the dual of a Reed–Muller code mentioned back in Remark 4.16. Along the way, we will ways to re-characterize the Reed–Muller code, and also introducing the Reed–Solomon which is very much similar.

Here we begin by providing a different definition to the Reed–Muller code from the one provided in Section 2.7.

Definition 4.26: Reed–Muller Code

The Reed–Muller code is defined as

$$\mathrm{RM}(r,m) = \left\{ \big(f(0), f(1), \ldots, f(2^m-1)\big) \;\middle|\; f \in \mathbb{F}_2[x_1,\ldots,x_m]^{\leq r} \text{ where } x_i \in \mathbb{F}_2 \right\}. \tag{4.22}$$

The set $\mathbb{F}_2[x]^{\leq r}$ denotes the polynomials in x with degree less than or equal to r.

Example: Consider $\mathrm{RM}(2,3)$, we have the following monomials as a basis to all $f \in \mathbb{F}_2[x_1,x_2,x_3]^{\leq 2}$, the respective codewords are as follows.

f	$\big(f(000), f(001), f(010), f(011), f(100), f(101), f(110), f(111)\big)$
0	$(0,0,0,0,0,0,0,0)$
1	$(1,1,1,1,1,1,1,1)$
x_1	$(0,0,0,0,1,1,1,1)$
x_2	$(0,0,1,1,0,0,1,1)$
x_3	$(0,1,0,1,0,1,0,1)$
x_1x_2	$(0,0,0,0,0,0,1,1)$
x_1x_3	$(0,0,0,0,0,1,0,1)$
x_2x_3	$(0,0,0,1,0,0,0,1)$

All other codewords can be generated as linear combination of the bases above. □

Theorem 4.27: Minimum Distance of RM

The minimum distance of $\mathrm{RM}(r,m)$ is 2^{m-r}.

Proof. Can be shown by induction on r. ■

Theorem 4.28: Dual of RM

The dual code to the Reed–Muller code $\mathrm{RM}(r,m)$ is again another Reed–Muller code:

$$\mathrm{RM}(r,m)^{\perp} = \mathrm{RM}(m-r-1,m). \tag{4.23}$$

Proof. Consider a g a polynomial in $x_1,\ldots,x_m$ with $\mathbb{F}_2$ coefficients that should generate the dual code to $\mathrm{RM}(r,m)$. Then for all $f \in \mathbb{F}_2[x_1,\ldots,x_m]^{\leq r}$, g should satisfy

$$\sum_{i=0}^{2^m-1} f(i)g(i) = 0.$$

The dimension of all possible g is then

$$\begin{aligned}\dim \mathrm{RM}(r,m)^{\perp} &= 2^m - \underbrace{\left[\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r}\right]}_{\dim \mathrm{RM}(r,m)} \\ &= \binom{m}{r+1} + \ldots + \binom{m}{m} = \binom{m}{m-r-1} + \binom{m}{0} = \dim \mathrm{RM}(m-r-1,m).\end{aligned}$$

■

Definition 4.29: Reed–Solomon Code

A Reed–Solomon code is defined with n distinct points $a_1,\ldots,a_n \in \mathbb{F}_p$ as follows

$$\mathrm{RS}(a_{1:n},k) = \left\{ \big(f(a_1),\ldots,f(a_n)\big) \;\middle|\; f \in \mathbb{F}_p[x]^{<k} \right\}. \tag{4.24}$$

Theorem 4.30: Minimum Distance of RS

The minimum distance of $\mathrm{RS}(a_{1:n}, k)$ is $n - k + 1$.

Proof. Consider the degrees of freedom: a degree $k-1$ single variable polynomial is uniquely defined by k parameters. ■

The dual of a Reed–Solomon code is not exactly another Reed–Solomon code. Some extensions are needed.

Definition 4.31: Generalized Reed–Solomon Code

A generalized Reed–Solomon code is defined with $2n$ distinct points $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{F}_p$ as follows

$$\mathrm{GRS}(b_{1:n}, a_{1:n}, k) = \left\{ \big(b_1 f(a_1), \ldots, b_n f(a_n)\big) \;\middle|\; f \in \mathbb{F}_p[x]^{<k} \right\}. \tag{4.25}$$

Theorem 4.32: Dual of GRS

The dual code to the generalized Reed–Solomon code is again another generalized Reed–Solomon code:

$$\mathrm{GRS}(b_{1:n}, a_{1:n}, k)^{\perp} = \mathrm{GRS}(c_{1:n}, a_{1:n}, n-k), \tag{4.26}$$

where $c_i^{-1} = b_i \prod_{j \neq i}(a_j - a_i)$.

Proof. For a code $\mathcal{C} = \mathrm{GRS}(b_{1:n}, a_{1:n}, k)$, consider another code

$$\mathcal{C}' = \left\{ \big(c_1, f(a_1), \ldots, c_n f(a_n)\big) \;\middle|\; f \in \mathbb{F}_p[t]^{<n-k} \right\} = \mathrm{GRS}(c_{1:n}, a_{1:n}, n-k).$$

If codewords from $\mathcal{C}'$ were to be orthogonal to $\mathcal{C}$, then we only need to demonstrate the orthogonality of the bases: the monomials. Consider the monomials $g(t) = t^{\ell} \in \mathbb{F}_p[t]^{<n-k}$ and $f(t) = t^m \in \mathbb{F}_p[t]^{<k}$, we have that $\ell + m < n - 1$, and

$$0 = \sum_{i=1}^{n} b_i f(a_i) \cdot c_i g(a_i) = \sum_{i=1}^{n} c_i b_i a_i^{\ell+m}.$$

Since the number of variables is n and the number of constraints is $n-1$, a single non-zero solution for c_i(including its span) exists. Hence, $\mathcal{C}' \subseteq \mathcal{C}^{\perp}$. Lastly, by checking the dimension: $\dim \mathcal{C}' = n - k = \dim \mathcal{C}^{\perp}$, we can now state for sure that $\mathcal{C}' = \mathcal{C}^{\perp}$.

The value to c_i can be directly found by solving

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-2} & a_2^{n-2} & \cdots & a_n^{n-2} \end{bmatrix} \begin{bmatrix} b_1 c_1 \\ b_2 c_2 \\ \vdots \\ b_n c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

We can add an ancillary constraint such that

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{bmatrix} \begin{bmatrix} b_1 c_1 \\ b_2 c_2 \\ \vdots \\ b_n c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \tag{4.27}$$

Now we can utilize what we know about the Vandermonde determinant to aid our analysis:

$$\begin{bmatrix} b_1c_1 \\ b_2c_2 \\ \vdots \\ b_nc_n \end{bmatrix} = \frac{1}{\Pi_{j>i}(a_j - a_i)} \begin{bmatrix} \Pi_{j>i\neq 1}(a_j - a_i) \\ \Pi_{j>i\neq 2}(a_j - a_i) \\ \vdots \\ \Pi_{j>i\neq n}(a_j - a_i) \end{bmatrix}.$$

Hence, $b_ic_i = [\Pi_{j\neq i}(a_j - a_i)]^{-1}$. The inclusion of 1 in Equation (4.27) is arbitrary, and any multiple of it works. ■

4.4. Algebra of Finite Fields

14 Apr. Birthday

Reed–Solomon codes are really useful and we would like to extend it from $\mathbb{F}_p$ to more general fields of order (size) $q = p^d$. Henceforth, some algebra is needed as prior knowledge. For more details, please refer to the book "Essential Coding Theory" [VS23].

As one already knows, a field $\mathbb{F}$ is a set with the usual addition, subtraction, multiplication, division, zero element, and multiplicative identity. A finite field is a field with finite elements. Finite fields are often called Galois fields, denoted by $\mathrm{GF}(q)$. Some common examples of a field are the usual $\mathbb{R}$, $\mathbb{C}$, and $\mathbb{F}_p \cong \mathbb{Z}_p$ where p is a prime.

Definition 4.33: Character

The character of a field is defined as

$$\operatorname{char}\mathbb{F} := \text{the smallest positive integer } c \text{ such that } \underbrace{1+1+\cdots+1}_{c \text{ copies}} = 0. \tag{4.28}$$

Example:

- We have $\operatorname{char}\mathbb{C} = \infty$ and $\operatorname{char}\mathbb{R} = \infty$, they are termed *characteristic-zero fields*.
- $\operatorname{char}\mathrm{GF}(p) = p$.

□

Theorem 4.34

For a finite field $\mathbb{F}$, $\operatorname{char}\mathbb{F}$ is a prime.

Proof. If we have $\operatorname{char}\mathbb{F} = ab$ as a product of two integers a and b bigger than 1, then

$$0 = \underbrace{1+1+\cdots+1}_{\times ab} = \underbrace{(1+\cdots+1)}_{\times a} \cdot \underbrace{(1+\cdots+1)}_{\times b},$$

meaning that one of a and b cannot have their inverse defined: $1 \cdot b = a^{-1}ab = a^{-1}0 = 0$. Thus, a contradiction is met, $\operatorname{char}\mathbb{F}$ can only be zero or prime. ■

Definition 4.35: Base Field

For any $\mathbb{F}$, if $\operatorname{char}\mathbb{F} > 0$, the field $\mathrm{GF}(\operatorname{char}\mathbb{F})$ is termed the *base field*, or the *prime subfield* of $\mathbb{F}$.

If we treat $\mathbb{F}$ as a vector space, then its base field is like the field the vector space is defined on. See the theorem below for a complete statement. For more details, refer to Theorem 4.42.

Theorem 4.36

For any field $\mathbb{F}$, we have its order (size) being $|\mathbb{F}| = p^n$, where $\operatorname{char}\mathbb{F} = p$ and n is the dimension of $\mathbb{F}$ over its base field $\mathrm{GF}(p)$. We have $\mathbb{F} \cong \mathrm{GF}(p)^n$ as a vector space.

Definition 4.37

The set $\mathrm{GF}(p)[x]$ is the polynomials in x with coefficients in $\mathrm{GF}(p)$. We say that $f \in \mathrm{GF}(p)[x]$, or that f is a polynomial over $\mathrm{GF}(p)$.

Definition 4.38: Reducibility

A polynomial $f \in \mathrm{GF}(p)[x]$ is *reducible* if there exists polynomials $g, h \in \mathrm{GF}(p)[x]$ with $\deg g$, $\deg h > 0$, such that $f = gh$. Else, the polynomial is called *irreducible*.

Example:

- The polynomial $x^2 + 1$ over $\mathrm{GF}(2)$ is reducible, since $(x+1)^2 = x^2 + 2x + 1 = x^2 + 1$.
- The polynomial $x^2 + x + 1$ over $\mathrm{GF}(2)$ can be shown to be irreducible by enumerating all possibilities.

□

4.4.1. Constructing a Finite Field

It is interesting to note that every time when we have an irreducible polynomial over a field, we can construct another finite field! Our end goal is to construct, from $\mathrm{GF}(p)$ where p is prime, into a general field $\mathrm{GF}(q)$ where $q = p^n$, showing its existence and uniqueness. The construction procedure is as below

Definition 4.39: Companion Matrix

Given a polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{d-1}x^{d-1} + x^d \in \mathrm{GF}(p)[x], \tag{4.29}$$

the companion matrix of f is defined as

$$A = \begin{bmatrix} 0 & 0 & & & -a_0 \\ 1 & 0 & & & -a_1 \\ 0 & 1 & \ddots & & \vdots \\ & & \ddots & 0 & -a_{d-2} \\ & & & 1 & -a_{d-1} \end{bmatrix} \in \mathrm{GF}(p)^{d\times d}, \tag{4.30}$$

with the right-most column containing the negative coefficients to f, the lower-left triangular containing an identity matrix of size $(d-1)\times(d-1)$, and the remaining entries all being 0.

Theorem 4.40

Let the companion matrix of the polynomial f be denoted by A. The characteristic polynomial of A is equal to f.

Proof. This can be directly shown by computing the characteristic polynomial from definition: $\det(\lambda\mathbb{1} - A)$. But we will show another method here. Observe the powers to A:

$$A^0 = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}, A^1 = \begin{bmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ & 1 & \ddots & & \vdots \\ & & \ddots & 0 & -a_{d-2} \\ & & & 1 & -a_{d-1} \end{bmatrix}, A^2 = \begin{bmatrix} 0 & & & & -a_0 & * \\ 0 & 0 & & & -a_1 & * \\ 1 & 0 & \ddots & 0 & \vdots & * \\ & 1 & \ddots & 0 & -a_{d-2} & * \\ & & & 1 & -a_{d-1} & * \end{bmatrix},$$

$$\ldots, A^{d-1} = \begin{bmatrix} 0 & -a_0 & * & \cdots & * \\ 0 & -a_1 & * & \cdots & * \\ \vdots & \vdots & * & \cdots & * \\ 0 & -a_{d-2} & * & \cdots & * \\ 1 & -a_{d-1} & * & \cdots & * \end{bmatrix}, A^d = \begin{bmatrix} -a_0 & * & \cdots & * & * \\ -a_1 & * & \cdots & * & * \\ \vdots & * & \cdots & * & * \\ -a_{d-2} & * & \cdots & * & * \\ -a_{d-1} & * & \cdots & * & * \end{bmatrix}.$$

From the Cayley–Hamilton theorem, we know that the characteristic polynomial, and that A^d can be represented as a linear combination of A^0 to A^{d-1}. Comparing the coefficients to the first column, we can immediately see that

$$A^d = -a_0 A^0 - a_1 A^1 - a_2 A^2 - \cdots - a_{d-1} A^{d-1}.$$

Hence it is shown. ■

Remark 4.41

In MATLAB and many other mathematical programs, given a polynomial f, a simple method for finding the roots to f is to first construct its companion matrix A, then find the eigenvalues to A. The eigenvalues of A are identical to the roots of f.

Theorem 4.42

Given an irreducible polynomial f over $\mathrm{GF}(p)$, let the companion matrix of f be denoted by A, then the set

$$\mathcal{F} := \left\{ b_0 + b_1 A + \cdots + b_m A^m \,\middle|\, b_i \in \mathrm{GF}(p), m > 0 \right\} \subseteq \mathrm{GF}(p)^{d \times d} \tag{4.31}$$

is a finite field of order p^d.

Proof. First of all, again by Cayley–Hamilton theorem, A^d can be represented as a degree $d-1$ polynomial of A, hence, we can restrict $m \leq d-1$. The set $\mathcal{F}$ is thus finite. Next, we need to check the closure of the four operations.

- Both addition and subtraction are trivial.

- Multiplication is just the convolution of coefficients, with suitable application of the Cayley–Hamilton theorem from time to time.
- Division is more involved: Let us first consider how division works in $\mathrm{GF}(p)$. For the first few integers, division (finding their inverse) is easy, we have

$$\begin{aligned}\frac{1}{2} &= \frac{p+1}{2},\\ \frac{1}{3} &= \frac{p+1}{3} \text{ or } \frac{2p+1}{3},\\ \frac{1}{4} &= \left(\frac{1}{4}\right)^2, \cdots .\end{aligned}$$

  However, in implementations, we can not afford this trial-and-error strategy of finding inverses, it is way too time-consuming. What we can do instead is applying a famous result in number theory – the Bézout's lemma.

Lemma 4.43: Bézout's Lemma

If $\gcd(p, k) = 1$, then there exists integers a and b such that $ap + bk = 1$.

  This lemma can be proven by induction on the Euclidean algorithm.

  Then the procedure to finding $1/k$ for any $k \in \mathrm{GF}(p)$ will be to first find $ak + bp = 1$ via the extended Euclidean algorithm, then since $ak \equiv 1$ modulo p, we have $a = 1/k$ in $\mathrm{GF}(p)$.

  Let us now extend the above to the set $\mathcal{F}$ and polynomials. For f the characteristic polynomial of A, consider f and g are coprime polynomials in $\mathrm{GF}(p)[x]$. Coprimality of polynomials means that they do not share any factors, and since f is irreducible, any g over $\mathrm{GF}(p)$ must be coprime to f. By an extension to the Bézout's lemma, there exists polynomials a and b such that $af+bg = 1$, the constant 1 polynomial. And since f satisfies $f(A) = 0$, we have that

$$b(A) \cdot g(A) = \mathbb{1},$$

  meaning that $g(A)^{-1} = b(A)$.

Having all the operations checked, we can now safely state that the set $\mathcal{F}$ is indeed a field of order $|F| = p^d$. ■

Example: We have the following field of

$$\mathrm{GF}(4) \cong \left\{ \underbrace{\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}}_{0}, \underbrace{\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}}_{1}, \underbrace{\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}}_{A}, \underbrace{\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}}_{B} \right\}.$$

The set satisfies $1 + A = B$, $1 + B = A$, $AB = 1$, $A^2 = B$, and $B^2 = A$. In fact, we can see that this field is generated by the degree 2 irreducible polynomial $f(x) = 1 + x + x^2$ over $\mathrm{GF}(2)$, where B is the companion matrix of f. □

In conclusion, we can generate the finite field $\mathrm{GF}(p^n)$ by considering an irreducible polynomial f of degree $n-1$ over $\mathrm{GF}(p)$, then we find the companion matrix A of f, lastly, we have the field extension $\mathrm{GF}(p^n) \cong \mathrm{GF}(p)[A]$.

Remark 4.44

A more often-seen scheme to introducing finite fields of general orders is by the notion of a quotient. Given an irreducible polynomial $f \in \mathrm{GF}(p)[x]$, the quotient

$$\mathrm{GF}(p)[x]/(f(x)) \tag{4.32}$$

is a field.

There are a few things we have not yet checked, however. To be precise, we have yet check the *existence* of irreducible polynomials of any degree over $\mathrm{GF}(p)$, nor the *uniqueness* of $\mathrm{GF}(p^n)$ when generated using different irreducible polynomials of same degree.

4.4.2. Uniqueness of a Finite Field

In this subsection, we will discuss the uniqueness of a finite field given its order. We further assume the existence of a finite field. Let $\mathbb{F}$ be a finite field, denote the abelian (commutative) group of

$$\mathbb{F}^\times := \left\{ B \in \mathbb{F} \,\middle|\, B^{-1} \text{ exists} \right\} = \mathbb{F}\backslash\{0\}. \tag{4.33}$$

We further denote $|\mathbb{F}| = p^n$, and hence $\left|\mathbb{F}^\times\right| = p^n - 1$.

Lemma 4.45

The set $\mathbb{F}^\times$ is a cyclic group, i.e., there exists an element $B \in \mathbb{F}$ such that

$$\mathbb{F}^\times = \{1, B, B^2, \ldots, B^{p^n-2}\}.$$

Before we proof this lemma, we must first show that it is indeed non-trivial. The more educated ones may argue that a finite abelian group can always be written as a finite direct product of cyclic groups: $G = \mathbb{Z}_{r_1} \times \mathbb{Z}_{r_2} \times \cdots$. So how can we be so sure that $\mathbb{F}^\times$ is a cyclic group and not a direct product of cyclic groups? A straightforward counter example will be that $\mathbb{F} \times \mathbb{F}$ is not a field! Since $(1,0)\cdot(0,1) = (0,0)$, $(1,0)$ is a zero divisor.

Proof. For every k in 1 to $p^n - 1$, let us denote the set

$$T_k = \left\{ B \in \mathbb{F} \,\middle|\, \mathrm{order}(B) = k \right\},$$

where order of an element is defined by the minimum $k > 0$ such that $B^k = 1$. Define S_k as the set of solutions to $x^k - 1 = 0$, and $T_k \subseteq S_k$. Since the size of S_k is at most k, $|T_k| \leq |S_k| \leq k$. From this, one can deduce that the number of elements in T_k must be $|T_k| \leq \varphi(k)$. The function $\varphi(k)$ is the Euler totient function, denoting the amount of numbers coprime to k from 1 to k. Thus, we have

$$\mathbb{F}^\times = \bigsqcup_{k|p^n-1} T_k \Rightarrow p^n - 1 = \left|\mathbb{F}^\times\right| = \sum_{k|p^n-1} |T_k| \leq \sum_{k|p^n-1} \varphi(k) = p^n - 1.$$

The last equality is a famous result from number theory. Hence, $|T_k| = \varphi(k)$ for all $k \mid n$. The notation $k \mid n$ reads "n is divisible by k." Finally, we know there exists at least an element in $\mathbb{F}^\times$ that has order $p^n - 1$, it is a cyclic group. ∎

Every $0 \neq B \in \mathbb{F}$ satisfying $B^{p^n-1} = 1$ is a root to $x^{p^n-1} - 1 = 0$, there can be at most $p^n - 1$ of them, and we have just shown that there are indeed $p^n - 1$ of them. We often write

$$x^{p^n-1} - 1 = \prod_{B\in\mathbb{F}, B\neq 0} (x - B) \Longleftrightarrow x^{p^n} - x = \prod_{B\in\mathbb{F}} (x - B). \tag{4.34}$$

Since all elements of a finite field of order p^n must be the solution to the polynomial $x^{p^n} - x$, all finite fields of order p^n are isomorphic to each other, i.e., their differences can only be a similarity transformation on the matrix elements.

4.4.3. Existence of a Finite Field

Lastly, let us prove the existence of a finite field by showing the existence of an irreducible polynomial of any order d over any prime field $\mathrm{GF}(p)$.

We will show the existence by building from a couple of lemmas.

Lemma 4.46

If f is an irreducible polynomial of degree d over $\mathrm{GF}(p)$, then $f \mid x^{p^d} - x$.

Proof. Since $f(x) = \prod(x - B)$, where $B \in \mathrm{GF}(p)[x]/(f) \cong \mathrm{GF}(p^d)$ enumerates over all roots of f. And we also have $(x - B) \mid x^{p^d} - x$. ■

Lemma 4.47

For all m, $x^{p^d} - x \mid x^{p^{md}} - x$.

Proof. Consider any root B to $x^{p^d} - x = 0$, it satisfies $B^{p^d} = B$. Then, $B^{p^{md}} = (B^{p^d})^{p^{(m-1)d}} = B^{p^{(m-1)d}} = \ldots = B$. ■

Combining the above two lemmas, we obtain the following.

Lemma 4.48

If f is an irreducible polynomial of degree d, then $f \mid x^{p^n} - x$ where $d \mid n$.

This leads to a final step in showing the existence of $\mathrm{GF}(p^n)$:

Theorem 4.49

Let $M(d)$ denote the number of irreducible polynomials over $\mathrm{GF}(p)$ of order d, then

$$\sum_{d|n} d \cdot M(d) = p^n \tag{4.35}$$

Proof. Since for any irreducible polynomial f over $\mathrm{GF}(p)$ of degree d with, we have $f \mid x^{p^d} - x$ where $d \mid n$, then

$$\prod_{f:\text{irred.},\deg f \mid n} f \;\Bigg|\; x^{p^n} - x. \tag{4.36}$$

Counting the highest degree, we have

$$\sum_{d|n} d \cdot M(d) \leq p^n.$$

Next, for any $B \in \mathrm{GF}(p^n)$, let f be its *minimum polynomial*, which is the polynomial over $\mathrm{GF}(p)$ with minimum degree such that B is root of it. Consider the field extension $\mathrm{GF}(p)(B) \cong \mathrm{GF}(p)[x]/(f)$. Since $B \in \mathrm{GF}(p^n)$, we have

$$\underbrace{[\mathrm{GF}(p)(B) : \mathrm{GF}(p)]}_{\deg f} \;\Big|\; [\mathrm{GF}(p^n) : \mathrm{GF}(p)(B)].$$

The notation $[K : F]$ means the dimension of the vector space K over the field F. Since each element in $B \in \mathrm{GF}(p^n)$ maps to an irreducible polynomial f_B of degree $d \mid n$, and f_B is mapped to at least d times since it has d distinct roots,

$$\begin{aligned}
\left|\mathrm{GF}(p^n)\right| = \#\left\{ B \,\middle|\, B \in \mathrm{GF}(p^n) \right\} &= \sum_{d|n} \#\left\{ B \,|\, \deg f_B = d \right\} \\
&\leq \sum_{d|n} d\#\left\{ f \in \mathrm{GF}(p)[x] \,\middle|\, \exists\, B \in \mathrm{GF}(p^n), f_B = f, \deg f_B = d \right\} \\
&\leq \sum_{d|n} d \cdot M(d).
\end{aligned}$$

The equality is proven. ∎

Finally, since $M(d) \leq p^d$, if $M(n) = 0$, then

$$\sum_{d|n, d\neq n} d \cdot M(d) \leq \sum_{d|n, d\neq n} d \cdot p^d \leq p^{n/2} \sum_{d=1}^{n/2} d < p^n.$$

Hence, we can argue that $M(n) > 0$, and that their exists a degree n irreducible polynomial. And hence $\mathrm{GF}(p^n)$ exists.

A more elegant description of $M(n) > 0$ is via the Möbius transform:

Theorem 4.50

Let $\mu(d) \in \{+1, -1\}$ be the Möbius function, we have

$$M(n) = \frac{1}{n} \sum_{d|n} \mu(d) \cdot p^{n/d}, \tag{4.37}$$

$$p^n = \sum_{d|n} d \cdot M(d). \tag{4.38}$$

For another application of the Möbius transform, one can refer to Appendix A.1.2.

Remark 4.51

From the Theorem 4.50 above, we see that

$$M(n) \sim \frac{1}{n}p^n \sim \frac{\#\text{ of polynomial up to degree } n}{\ln(\text{numerator})}. \tag{4.39}$$

This is exactly the form of the *prime number theorem* over function fields!

The prime number theorem over the field $\mathbb{N}$ states the following: let $\pi(n)$ be the number of primes up to n, then

$$\pi(n) \sim \frac{N}{\ln N}. \tag{4.40}$$

In fact, the prime number theorem holds for numbers, extension fields, function fields, and even permutations!

Remark 4.52

For all possible p and d, we have shown that there exists an irreducible polynomial, however, which one is the best? I.e., how can we fix a representation that is "good" enough?

The *Conway polynomials* are the first such representation, standardizing the choosing of the irreducible polynomials via lexicographic ordering.

4.5. Reed–Solomon Codes over $\mathrm{GF}(p^n)$

Let us upgrade from the simpler Reed–Solomon code introduced back in Section 4.3.

Definition 4.53

Given a prime number p and an integer n. Choose an integer k and another integer $N \leq p^n$, we can select distinct *evaluation points* $a_1, a_2, \ldots, a_N \in \mathrm{GF}(p^n)$. Then, the Reed–Solomon code is defined as

$$\mathrm{RS}(a_{1:N}, k) := \left\{ \big(f(a_1), f(a_2), \ldots, f(a_N)\big) \;\middle|\; f \in \mathrm{GF}(p^n)[x]^{<k} \right\} \tag{4.41}$$

One often writes q in place of p^n.

Remark 4.54

The Reed–Solomon code is a really widely used code, appearing in our daily lives. For example the case where $p = 2$ and $n = 8$ is used in AES and QR codes. However, the polynomials used in the representation of AES are not the Conway polynomials, hence AES is often criticized for this reduce in efficiency.

Theorem 4.55

$\mathrm{RS}(a_{1:N}, k)$ is an $[N, k, N - k + 1]$-code over $\mathrm{GF}(q)$.

The proof is same as that of Theorem 4.30 by using an argument on the degrees of freedom.

Proof. An equivalent statement is that for all polynomials $f, g \in \mathrm{GF}(q)[x]^{<k}$, $d = n - k + 1$ of the evaluation points, say a_{i_1} to a_{i_d}, satisfy

$$f(a_{i_j}) \neq g(a_{i_j}) \qquad \forall\, j = 1 \sim d.$$

If the minimum distance of the Reed-Solomon code is less than d, say it is $N - k$, then any two polynomials should differ at $n - k$ points, and coincide at k points. Without loss of generality, let f and g coincide at $a_1, a_2, \ldots, a_k$, then

$$\underbrace{(x - a_1)(x - a_2) \cdots (x - a_k)}_{\text{degree } k} \mid \underbrace{f - g}_{\text{degree } <k}.$$

The only possibility is if $f - g = 0$, which is a contradiction. Hence it is proven. ■

Lemma 4.56: Singleton Bound

All $[N, k]$ linear codes have a minimum distance $\leq N - k + 1$.

Thus, we can see that Reed–Solomon codes are optimal in terms of their minimum distances by using the Singleton bound!

Next, we can also define the general Reed–Solomon code over $\mathrm{GF}(q)$ as

Definition 4.57

Continue on the setting of a Reed–Solomon code. Select $c_1, c_2, \ldots, c_N \in \mathrm{GF}(q)$ that need not be distinct. The generalized Reed–Solomon code is defined as

$$\mathrm{GRS}(c_{1:N}, a_{1:N}, k) := \left\{ \big(c_1 f(a_1), c_2 f(a_2), \ldots, c_N f(a_N)\big) \,\middle|\, f \in \mathrm{GF}(q)[x]^{<k} \right\}. \tag{4.42}$$

When using the generalized Reed–Solomon code, one simply divides the received codeword by c_i's to obtain the Reed–Solomon code. The only interesting thing and reason that we introduce it is that the dual of a Reed–Solomon code is $\mathrm{RS}^{\perp} = \mathrm{GRS}$.

Example: A Reed–Solomon code is called a *full Reed–Solomon code* when the length $N = q$.

Theorem 4.58: Dual of a Full Reed–Solomon Code

Pick any dimension k and form the full Reed–Solomon code $\mathcal{C}$. Them, $\mathcal{C}^{\perp}$ is also a full Reed–Solomon code of dimension $q - k$.

Proof. Consider any polynomial $f \in \mathrm{GF}(q)[x]^{<k}$ and $g \in \mathrm{GF}(q)[x]^{<q-k}$. Then

$$\big(f(a_1), \ldots, f(a_q)\big) \cdot \big(g(a_1), \ldots, g(a_q)\big) = \sum_{i=1}^{q} (\underbrace{f \cdot g}_{h})(a_i).$$

For any h of degree less than q, we have by symmetry

$$\sum_{i=1}^{q} h(a_i) = 0.$$

Hence we see that f and g are always orthogonal, and the theorem is proven. ■

The dual to a full Reed–Solomon code is really simple. □

Example: For non-full Reed–Solomon codes, we can also have a simple description of its dual by viewing them as punctured full Reed–Solomon codes. This proof is different from the one given in Theorem 4.32. By the following diagram:

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\mathrm{Pun}_N} & \mathrm{Pun}_N(\mathcal{C}) \\ {\scriptstyle\perp}\downarrow & & \downarrow{\scriptstyle\perp} \\ \mathcal{C}^\perp & \xrightarrow{\mathrm{Sho}_N} & \mathrm{Sho}_N(\mathcal{C}^\perp) \end{array}$$

We have, for the code

$$\mathrm{Pun}_N(\mathcal{C}) = \left\{ \big(f(a_1), \dots, f(a_{N-1})\big) \,\middle|\, f \in \mathrm{GF}(q)[x]^{<k} \right\},$$

its dual code being

$$\begin{aligned} \mathrm{Sho}_N(\mathcal{C}^\perp) &= \mathrm{Sho}_N \left\{ \big(g(a_1), \dots, g(a_{N-1}), g(a_N)\big) \,\middle|\, g \in \mathrm{GF}(q)[x]^{<N-k} \right\} \\ &= \left\{ \big(g(a_1), \dots, g(a_{N-1})\big) \,\middle|\, g \in \mathrm{GF}(q)[x]^{<N-k}, g(a_N) = 0 \right\} \\ &= \left\{ \big(g(a_1), \dots, g(a_{N-1})\big) \,\middle|\, g(x) = h(x)\cdot(x - a_N), h \in \mathrm{GF}(q)[x]^{<N-k-1} \right\} \\ &= \left\{ \big((a_1 - a_N)h(a_1), \dots, (a_{N-1} - a_N)h(a_{N-1})\big) \,\middle|\, h \in \mathrm{GF}(q)[x]^{<N-k-1} \right\}. \end{aligned}$$

By continuing puncturing the code, we will have more Vandermonde-like terms in the front, and finally obtaining the dual code to a general Reed–Solomon code as in Theorem 4.32. □

4.6. Decoding Reed–Solomon Code

29 Apr.

Though the Reed–Solomon is a very strong code, it is only used in QR codes and cloud storage (secretly). As we will discuss in the next chapter, modern day wireless communication utilizes polar codes or LDPC codes. Why the difference?

For wireless communication, agents communicate via sending electromagnetic waves. One of the oldest method to encode information into the wave is by sequentially turning on and off the wave to generate pulses which the decoder can decode. A modern dat method is to encode the information into the amplitude and phase of the wave:

$$A\mathrm{e}^{i\theta}\mathrm{e}^{i2\pi f t}.$$

Each codeword will be a different complex number $A\mathrm{e}^{i\theta}$ on the complex plane, forming a *constellation set.*

For cloud storage, a data center contains lots of hard disk drives containing information. Sometimes a disk fails, but we can fix it by employing a large alphabet. For example, we can let an entire 1TB hard disk be a single codeword, then we can have a Reed–Solomon code over $\mathrm{GF}(2^{2^{12}})$, this becomes really fault-tolerant.

4.6.1. Decoding over BEC

Given a string with erasure:

$$(y_1, \dots, y_k, \underbrace{\mathcal{E}, \dots, \mathcal{E}}_{\times(N-k)}) \in \mathrm{GF}(q)^N,$$

we can recover it by finding the polynomial $f(x) = \gamma_0 + \gamma_1 x + \cdots + \gamma_{k-1}x^{k-1}$ by solving the system

$$[\gamma_0, \ldots, \gamma_{k-1}] \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_k \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{k-1} & a_2^{k-1} & \cdots & a_k^{k-1} \end{bmatrix} = [y_1, \ldots, y_{k-1}].$$

This system is solvable since a_i's are distinct and the determinant of the matrix above is non-zero by the well-known fact over Vandermonde determinant. In practice, the polynomial f can be found in a much faster fashion via interpolation: either by Newton's, Lagrange's, or other interpolation schemes.

4.6.2. Decoding over BSC

Consider given a string $y = (y_1, y_2, \ldots, y_N)$ where it is guaranteed that some of the y_i's are wrong, and that there are less than $d/2 = (N - k + 1)/2$ errors, then what can we do to uniquely decode?

As a first approach, we can simply compute $d_{\mathrm{H}}(w, y)$ for all codeword w in $\mathcal{C}$, and see which one has the minimum distance with y. Since the amount of error is strictly less than $d/2$, we know that the minimizer w must be unique. Good as this method may be, it is way too costly.

Another method is to find, again, a low complexity polynomial f that passes through most of the points (a_i, y_i), where a_i is the ith evaluation point. As a tool to aid our evaluation, let us define the following polynomial:

Definition 4.59: Error Locating Polynomial

The error locating polynomial $\ell(t)$ satisfies $\ell(a_i) = 0$ if and only if y_i is an error ($y_i \neq f(a_i)$).

For example, if the error locates only at a_1 and a_2, then $\ell(x) = (x - a_1)(x - a_2)$. For any a_i, we have either $\ell(a_i) = 0$ or $f(a_i) = y_i$ holding true. Then,

$$\ell(a_i) \cdot y_i = \ell(a_i) \cdot f(a_i) \tag{4.43}$$

must hold true for all evaluation points. However, this equation cannot be directly used to compute f. Instead, we have the following decoding algorithm:

Welch–Berlekamp algorithm: Let us define the polynomial $g := f\ell$, then

$$y_i \ell(a_i) = g(a_i) \tag{4.44}$$

can be solved with linear algebra. By representing g as a degree q polynomial and ℓ as a degree e polynomial,

$$\Rightarrow y_i \begin{bmatrix} \ell_0, \ell_1, \ldots, \ell_e \end{bmatrix} \begin{bmatrix} 1 \\ a_i \\ \vdots \\ a_i^e \end{bmatrix} = \begin{bmatrix} g_o, g_1, \ldots, g_q \end{bmatrix} \begin{bmatrix} 1 \\ a_i \\ \vdots \\ a_i^q \end{bmatrix} \qquad (\forall\, i \in \{1, 2, \ldots, N\})$$

$$\Rightarrow \begin{bmatrix} \ell_0, \ldots, \ell_e, g_0, \ldots, g_q \end{bmatrix} \begin{bmatrix} a_1^0 y_1 & a_2^0 y_2 & \cdots & a_N^0 y_N \\ \vdots & \vdots & & \vdots \\ a_1^e y_1 & a_2^e y_2 & \cdots & a_N^e y_N \\ -a_1^0 & -a_2^0 & \cdots & -a_N^0 \\ \vdots & \vdots & & \vdots \\ -a_1^q & -a_2^q & \cdots & -a_N^q \end{bmatrix} = 0.$$

An additional constraint of $\ell_e = 1$ is often included, we then have

$$\begin{bmatrix}\ell_0, \dots, \ell_{e-1}, g_0, \dots, g_q\end{bmatrix} \begin{bmatrix} a_1^0 y_1 & a_2^0 y_2 & \cdots & a_N^0 y_N \\ \vdots & \vdots & & \vdots \\ a_1^{e-1} y_1 & a_2^{e-1} y_2 & \cdots & a_N^{e-1} y_N \\ -a_1^0 & -a_2^0 & \cdots & -a_N^0 \\ \vdots & \vdots & & \vdots \\ -a_1^q & -a_2^q & \cdots & -a_N^q \end{bmatrix} = \left[a_1^e y_1, a_2^e y_2, \dots, a_N^e y_N\right].$$

For the system to be uniquely solvable, we require $q = N - e - 1$.

At the start, we initialize e with the largest amount of errors allowed, which is $e = \lfloor (d-1)/2 \rfloor = \lfloor (N-k)/2 \rfloor$. If the system is solvable, then we can obtain $g(x)$ and $\ell(x)$, and hence also $f(x)$. If the system is not solvable due to redundancy, then we set e to $e - 1$ and repeat the process all over again.

4.6.3. List Decoding

The task is the same: Given a word $y = (y_1, \dots, y_N)$, we want a decoding scheme that uniquely determines w that minimizes $d_{\mathrm{H}}(w, y)$. The method of *list decoding* provides a list of possible candidates. By providing a list of candidates, we can actually decode beyond the $d/2$ bound.

Let us rephrase the Welch–Berlekamp algorithm finding the codeword-generating polynomial f:

(a) Define $Q(x, y) := y \cdot \ell(x) - g(x)$, with $Q(a_i, b_i) = 0$.

(b) Find polynomial f such that $y - f(x) \mid Q(x, y)$.

(c) The function Q thus has $\deg_y Q = 1$.

Now we consider Q with higher y-degrees.

(a) Find a general Q such that $Q(a_i, b_i) = 0$ for all i.

(b) Factorize Q to find all possible f subject to $y - f(x) \mid Q(x, y)$.

(c) Add every such f to the list of candidates L.

We thus have a list of size equal to $\deg_y Q$.

Example: For instance, if we want the list size to be 2, we can have a general Q be of the form

$$\begin{aligned} Q(x, y) = y^0 &\left(Q_{00} + Q_{10}x + \cdots + Q_{\frac{N}{3}0}x^{N/3}\right) \\ &+ y^1 \left(Q_{01} + Q_{11}x + \cdots + Q_{\frac{N}{3}1}x^{N/3}\right) + y^2 \left(Q_{02} + Q_{12}x + \cdots + Q_{\frac{N}{3}2}x^{N/3}\right). \end{aligned}$$

There are a total of N variables Q_{ij} to be determined, hence it is solvable. □

4.7. Application: Group Testing

Here we digress for a moment and discuss the interesting topic of group testing. In the end, we will see how, amazingly, a Reed–Solomon decoder can aid us in this task.

Let us consider a usual scenario of group testing: Some people are infected by a disease and some are not, given testing kits with 100% accuracy (noiseless), how can we minimize the amount of testing kits

used to determine exactly who are infected? Let the number of people be n, the amount of infected people be k, and the amount of test kits used be m.

A naïve approach is to use exactly $m = N$ test kits. But we can do much better than this by using shared tests!

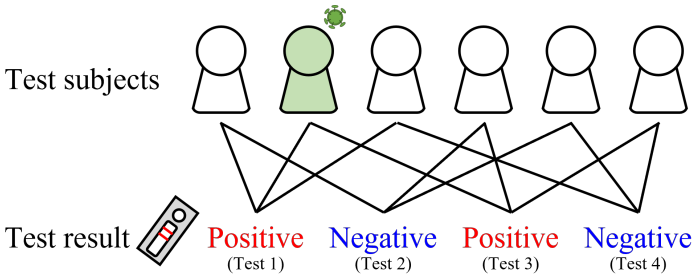


Figure 4.1. Group testing. The connected lines show the test subjects sharing a test. The result of a test is negative only if all the people sharing the test is negative (doesn't carry the disease).

See the bipartite graph of Figure 4.1, we only need $m = 4$ tests to determine the infected ones out of $n = 6$ people. If a person appears in a negative test, then that person must be negative. The remaining people that cannot be determined are suspicious, and require further testing.

Let us generalize the analysis. Let us use a matrix $A \in \{0,1\}^{m\times n}$ to represent the test schedule, with

$$A_{ij} : \text{the } i\text{th test contains the } j\text{th person.} \tag{4.45}$$

Then we have the result of the ith test as

$$y_i = \bigvee_{j=1}^{n} (A_{ij} \wedge x_j), \tag{4.46}$$

where $x_j \in \{0,1\}$ is 1 if and only if the jth person is infected. The symbol $\vee$ and $\wedge$ represents the logical (boolean) OR and AND operation. As an abuse of notation, one often rewrites the system of equations into the form of $y = Ax$. However, this is by no means a linear algebra problem, it is instead a logic problem.

How do we solve this system of equations to determine x?

Example: Consider $n = 2^5$ and $k = 1$, the optimal test is by separating the test subjects into two sets. Consider

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & \cdots & 1 \end{bmatrix}$$

with the jth column of A being the binary representation of the number j. Then if the jth person is infected, i.e. $x_k = \delta_{jk}$, we directly obtain the binary representation of j as $y = Ax$. We only need $m = 5$ tests to find the $k = 1$ infected person in $n = 2^5$ people. □

Example: Consider this time $k = 2$. For the jth column of A, let it represent j in its q-ary representation ($q = p^n$), for example, (a_0, a_1, a_2). Then we can map j to a polynomial $f(x) = a_0 + a_1x + a_2x^2 \in \mathbb{F}_q[x]$. We then evaluate f at $0, 1, \ldots, q-1$ and obtain the values $(b_0, b_1, \ldots, b_{q-1}) \in \mathbb{F}_q^{q-1}$. We then further turn each $b_i \in \mathbb{F}_q$ into its binary representation, this large vector of 0's and 1's will be the jth column of A. This was introduced in the work by Kautz and Singleton [KS64]. In fact, the decoding can reduce to a Reed–Solomon codes. For more details, one should refer to Section 6.3.5. □

For more discussions on group testing, please refer to Section 6.2.

5. Low Density Parity Check Code

22 Apr.

Coding theory is very much related to the communication protocols we use. In 3G networks, turbo code was used; in 4G, LDPC code was used; and now as of 2025, in 5G, we use a combination of LDPC code and polar code. It would be interesting to see which will appear in our future use of 6G.

Though, as we have seen, polar code is capacity-achieving and it has good second order scaling properties, it is only used in a quarter of the scenarios in 5G; LDPC code, on the other hand, is not capacity-achieving! However, for the block lengths that we cared about, LDPC code is actually better, but it is more of an engineering problem than a theoretical one.

In this chapter, we will dive into the theory and analysis of LDPC code, and see what it has to offer. For more details, one should refer to chapter 3 of "Modern Coding Theory" [RU07].

5.1. LDPC

To define a low-density parity-check (LDPC) code, we draw a bipartite graph:

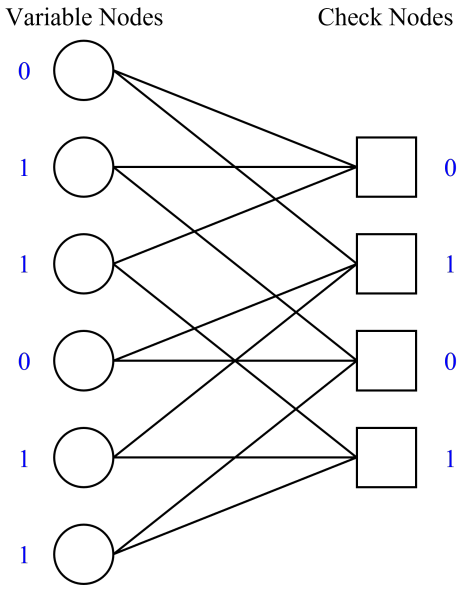


Figure 5.1. Tanner graph of an LDPC code.

The graph consists of two types of nodes: variable nodes (VN) which carry information, and check nodes (CN) who's check sums determine the symptom of a received word. A valid codeword is equal to a vector of assignments on the variable nodes such that all values to the check nodes are zero, else the word is corrupted.

To draw a Tanner graph, it is required that the VN's are drawn on the left, indicated by circular nodes, and the CN's are drawn on the right, indicated by square nodes. Besides Figure 5.1, the only allowed variant of drawing a Tanner graph is Figure 5.2.

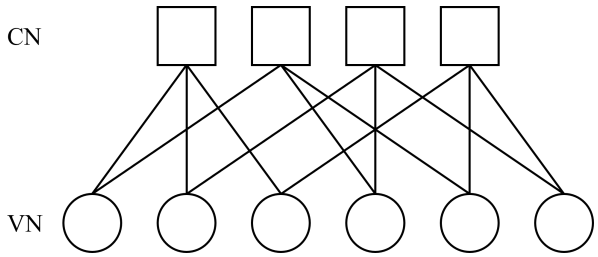


Figure 5.2. Horizontal Tanner graph.

Definition 5.1: Parity-Check Matrix Formulation

An LDPC code can be uniquely defined by the *biadjacency matrix* $H \in \mathbb{F}_2^{|\mathrm{VN}|\times|\mathrm{CN}|}$ of the bipartite graph, where VN is the set of variable notes, and CN is the set of check nodes. The graph is also called a *Tanner graph*.

The biadjacency matrix is related to the *adjacency matrix* of the Tanner graph by

$$A = \begin{bmatrix} 0 & H \\ H^{\mathsf{T}} & 0 \end{bmatrix}, \tag{5.1}$$

where the first $|\mathrm{VN}|$ rows and columns are the VN's, and the latter $|\mathrm{CN}|$ rows and columns are the CN's.

Example: Take Figure 5.1 as an example. Listing the ten vertices in order from VN to CN, from top to bottom, we have the following biadjacency matrix:

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Given a word $w \in \mathbb{F}_2^{|\mathrm{VN}|\times 1}$, one can check whether it is a valid codeword by seeing if Hw is the zero vector or not. □

Theorem 5.2: Rate of LDPC Code

The rate R of an LDPC code satisfies

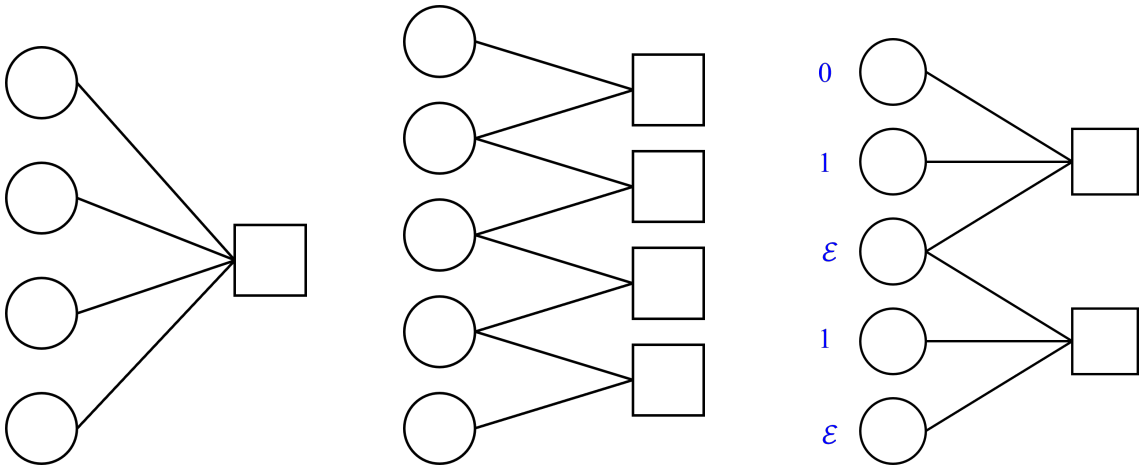
$$R \geq 1 - \frac{|\mathrm{CN}|}{|\mathrm{VN}|}, \tag{5.2}$$

where the equality is met if and only if H is full rank.

5.2. LDPC over BEC

Below we see a couple of examples of LDPC over BEC,

Example: Figure 5.3a shows the Tanner graph of a single parity-check (SPC) code. The biadjacency matrix of the code is the matrix of ones $H = [1, 1, 1, 1]^{\mathsf{T}}$. If one receives a codeword with a single



(a) Single parity-check code. (b) Repetition code. (c) Onion-peeling decoding.

Figure 5.3. LDPC over BEC

erasure, say $[0, 1, 0, \mathcal{E}]^\mathsf{T}$, it can be readily seen from the check sum that the erasure should be a 1. However, if there are more than one erasures, say $[0, \mathcal{E}, 0, \mathcal{E}]^\mathsf{T}$, nothing can be learned. Hence, we see that SPC can deal with code having only one erasure. □

Example: Figure 5.3b shows the Tanner graph of a repetition code (RC). Its biadjacency matrix is a *Toeplitz matrix* of the form

$$H = \begin{bmatrix} 1 & & & & \\ 1 & 1 & & & \\ & 1 & 1 & & \\ & & 1 & 1 & \\ & & & 1 & 1 \end{bmatrix},$$

and the generator matrix of the codebook is $G = [1, 1, 1, 1, 1]$. □

Example: In Figure 5.3c, we consider decoding with a not-so-trivial LDPC code over a BEC channel. Upon receiving $[0, 1, \mathcal{E}, 1, \mathcal{E}]$, the first CN can immediately recognize that the first erasure is 1. After finding out the first erasure, the second erasure can be determined to be 0 from the second CN.

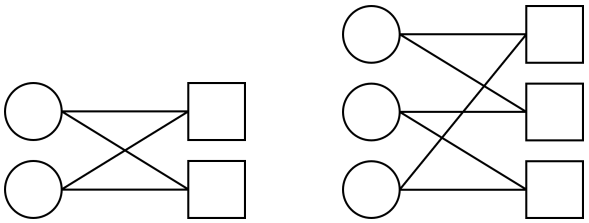
This decoding method is called the "*onion-peeling decoder*." Other names exist, too, including the iterative decoder, and my favorite – the "sudoku decoder." □

Definition 5.3: Stopping Set / Trapping Set

A subset of VN that cannot be decoded if they are all erasure is called a stopping set.

A stopping set can be removed by adding a new check node, connecting to variable nodes both in the stopping set and outside of the stopping set. However, this decreases the code rate.

Example: A common example of stopping sets are cycles. Figure 5.4 are some examples.



(a) 4-cycle. (b) 6-cycle.

Figure 5.4. Cycles as trapping sets.

Let the block length of the code be N, which is often in the orders of thousands, finding a 4-cycle requires traversing the graph by brute force, requiring a complexity of $O(N^4)$. Similarly, for 6-cycles, $O(N^6)$ computations are needed. The complexity quickly go out of control as the size of the cycle one wish to check increases. □

5.2.1. Decoding Analysis

To see whether a setup of VN's and CN's can be decoded, we can analyze the performance on a random graph instead.

Definition 5.4: Regular Graph

A graph is regular if $\deg v$ is a constant for all vertex v, where $\deg v$ is the amount of edges the vertex is connected to.

Definition 5.5: Regular LDPC

An LDPC code is regular if the degree of the VN's (variable degree) and the degree of the CN's (check degree) are constants.

Example: A $[3, 6]$-LDPC is one where its variable degree is 3, and its check degree is 6. □

Here we will be dealing with regular LDPC codes, especially having the degree of VN's be 3 and the degree of CN's be 6 such that the rate is $R = 1/2$.

Pick a random graph that satisfies the above setup and send a codeword over $\mathrm{BEC}(x)$. Initially, the fraction of VN's that remain as erasures is $x_0 := x$.

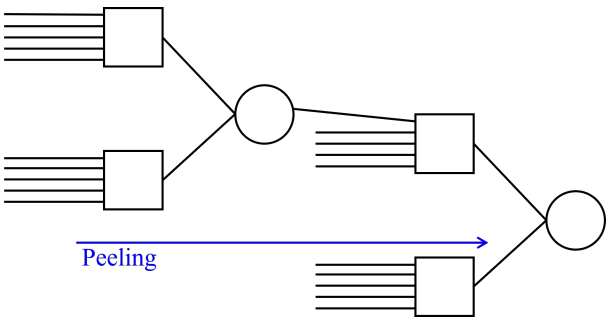


Figure 5.5. Tree of onion-peeling / sudoku decoding.

After a layer of onion-peeling decoding, a VN that was originally an erasure (say the first VN drawn in

Figure 5.5) can be decoded if at least one of the check nodes on its left connects to no other erasures. I.e., the remaining fraction of erasures after an iteration becomes

$$x_1 := x\left(1-(1-x)^5\right)^2$$

on expectation. If we peel again, the remaining fraction of erasures become

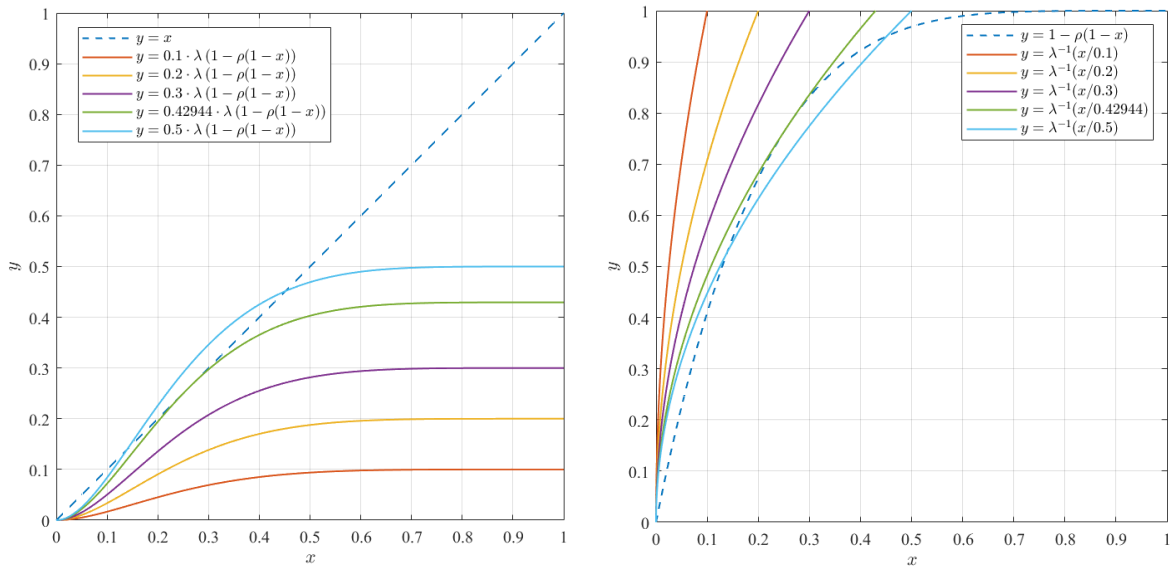
$$x_2 := x\left(1-(1-x_1)^5\right)^2.$$

And we do this again and again until achieving $x_n \to 0$, then we "believe" that this code is good enough.

Though the iteration is strictly decreasing, it does not always converge to zero. See Figure 5.6 below. Let us consider the code having a rate $R = 1/2$, what parameters can the x_0 in $\mathrm{BEC}(x_0)$ be so that codewords can be successfully decoded? In theory, the optimal case is $x_0 = 1/2$. However, by drawing out the iteration

$$x_n = x_0\left(1-(1-x_{n-1})^5\right)^2 \tag{5.3}$$

via Figure 5.6a, we can see that only when $x_0 \lesssim 0.42944$ will the sequence converge to zero. Hence, LDPC code is NOT capacity-achieving.



(a) (Part of a) Cobweb plot.

(b) EXIT chart.

Figure 5.6. Convergence of fraction of erasures through decoding.

Another often seen diagram of showing the convergence of the fraction of erasures is the EXIT graph, as seen in Figure 5.6b. It performs a transformation on the two curves $y = x$ and $y = x_0(1-(1-x)^5)^2$ to make the analysis for more complicated Tanner graphs simpler, as we will see in the next section.

5.3. Irregular LDPC

In this section we discuss a general LDPC code, containing nodes of various degrees not constrained to be constants, called irregular LDPC.

Definition 5.6: Irregular LDPC

For an irregular LDPC, various terms can be defined:

- From the perspective of nodes:

$$\Lambda_i := \#\text{ of VN with degree } i, \tag{5.4}$$
$$P_i := \#\text{ of CN with degree } i, \tag{5.5}$$

The generating function of the two enumerators are denoted as

$$\Lambda(x) := \sum_i \Lambda_i x^i, \tag{5.6}$$
$$P(x) := \sum_i P_i x^i. \tag{5.7}$$

Λ and P are the *variable* and *check degree distributions from a node perspective*. The normalized degree distributions are

$$L_i := \frac{\#\text{ of VN of degree } i}{\#\text{VN}} = \frac{\Lambda_i}{\Lambda(1)}, \qquad L(x) := \sum_i L_i x^i = \frac{\Lambda(x)}{\Lambda(1)}, \tag{5.8}$$
$$R_i := \frac{\#\text{ of CN of degree } i}{\#\text{CN}} = \frac{P_i}{P(1)}, \qquad R(x) := \sum_i R_i x^i = \frac{P(x)}{P(1)}, \tag{5.9}$$

the letter L and R stands for left and right, respectively.

- From the perspective of edges:

$$\lambda(x) = \sum_i \lambda_i x^{i-1} := \frac{\Lambda'(x)}{\Lambda'(1)} = \frac{L'(x)}{L'(1)}, \tag{5.10}$$
$$\rho(x) = \sum_i \rho x^{i-1} := \frac{P'(x)}{P'(1)} = \frac{R'(x)}{R'(1)}. \tag{5.11}$$

The coefficient $\lambda_i \propto i\Lambda_i$ (resp. $\rho_i \propto iP_i$) equals to the fractions of edges that connect to VN's (resp. CN's) of degree i. We call λ and ρ are the *variable* and *check degree distributions from an edge perspective*.

Example: For a regular LDPC, say the $[3, 6]$-LDPC, we have $L(x) = x^3$ and $R(x) = x^6$, with $\lambda(x) = x^2$ and $\rho(x) = x^5$. Note that the iteration Equation (5.3) can be rewritten as

$$x_n = x_0 \cdot \lambda\left(1 - \rho(1 - x)\right). \tag{5.12}$$

This result can be further generalized. □

Consider $L(x) = \frac{1}{2}x^3 + \frac{1}{2}x^4$ and $R(x) = \frac{2}{3}x^6 + \frac{1}{3}x^{20}$. We have $\lambda(x) = \frac{3}{7}x^2 + \frac{4}{7}x^3$ and $\rho(x) = \frac{3}{8}x^5 + \frac{5}{8}x^{19}$. Its Tanner graph can be roughly drawn as in Figure 5.7, it is a random ensemble of LDPC codes.

When applying onion-peeling decoding (refer to Figure 5.5), a VN (say VN_1) connects to CN's with 6 edges with probability ρ_6, and CN's with 20 edges with probability ρ_{20}. Of these CN's with 6 or 20 edges, only 5 or 19 edges are connected to the other VN's. If VN_1 can be decoded successfully, it requires that at least one of the CN's that it is connected to connects to no other erasures. Suppose the

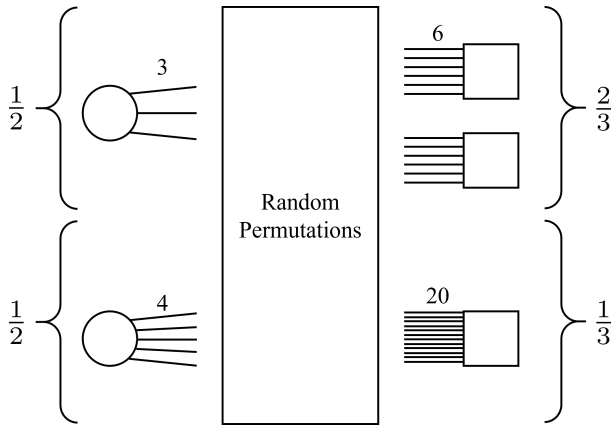


Figure 5.7. Irregular LDPC with $L(x) = \frac{1}{2}x^3 + \frac{1}{2}x^4$ and $R(x) = \frac{2}{3}x^6 + \frac{1}{3}x^{20}$.

fraction of erasures is x, the probability on average for a single CN to connect to no erasure is

$$\frac{3}{8}(1-x)^5 + \frac{5}{8}(1-x)^{19} = \rho_6(1-x)^5 + \rho_{20}(1-x)^{19} = \rho(1-x).$$

And for each CN, it connects to a degree 3 VN_1 with probability λ_3, and to a degree 4 VN_1 with probability λ_4. Since one extra edge of VN_1 should be used to help the future decoding, if VN_1 has degree i, we only need one of the remaining $i-1$ check nodes it connected to be helpful. The probability on average that none of the CN's it is connected to is helpful in its decoding will be

$$\frac{3}{7}(1-\rho(1-x))^2 + \frac{4}{7}(1-\rho(1-x))^3 = \lambda_3(1-\rho(1-x))^2 + \lambda_4(1-\rho(1-x))^3 = \lambda(1-\rho(1-x)).$$

Henceforth, we see:

Theorem 5.7: Density Evolution

The evolution of the density of erasures of an LDPC over $\mathrm{BEC}(x_0)$ after n iterations of decoding follows the following iteration:

$$x_n = x_0 \cdot \lambda(1-\rho(1-x_{n-1})). \tag{5.13}$$

The iteration between the functions

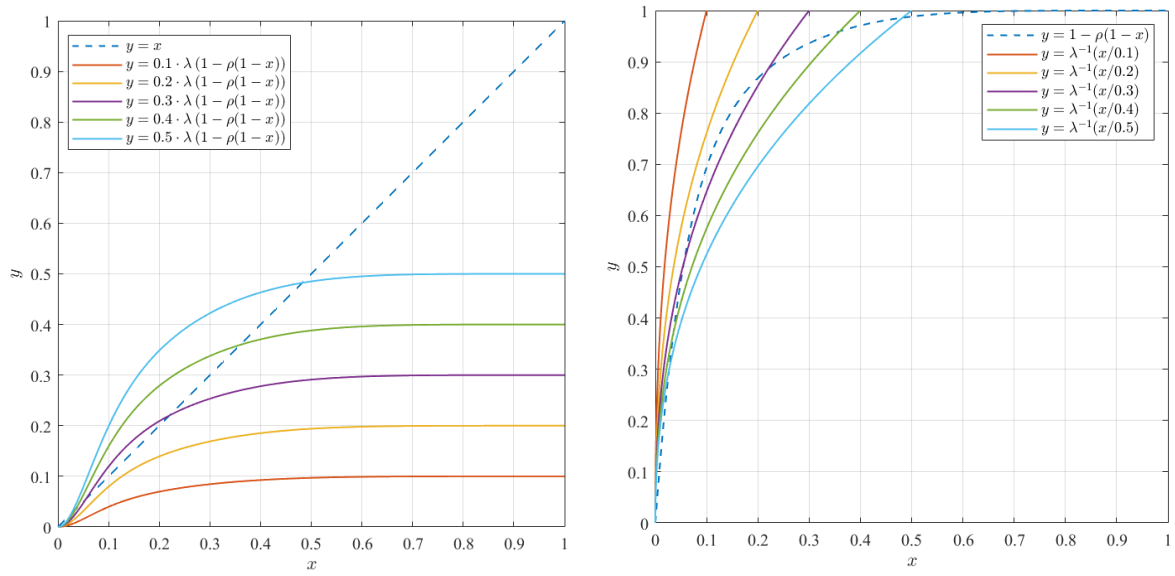
$$\begin{aligned} y_n &= x_0 \cdot \lambda(1-\rho(1-x_{n-1})), \\ x_n &= y_n \end{aligned}$$

can be translated the iteration between the functions

$$\begin{aligned} y_n &= 1-\rho(1-x_{n-1}), \\ x_n &= x_0\lambda(y_n). \end{aligned}$$

We can again draw the cobweb plot and EXIT chart as Figure 5.8 below.

The design of a suitable code will hence be translated to the design of suitable polynomials ρ and λ.



(a) Cobweb plot.

(b) EXIT chart.

Figure 5.8. Convergence of fraction of erasures for $\lambda(x) = \frac{3}{7}x^2 + \frac{4}{7}x^3$ and $\rho(x) = \frac{3}{8}x^5 + \frac{5}{8}x^{19}$.

5.4. LDPC over BSC

29. Apr

Previously, we have studied how to decode an LDPC code over BEC. It is really a simple task, with the erasure density evolving acoording to Theorem 5.7. However, if the decoding is over BSC, this task becomes much more difficult.

Consider the following figure, where we again consider the decoding of a $[3, 6]$-LDPC.

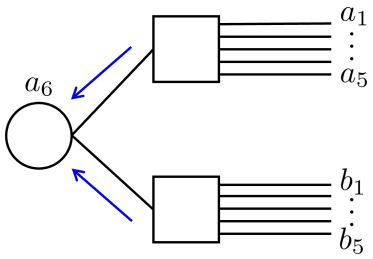


Figure 5.9. Belief propagation over BSC.

If we propagate the value of a_6 directly from a_1 to a_5, then

$$a_6 = a_1 + \cdots a_5,$$

where the addition is over $\mathbb{F}_2$. However, there might be conflict with the lower branch of

$$a_6 = b_1 + \cdots + b_5.$$

One way to solve this conflict is by majority voting, where a variable node updates its value based on the suggestion that is the majority. However, this doesn't work in our case since both options are equally feasible.

A better method will be to utilize a *soft decoder*.

5.4.1. Gallager's Algorithm

Instead of a strong decoder forcing a node to be an exact value causing major conflicts, we can use a soft decoder that *suggests* the value a variable node should be.

For the following discussion, please also refer to Section 3.2.3, especially Theorem 3.17 and Equation (3.38). Each variable node, for example VN_i now carries the log-likelihood ratio (LLR) of value

$$\mathrm{LLR}_i = \ln \frac{\Pr\{\mathrm{VN}_i = 0\}}{\Pr\{\mathrm{VN}_i = 1\}}. \tag{5.14}$$

The LLR can also be transformed into the signed total variational distance (TV) via

$$\mathrm{TV}_i = \tanh\left(\frac{\mathrm{LLR}_i}{2}\right). \tag{5.15}$$

When the information are sent from $a_1, a_2, \ldots, a_5$ to a_6, the combined information is

$$\mathrm{TV}_{a_6,a_{1:5}} = \prod_{i=1}^{5} \mathrm{TV}_{a_i} = \tanh\left(\frac{\mathrm{LLR}_{a_6,a_{1:5}}}{2}\right). \tag{5.16}$$

However, information from the branch of $b_1, b_2, \ldots, b_5$ should also be considered. Combining with the original LLR that a_6 has, its updated LLR will be

$$\mathrm{LLR}_{a_6,\mathrm{new}} = \mathrm{LLR}_{a_6,\mathrm{old}} + \mathrm{LLR}_{a_6,a_{1:5}} + \mathrm{LLR}_{a_6,b_{1:5}}. \tag{5.17}$$

The propagation and update of the probability of a VN being 0 or 1 is termed the belief propagation.

However, remember that the use of TV's and LLR's are just a matter of representation, and are only used since they are easy to combine under channel serial and parallel combination, respectively. For the example above, a_6 is originally a BSC. But after a single iteration of belief propagation, the a_6 channel updates to

$$a_6 \circledast (a_1 \boxbslash \cdots \boxbslash a_5) \circledast (b_1 \boxbslash \cdots \boxbslash b_5). \tag{5.18}$$

For a general LDPC over general channels, we have

Theorem 5.8: Density Evolution

Given an LDPC code with degree distributions from an edge perspective $\lambda(x)$ and $\rho(x)$. If the channel one communicates over with is $x_0 = \mathcal{E}$, then through decoding, the equivalent channel updates according to

$$x_{n+1} = \mathcal{E} \circledast \lambda^{\circledast}\left(\rho^{\boxbslash}(x_n)\right). \tag{5.19}$$

The function $\lambda^{\circledast}$ is a polynomial with multiplication as parallel combination (equiv. to addition in the LLR representation); the function $\rho^{\boxbslash}$ is a polynomial with multiplication as serial combination (equiv. to multiplication in the TV representation).

The step $\rho^{\boxbslash}(x_n)$ worsens the channel, but the step $\mathcal{E} \circledast \lambda^{\circledast}(\ldots)$ improves the channel. As the iteration goes on, we would like to expect that $x_n \to \mathrm{BSC}(0)$ (the distribution over LLR goes to $\pm\infty$), then the LDPC code can be decoded as the quality of the bits becomes better and better, and the words converges to the correct codeword. If the distribution over LLR stays around zero, however, then we cannot decode the LDPC code. We can see whether decoding is possible or not given λ and ρ by plotting the EXIT chart.

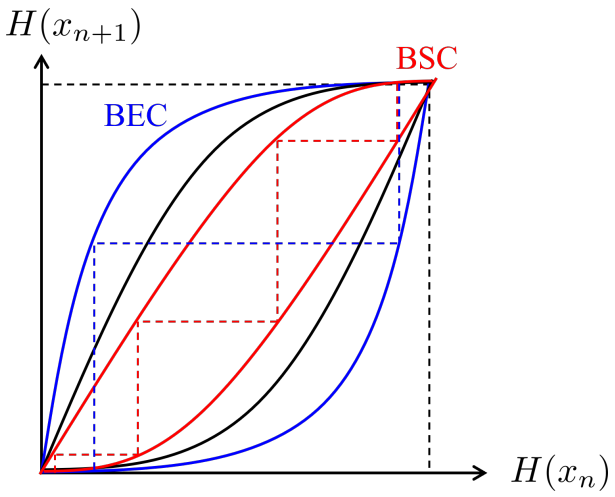


Figure 5.10. Illustration of the EXIT chart of LDPC code over general channels. The blue curves represent BEC, with the best performance; the red curves represent BSC, with the worst performance; the black curves represent any other channel, with intermediate performance.

See Figure 5.10, we use the conditional entropy as an index of the channel quality. The two curves drawn is $(H(x_n), H(y_{n+1})$ and $(H(x_{n+1}), H(y_{n+1}))$, where

$$\begin{aligned} y_{n+1} &= \rho^{\boxplus}(x_n), \\ x_{n+1} &= \mathcal{E} \circledast \lambda^{\circledast}(y_{n+1}). \end{aligned} \tag{5.20}$$

One can see that BEC converges faster to $(0, 0)$ in comparison to BSC, and any other channel has performances in between the former two. Moreover, since the parallel and serial combination of both BEC and BSC remain as BEC and BSC, respectively, we can actually trace the dotted lines on the EXIT chart to see how the entropy evolves as one iterates the decoding scheme. This is not the case for other channels, however. For example, though the parallel combination of AWGN remains AWGN, the serial combination of AWGN becomes a different channel. Hence, one cannot use the EXIT chart to see how the channel conditional entropy iterates.

Remark 5.9

Even though the serial combination of AWGN is not an AWGN, it is approximately one. Henceforth, as more of an engineering practice, we can assume it to be so and trace the evolution of the mean and variance of the channel output distribution. We can also argue the quality of the channel by seeing how the mean and variance evolve.

6. Other Topics

This chapter introduces us to plenty topics not directly related to coding theory. However, these topics such as group testing more or less utilize some results of coding theory (mostly Reed–Solomon code) and are interesting enough to be mentioned. The more you know, the better.

6.1. Entropy Complexity

22 Apr.

This section considers the following task: given $\mathrm{Ber}(1/2)$, how do we generate $\mathrm{Ber}(1/3)$? But before any of that, why is the generation of a random variable so important? If the random numbers generated for security purposes are not "random" enough, attackers knowing this knowledge can tailor their attack design to make the encryption scheme especially vulnerable. The generation of a random number is so vital in many of our algorithm design that our societies cannot afford it breaking.

A solution algorithm to the initial problem is shown below:

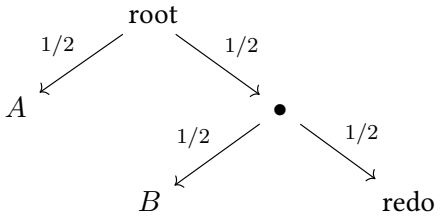


Figure 6.1. Generating $\mathrm{Ber}(1/3)$ from $\mathrm{Ber}(1/2)$.

See the decision tree above in Figure 6.1. The algorithm starts at the root, and go to either left or right based on the output of a single $\mathrm{Ber}(1/2)$ trial. If "redo" is reached, one returns back to the root. The process is repeated until one lands on either A or B. The probability of obtaining the two results is

$$\Pr\{A\} : \Pr\{B\} = \frac{1}{2} : \frac{1}{4} = 2 : 1 \Rightarrow \Pr\{A\} = \frac{2}{3}, \Pr\{B\} = \frac{1}{3}.$$

This algorithm can be easily generalized. Suppose that we want to generate a random variable following $\Pr\{A\} : \Pr\{B\} : \Pr\{C\} = 1 : 2 : 3$, the algorithm can be drawn by a tree of height 3 (since $2^3 > 6 = 1 + 2 + 3$), as shown in the figure below:

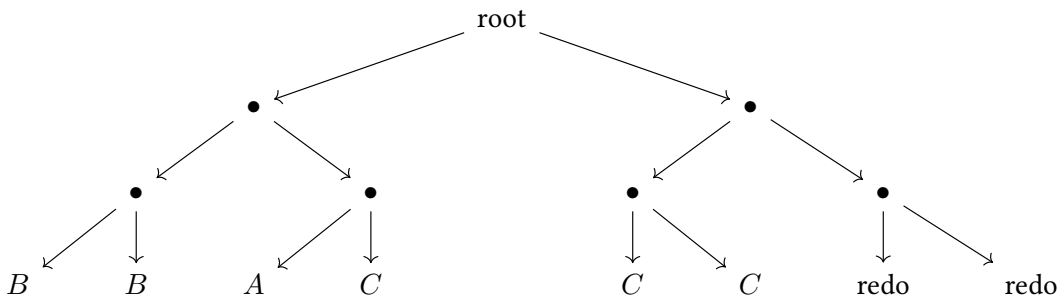


Figure 6.2. Generating $\Pr\{A\} : \Pr\{B\} : \Pr\{C\} = 1 : 2 : 3$ from $\mathrm{Ber}(1/2)$.

The figure above is a waster of resource, however, and we don't really need to do $\mathrm{Ber}(1/2)$ as many times as it suggests. Of course! We can reduce the tree above to a simpler one below:

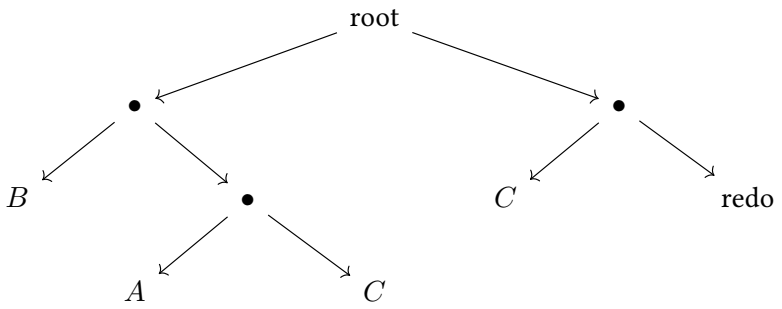


Figure 6.3. Reduced tree of Figure 6.2.

Let the average number of trials in the above algorithm be T, it satisfies

$$T = 2 \cdot \frac{1}{2} + 3 \cdot \frac{1}{4} + T \cdot \frac{1}{4} \Rightarrow T = \frac{7}{3}.$$

This is the optimal algorithm for this example, but in general, how do we know if we can do better? Is there a theoretical bound?

Theorem 6.1: (Knuth & Yao, 1976) [KY76]

The amount of bits needed to describe a discrete random variable X is on average at most

$$H(X) + 2 \text{ bits}. \tag{6.1}$$

This describes the *entropy complexity* of a random variable. To demonstrate, consider the following example:

Example: The worst case possible is if the distribution is the extreme case where the denominator of the distribution is $2^k + 1$, then we need a tree of $k + 1$ layers. Say $\Pr\{A\} : \Pr\{B\} = 2^k : 1$, then expected number of $\mathrm{Ber}(1/2)$ trials will be

$$\begin{aligned} x &= \frac{1}{2} + \frac{1}{2^2}(x+1) + \frac{1}{2^3}(x+2) + \cdots + \frac{1}{2^{k+1}}(x+k) + \frac{k+1}{2^{k+1}} \\ &= \frac{1}{2}\left(1 + \frac{k+1}{2^k}\right) + \frac{1}{2}\left(2 - \frac{k}{2^k} - \frac{1}{2^{k-1}}\right) + x\left(1 - \frac{1}{2^k}\right) \\ \to x &= \frac{2 - (k+2)/2^k}{1 - 1/2^k}. \end{aligned}$$

The entropy is

$$H(X) = \frac{2^k}{2^k+1} \lg\left(\frac{2^k+1}{2^k}\right) + \frac{1}{2^k+1} \lg(2^k+1).$$

We have $x - H(X)$ strictly increasing, and its limit is

$$x - H(X) \xrightarrow{k\to\infty} 2 \text{ bits}. \tag{6.2}$$

□

It seems like $H(X)+2$ is the best guarantee that we can have, and the extra cost of 2 bits is inevitable. Or is it? Given a random vector $Y = (X_1, X_2)$, where X_1 and X_2 are both i.i.d. copies of X. Then, by the theorem above, we will only need

$$H(Y) + 2 = 2\left(H(X) + 1\right) \text{ bits.}$$

On average, we only need $H(X) + 1$ times $\mathrm{Ber}(1/2)$ to generate each X. If we want to generate n copies of X, then the amount of trials needed becomes $H(X) + \frac{2}{n}$. In the limit, we can achieve the entropy. For example, to simulate rolling a dice, you would need $7+1/3$ times $\mathrm{Ber}(1/2$ trials; however, if you were to simulate two rolls of a dice, you only need to do the Bernoulli trial $13+1/3$ times [KY76].

Though we saved the number of Bernoulli trials (the entropy complexity), it tradeoffs with a higher computational complexity.

Remark 6.2

In the 1976 work by Knuth and Yao [KY76], they discussed the optimality of generating any finite or countably infinite set of rational or irrational probabilities. It is definitely worth a read if one wishes to seek deeper understanding of the topic.

6.1.1. Inverse Problem

27 May

We have discussed how we can generate a distribution with a finite support and rational probabilities from $\mathrm{Ber}(1/2)$'s, let us consider now consider the inverse problem: How many $\mathrm{Ber}(1/2)$'s can we generate from a single throw of a uniform distribution of support size d?

Theorem 6.3

Given a uniform distribution with support of size d, we can generate at least

$$\lg d - 2 \tag{6.3}$$

i.i.d. $\mathrm{Ber}(1/2)$'s.

Example: Before we go into the proof, let us first consider an example. Given $X \sim \mathrm{Unif}([13])$, if the outcome is $X = 1$ to 8, then we can assign to X three $\mathrm{Ber}(1/2)$'s. If the outcome is $X = 9$ to 12, then we can assign to X only two $\mathrm{Ber}(1/2)$'s. For the remaining outcome of $X = 13$, no randomness / information can be assigned. Hence, in expectation, the amount of $\mathrm{Ber}(1/2)$'s that one can generate with a single sample of X is

$$\sum(\text{probability}) \cdot (\text{amount}) = \frac{8}{13} \cdot 3 + \frac{4}{13} \cdot 2 + \frac{1}{13} \cdot 0 = \frac{32}{13},$$

which is larger than $\lg 13 - 2 \approx 1.7$. □

Proof. Continue on using the strategy above, assign to each cases a probability p_i where one can assign $\lg(p_i d)$ $\mathrm{Ber}(1/2)$'s. Note that $p_1 \geq \frac{1}{2}$, $p_2 \geq \frac{1}{2}(1 - p_1)$, and so on. Observe that the expected amount of $\mathrm{Ber}(1/2)$'s one can generate is

$$\sum_i p_i \lg(p_i d) = \sum_i p_i \lg p_i + \sum_i p_i \lg d = \lg d - H(\{p_1, p_2, \cdots\}).$$

The function H is the entropy to a sequence of probabilities. Since we have $p_i \geq \frac{1}{2}(1 - p_{i-1})$, one can observe that the sequence of probabilities

$$\boldsymbol{p} := (p_1, p_2, p_3, \cdots) \succeq \boldsymbol{q} := (1/2, 1/4, 1/8, \cdots),$$

where $\succeq$ is the *majorization* between two sequences (for more details, see Appendix A.3). Then, since $\boldsymbol{p}$ *majorizes* $\boldsymbol{q}$, or simply $\boldsymbol{p} \succeq \boldsymbol{q}$, by the *Karamata inequality* over the concave function $-x \ln x$, we have that

$$H(\boldsymbol{p}) \leq H(\boldsymbol{q}).$$

Thus, the amount of $\mathrm{Ber}(1/2)$'s generated is

$$\lg d - H(\boldsymbol{p}) \geq \lg d - H(\boldsymbol{q}) = \lg d - 2.$$

The theorem is proven. ■

Another argument that one can use to proof the theorem above instead of using majorization is by using the additive property of conditional entropy:

$$H(X, Y) = H(X|Y) + H(Y). \tag{6.4}$$

Remark 6.4

The "-2" from the two theorems above are actually the same thing.

6.1.2. The von Neumann Trick

The following problem comes about very naturally: given a biased coin with the bias unknown, how do we play a fair game of "head or tail"?

The von Neumann trick for tackling this problem is:

(a) Toss the biased coin twice.

(b) If you get a Head-Tail, return 1.

(c) If you get a Tail-Head, return 0.

(d) Otherwise, go back to the first step.

03 Jun.

But we can do far better than this naïve strategy! Expand on the idea of the von Neumann trick,

6.2. Group Testing

6 May

Here we continue on with the discussions on group testing from Section 4.7. Let us recap the problem setting: we have $x \in \{0.1\}^{n \times 1}$ where $x_i = 1$ if and only if the ith person is infected, with m being the number of test / measurement. Normally, $k = \|x\|_0$ is small, for example, $k \sim \sqrt{n}$ or n^c with $c \in [0, 1)$, or else the population simply dies out as the disease is so widespread and group testing will be the last of our concerns. Our task is to solve the system of *logical* equations $y = Ax$. One can rewrite this system of logical equations using the usual linear algebra:

$$y^\star = Ax,\ y_i^\star = \sum_j A_{ij} x_j$$

$$\Rightarrow y_i = \min(y_i^\star, 1) \tag{6.5}$$
$$\Rightarrow y = \min(y^\star, \mathbf{1}), \tag{6.6}$$

where $\mathbf{1} \in \mathbb{R}^{n\times 1}$ is the vector of ones.

Besides using the *activation function* $\min(\cdot, 1)$, one can opt for arbitrary functions of similar behavior. For example, the ReLU, sigmoid, or $\arctan$ are some suitable replacements. All that is needed is the monotonicity of the activation function. For more information, please refer to the reference [CJZ22].

6.2.1. Applications of Group Testing

The study of group testing originated from WWII, where it is used to reduce the amount of test required to find out syphilitic males in soldiers. Nowadays, its applications become almost pervasive, appearing in every corner of modern science. Some examples are listed below, with special emphasis put on heavy hitters and bloom filter.

Genotyping: The task of genotyping is to find out whether you have a certain strand of gene that, for example, may cause cancer. The human DNA consists of around 3 billion base pairs, it is too costly and impossible to test one-by-one whether a sequence of gene appears in the DNA. Instead, we can use group testing to aid our task. Given a solution of one's DNA, we can apply the CRISPR (a DNA scissor technology) to cut the DNA into suitable strands and heat up the solution so that the double helix unwinds. Next, we prepare a bunch of already-known sequences of cancer genes and connect them with probes that are light emitting when the sequences are bound by its complementary sequence. By mixing the cancer gene sequences with the DNA solution, some of them combines and emits light. Our detector captures this light emission and determines whether a cancer gene is present. Due to the noise inherent in the experiment, this is actually a harder problem of noisy group testing.

Unsourced access: Given n cellphones near a base station with k of them wanting to talk, how does the base station know which phone is talking what? The time-frequency spectrum is cut into different regions in the communication frequency band: each time slot and frequency slot allows for only one phone transmitting their data (with error correcting code of course). A phone sends its signals by *randomly* choosing a slot. Inevitably, collision may occur where two different phones are talking, and the signals they sent become undecodable. How then, can we resolve collisions in a simplest possible way? The result also requires group testing.

Computer forensics: Imagine having files saved on your computer and a hacker comes along and modified some of them. How do you detect the modification and fix it, or even better, prevent the modification from even happening? The simplest way is to have your computer never connect to the internet, which is impossible for most practical purposes. Instead, one can utilize what is called a *hash function*. A hash function is a special kind of function that is easy to compute, but *very* difficult to invert, viz., $y = \mathrm{hash}(x)$ is easy to compute, $x = \mathrm{hash}^{-1}(y)$ is hard to compute. For each file, we can create their hash values and store it in a separate database – so that if a file doesn't match its hash value, we know that it is modified. Except that it doesn't, the hash values can also be modified by the hacker, and a document may be regarded as modified even though it is not! Instead of having a hash value for each file separately, we can group multiple files together to form a single hash value. These hash values are stored in a separate computer. Since the data size for the hash values is smaller then the previous case, we can apply a higher security level on these data to make them difficult to be modified. Just like the bipartite graph of Figure 4.1, if a file is modified, some of the hash values do not match, and we can use group testing to pinpoint exactly which file is modified.

Image compression: An image can be compressed or be represented by fewer bits than its naïve representation due to the fact that most natural images are sparse in its Fourier domain. Viz., when applying the Fourier transform to a natural image, most of the Fourier coefficients are close to zero and can be omitted. How do we quickly find out the non-zero coefficients? This task, again, can be solved via group testing. For the more educated readers, one might also draw parallel of this task with the research of *compressed sensing* (or compressive sensing). In fact, they are the same! The only difference lies in their choice of the activation function. For generalized group testing, one might use the logarithmic function as an activation function since it mimics the log-scale sensitivity of human perception (in visual and auditory system).

Heavy hitter: The task of heavy hitter is to find out which elements are more popular, i.e., which YouTuber has the most subscribers, who's videos has the most view count, which computer file is used most often. Take the last one for example: In a server, we have SSD (solid state drive) and HDD (hard disk drive) as hardware disks, the former is faster in its read and write processing but more expensive, while the latter is slower but cheaper. A reasonable setup is to have, for exmaple, 250GB to SSD used as system disk and 4TB of HDD allocated for saving less-commonly used documents or old photos. Our task is to find out, out of all the n files, which of the k ones are the most popular? It should be noted that the probability of a file being used follows the famous *Zipf distribution*[1], hence only a small fraction of files are actually commonly used, and we can order them from the most often-used ones to the least used ones by allocating them to registers, caches, RAM, SSD, then finally HDD. To efficiently count the frequency of each entry, one can use a *bloom filter*, which will be introduced immediately.

6.2.2. Bloom Filter

When registering for an account on a website, one often sees the sentence "Invalid username, the username is already taken." How does the server immediately knows that a username appears in the database? It seems impossible to check for every single username in the database for repetition on the fly.

And you are absolutely correct, the server indeed does not check the entire database. Instead, it prepares a string of, for example, 200 bits at hand. The string is all-zero initially when there are no accounts in the database. When an account is added, the database computes a few hash values to the username, these hash values are between 1 to 200. If the username "apple" is registered, the database computes $\mathrm{hash}(0\text{ apple}) = 10$, $\mathrm{hash}(1\text{ apple}) = 3$, and $\mathrm{hash}(2\text{ apple}) = 5$, then it sets the position 10, 3, and 5 of the bit string to 1. The same applies to all usernames created. Then the final string should consist of 0's and 1's. If one register for a username, the server quickly computes the hash values and compare with its bit string. If all the positions corresponding to the hash values are 1 one the bit string, then it is likely that the username is already taken, and the server rejects your registration (see Figure 6.4 below). This is the task of membership testing. Although this method has no missed detection, false positives are still possible. That is not a problem, however, since the company is the boss, and it gets to decide which usernames are possible and you can't argue with it. There is one thing this naïve bloom filter cannot do, that is, it cannot delete accounts.

0	1	1	0	1	0	1	0	0	1	0	⋯

Figure 6.4. Bloom filter. Assume two strings "apple" and "banana" added. They have hash values of 10, 3, 5, and 2, 5, 7. If one wants to add "cherry" into the database, we have to check whether its hash values collide with the 1's in the list.

[1] This distribution appears everywhere!

Bloom counter: Unlike the bloom filter, setting the hash value positions of registered usernames to 1 with logical arithmetic; the bloom counter *adds 1* to the string with usual integer arithmetic. However, now the string will be 200 *bytes* long (1 byte = 8 bits). With this small modification, the bloom counter can delete members. Nevertheless, it cannot delete false positives... why would one implement such functionality anyways?

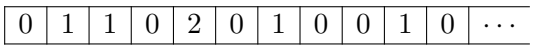


Figure 6.5. Bloom counter. Having the same settings as Figure 6.4.

Database summary: Given a bloom filter, can we quickly summarize who is in the database? The answer is yes! Though up to a small chance of false positives. The simplest approach is to enumerate all possible accounts and check if they are in the database. If the format of the usernames is restricted, then the set of usernames is finite, and the algorithm converges... after a long time. Instead, we can use a bloom filter to quickly summarize the *heavy hitters* in the database. The technique we will introduce now is called the "count sketch" or the "min count sketch", some other algorithms like GroTesQuE (group testing quick and efficient) [Cai+13], AC-DC [Gab+21], SAFFRON [LPR15] can also be utilized. We will touch on a few of these methods in a later subsection.

The count sketch method is basically just a bloom counter but with a really nice hash function. Consider having a string of 1000 integers ($\mathbb{Z}^{1000}$), we will now describe the procedure and details of adding an item into the database and summarizing the whole database.

(1) To add "apple" into the database, turn it into a string of integers, for example, 10010001. How this string is chosen will be clarified later on.

(2) Find the hash values to "apple", say we have $\mathrm{hash}(0\ \mathrm{apple}) = 2$ and $\mathrm{hash}(1\ \mathrm{apple}) = 4$.

(3) Add the string 10010001 to the subarrays with index coinciding with the hash values. See Figure 6.6.

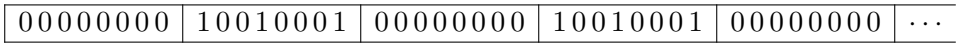


Figure 6.6. Count sketch: adding an item "apple" into the database.

(4) Anytime I see "apple", I add 10010001 to the above hash-value positions again.

(5) Say we added "apple" a total of 25 times, and "banana" a total of 75 times. The string "banana" is represented by 11001001, with hash values 4 and 5. The fourth subarray should have a value of 100 75 0 25 75 0 0 100.

(6) Some other not-so-important entries are added, like "cherry" 10 times, "durian" 5 times, and so on. There might also be noise within the data. The final string should have a form similar to Figure 6.7.

(7) However, due to the noise present, we cannot immediately find out the heavy hitters. The solution is simple, we can use threholding by, for example, 20.

7 1 0 0 1 0 3 1 0	2 5 0 5 2 5 1 0 6 2 5	0 3 0 3 1 0 0 1 0	1 0 0 7 5 0 2 7 7 5 0 2 1 1 0	7 5 8 5 3 0 7 5 0 0 7 7	···

↓ thresholding by 20

0 0 0 0 0 0 0 0	1 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0	1 1 0 1 1 0 0 1	1 1 0 0 1 0 0 1	···

Figure 6.7. Count sketch: final database followed by thresholding to find the heavy hitters.

(8) The second subarray tells us that "apple" is a heavy hitter, and the fifth subarray tells us that "banana" is also a heavy hitter. The fourth subarray, however, is a combination of "apple" and "banana", hence it is undecodable. In the case where there is larger noise, we can use error correcting codes (ECC) to set the string representations of "apple" and "banana." With ECC used, we can counter with 1 or 2 errors.

6.2.3. Binary Search + Group Testing

The tactics and algorithms for group testing are vast, for further reading, one can refer to the references [CN20; LM25; WGG23].

Below, we start our discussion of the naïve tactic of binary search. We first separate the population into two halves, then conduct group testing on both halves. Depending on which half being positive on the first round of tests, we continue on with the second round of tests on it. The amount of tests required is of order $O(\log n)$. This scheme is *adaptive*, since one saw the test results before conducting further tests. In practical cases, this means that this scheme takes a longer time to conduct since it requires so many stages, which is not ideal, we aim for single round group testing which is non-adaptive.

The following discussion follows mainly from the work by Wang, Gabrys, and Guruswami [WGG23]. A non-adaptive scheme for group testing is by binary search. Any binary search is a binary tree.

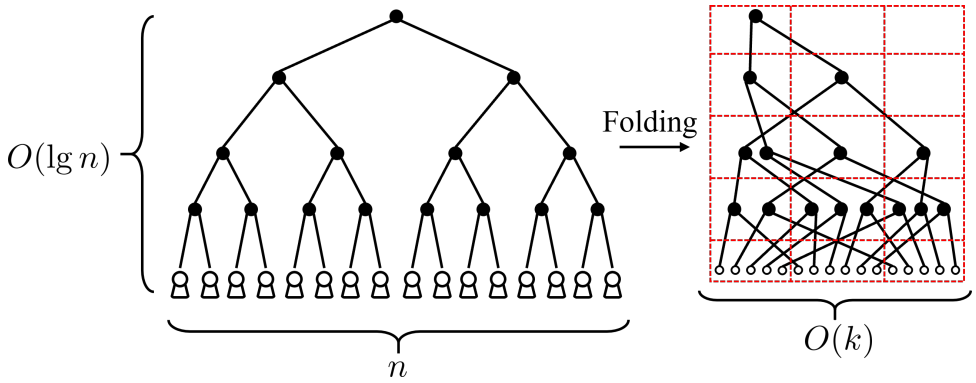


Figure 6.8. Binary search for group testing. The amount of tests required can be greatly reduced by folding the tree.

Normally, a non-adaptive group testing on a binary tree requires $O(n \lg n)$ tests. By simply folding the tree, however, one can devise a scheme that requires only $O(k \lg n)$ tests, this is also the information theoretical limit. See Figure 6.8, each red box is a single test, testing all the leaves of the nodes within it. By pruning the tree with the negative tests, we should be able to find the infected population.

Tracing from the root to the leaves, one can construct a watch list W of all the people suspected to be infected. For simplicity, let both n and k be a power of two, then each person is labeled by a bit string in $\{0,1\}^{\lg n}$. For each layer j, W_j contains the prefixes of the k sick people and with some healthy people

that are still suspicious 🤔. At the first layer (the root), we have $W_1 = \{0, 1\}$ since the test on the root must be positive ($k > 0$). Then, on the second layer, $W_2 = \{\, a0, a1 \,|\, a \in W_1, a \text{ is in a positive test} \,\}$. Continue on, we have

$$W_{j+1} = \left\{\, a0, a1 \,\middle|\, a \in W_j, a \text{ is in a positive test} \,\right\}. \tag{6.7}$$

At each level, we use $16k$ tests (so that the width to the right figure of Figure 6.8 is $16k = O(k)$). Let $w_j = |W_j|$,

$$w_{j+1} = k + \operatorname{Ber}\left(2(w_j - k), \frac{1}{16}\right), \tag{6.8}$$

where the probability $\frac{1}{16}$ is due to the fact that: there are $16k$ tests each layer, and there are k infected person, the probability that each prefix contains no infected person will be pruned away is $15/16$.

Definition 6.5: Probability Generating Function

For a discrete random variable X defined on non-negative integers, its probability generating function is defined as

$$F_X(x) = \sum_{\ell=0}^{\infty} \Pr\{X = \ell\} x^\ell. \tag{6.9}$$

To facilitate discussion, let us define the *excess size* of the watch list as $N_j := w_j - k$ and $F_j = F_{N_j}$. We will also assume that each prefix of length $\lg k$ contains exactly one infected person, then

$$F_{\lg k}(x) = 1, \tag{6.10}$$

$$F_{j+1}(x) = F_j\left(\frac{15}{16} + \frac{x^2}{16}\right) \cdot x^k. \tag{6.11}$$

Equation (6.11) defines a *branching process*, where with probability $\frac{15}{16}$ a suspicious prefix is proven to be innocent, but with probability $\frac{1}{16}$ the number of suspected prefixes doubles. The term x^k means that the k infected people will each result in an innocent prefix.

Let us study the evolution of F_j. Define $f(x) = \frac{15}{16} + \frac{x^2}{16}$, such that

$$F_{j+1}(x) = x^k \cdot f^k(x) \cdot \left(f^{\circ 2}(x)\right)^k \cdot \left(f^{\circ 3}(x)\right)^k \cdot \cdots \cdot \left(f^{\circ j}(x)\right)^k. \tag{6.12}$$

The function f is not entirely easy to iterate, hence we consider using a Möbius transform g to approximate it:

$$g(x) := \frac{6}{y - x} = f(x) + \underbrace{\frac{(x-1)(x-3)^2}{16(7-x)}}_{>0 \text{ for } x \in (1,6)}. \tag{6.13}$$

Then

$$g^{\circ j}(x) = 1 + \frac{5(x-1)}{6^j(6-x) + (x-1)}, \tag{6.14}$$

in particular $g^{\circ j}(5) \approx 1 + \frac{1}{6^j}$. Hence, we can bound F_j by

$$F_j(5) \le \prod_{i=0}^{j - \lg k} g^{\circ i}(5)^k \le \prod_{i=0}^{\infty} \left(1 + \frac{20}{6^i + 4}\right)^k \le \prod_{i=0}^{\infty} \exp\left(\frac{20}{6^i + 4}\right)^k \le \exp(7k).$$

Then we can bound w_j by using the Chernoff bound:

$$\Pr\{w_j - k \ge 5k\} \le \frac{F_j(5)}{5^{5k}} \le \mathrm{e}^{-k}.$$

With high probability, we have $|W_j| \leq 6k$. Our next goal will be to reduce the $6k$ down to k. Which can be done by random testing on these $6k$ people. Those $5k$ people that are innocent will quickly be found out.

Now that we have controlled $|W_j|$, we are not satisfied, yet. We also want to control

$$|W_1| + |W_2| + |W_3| + \cdots .$$

This is the idea of *total progeny*: the sum of all the population from past to present. Our discussion is further aided by the following result.

Theorem 6.6: Dwass

Let TP be the total progeny of some branching process whose probability generating function is $f(t)$, then

$$\Pr\{\mathrm{TP} = m\} = \frac{1}{m}[x^{m-1}]\left(f(x)^m\right), \tag{6.15}$$

i.e., the $(m-1)$th coefficient of $f(x)^m$ over m.

Remark 6.7

The original proof to Theorem 6.6 is given by complex analysis. As a simple exercise, can you think of a combinatorial proof?

Back to our case where $f(x) = \frac{15}{16} + \frac{x^2}{16}$, let M_j be the population of suspicious innocent prefixes on layer j. We have $M_\infty := \lim_{j\to\infty}$ as the total progeny. From the result of Dwass, we have the probability generating function of M_∞ as

$$\begin{aligned} h_\infty(x) &= \sum_{m=1}^{\infty} \frac{x^m}{m}[t^{m-1}]\left(f(t)^m\right) \\ &= \sum_{j=0}^{\infty} \frac{x^{2j+1}}{2j+1}\binom{2j+1}{j}\frac{15^{j+1}}{16^{2j+1}} \\ &= \frac{15x}{16}\sum_{j=0}^{\infty}\frac{1}{j+1}\binom{2j}{j}\left(\frac{15x^2}{16^2}\right)^j \\ &= \frac{15x}{16}\cdot\frac{1-\sqrt{1-4\cdot 15x^2/16^2}}{2\cdot 15x^2/16^2}. \end{aligned}$$

The last equality utilizes the well known generating function of the Catalan numbers, some extra details are provided in Appendix A.2. Let $H_j(x)$ be the total total progeny (TTP) with k added at each level until layer j, i.e.,

$$H_j(x) = \prod_{i=0}^{j-\lg k} h_i(x)^k \tag{6.16}$$

with $h_j(x)$ the probability generating function of M_j. Then, since $h_\infty(2) = 3$,

$$H_j(2) \leq \prod_{i=0}^{j} h_\infty(2)^k = 3^{kj}.$$

Using the Chernoff bound again:

$$\Pr\{\mathrm{TTP} \geq 3k\lg n\} \leq \frac{H_{\lg n}(2)}{2^{3k\lg n}} \leq \frac{\exp(\ln 3\cdot k\lg n)}{\exp(3\ln 2\cdot k\lg n)} \leq n^{-k}. \tag{6.17}$$

This suggests that the time complexity is linear in $k \log n$ with high probability.

6.2.4. Tradeoffs of Group Testing

13 May

As a final remark to group testing, let us consider the tradeoffs between the five dimensions of group testing:

(a) Non-adaptive / adaptive: The group testing is done a single round for non-adaptive ones. For adaptive ones it can be done with multiple round, and future rounds can utilize the knowledge gained from past rounds.

(b) The number of tests m: we often want $m = O(k \log n)$ since it is the information-theoretic bound. In reality, it might be hard to reach.

(c) Noisy / noiseless: whether you can trust the test results.

(d) Decode complexity: we want it to be $O(k \log n)$, $O(k \operatorname{polylog} n)$, or $\operatorname{poly}(k \log n)$.

(e) Number of false positives and false negative (#FP & FN): should be $O(1)$ or $o(k)$.

Our preferred combination would be to have a nonadaptive noiseless group testing with $O(k \log n)$ tests, $O(k \operatorname{polylog} n)$ decoding complexity, and 0 FN & FP with high probability.

The current state of the art can only achieve any combination of four of the above, but never all five.

(a) If we give up the first: then we can use the adaptive binary search method as introduced in the start of Section 6.2.3.

(b) If we give up the second: in GROTESQUE [Cai+13] and SAFFRON [LPR15], we have the number of tests being $m = O(k \log k \log n)$.

(c) If we give up the third: consider the binary tree folding as introduced in Section 6.2.3. Normally when the tests are noiseless, we can cut all the leaves to a node if it is tested to be negative. However, for the case where the tests are noisy, say $\mathrm{BSC}(p)$-noisy, then we must do more tests to be sure. Setting a threshold for the amount the negative results that we accept it as truly being negative, the complexity becomes $p^{c \log k}$.

(d) If we give up the fourth: we can relax the decoding complexity to mn by using a random testing matrix A with $\mathrm{Ber}(1/ck)$ entries. Every $y_i = \min(Ax, 1)$ should hence behave like $\mathrm{Ber}(1/2)$ so that we can actually learn something from y_i (the maximum entropy Bernoulli random variable). To decode, one computes $d_{\mathrm{H}}(y, Ax_i)$ for all $i \in [n]$. If the value is *low*, then the ith person is sick; if the ith person is healthy, the Hamming distance should be $m/2$. Hence, we can set the threshold at about $m/3$. This is the COMP algorithm proposed by Chan et al. [Cha+11]. If done in a brute force manner, the decoding complexity is $m\binom{n}{k}$. Note that in real world applications: in the case of disease control, n is often of the order of hundreds; in communication, n is often of the order of 30^{160}.

(e) If we give up the fifth: the number of FN and FP is of order εk.

Let us discuss a bit more when we give up the fifth criterion and enforce the first four criteria. The current bound is

$$\#\mathrm{FP}\&\mathrm{FN} = O\left(2^{-\sqrt{\lg n}} k\right) \tag{6.18}$$

on average.

In GROTESQUE [Cai+13] and SAFFRON [LPR15], they consider the following scheme for group testing:

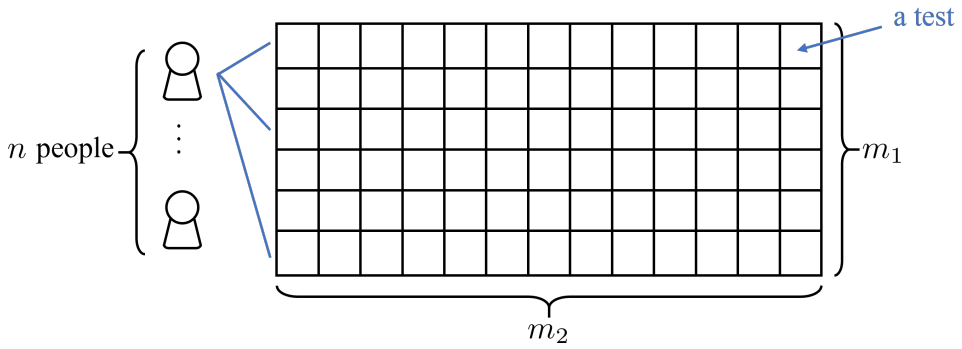


Figure 6.9. A general setup for group testing

See Figure 6.9, there are a total of n people, each with an ID number of length m_2. Each person chooses three rows (out of the total m_1 rows) and fill in their ID number. Tests will be conducted for all the $m = m_1 \cdot m_2$ boxes, and the corresponding numbers in each boxes will show up if at least one person choosing the box is sick. For starters, let us choose an ID string of length m_2, with $m_2 = 100 \lg n$ and $m_1 = 100k$.

Assume that we can always recover the person of an ID string if that string is the only one occupying a row. What happens when multiple people choose the same row? The simplest scheme is to simply give up on any overlaps. Out of the $100k$ rows, there should be $3k$ rows that are sick, so the occupancy rate of sick rows is 3%. In other words, any random person has a probability of $(3\%)^3$ to pick exactly three sick rows. So the number of FPs and FNs is $(3\%)^3 k = \varepsilon k$.

How do we fix this? The following discussion follows from the work by Guruswami and Wang [VW24].

This time, let us set the length of the ID strings to $\lg n$, with $m_2 = 100\sqrt{\lg n}$ and $m_1 = 100k\sqrt{\lg n}$. The amount of tests doesn't change, as it is still of the order $O(k \lg n)$. Further, each person picks 10 rows to fill in their ID a total of 10 times (if a row ends without completing the ID string, the string continues on another row). In this setup, the occupancy rate of sick rows is

$$\frac{10\sqrt{\lg n} \times k}{100k\sqrt{\lg n}} = 10\%.$$

A naïve analysis of the number of FPs and FNs will be

$$\Pr\{\text{all rows chosen are occupied}\} = \left(10\%\right)^{10\sqrt{\lg n}}.$$

Hurray! This is no longer a constant, but a decreasing function of n. However, the above analysis is crude, since one cannot identify themselves by using a single row. Instead, we should use Hoeffding's inequality:

$$\begin{aligned} \Pr\{\text{have fewer than } \sqrt{\lg n} \text{ rows left}\} &< \exp\left(-\text{const.} \times \text{deviation}\right) && (6.19) \\ &= \exp\left(-c \cdot 8\sqrt{\lg n}\right). && (6.20) \end{aligned}$$

On average, one should only loose 1 row due to it being occupied, the worst case is loosing 9 rows, hence the deviation is 8 rows. Finally, we have

$$\#\text{FP\&FN} = O\left(2^{-\sqrt{\lg n}} k\right) = o(k). \tag{6.21}$$

We have achieved our goal.

Some final remarks on how one decode themselves:

Remark 6.8

One can apply Reed–Solomon code to the $10\sqrt{\lg n}$ rows a person chooses to ensure any $\sqrt{\lg n}$ rows given can be used to decode the corresponding string.

Furthermore, how do we know whether two different rows correspond to the same person? The solution is simple, we simply add hash, say of length $50\sqrt{\log n}$, to the front of each row. This was also proposed in the work by Guruswami and Wang [VW24].

Remark 6.9

Instead of having $(\lg n)^{1/2}$, other exponents can be used.

Say each string of ID is of length $(\lg n)^{2/3}$, each person chooses enough rows such that 10 copies of the ID are written. Let $m_1 = 100k(\lg n)^{2/3}$, $m_2 = 100(\lg n)^{1/3}$, and the length of hash is $50(\lg n)^{1/3}$. What happens now? The probability of having the chosen rows being too occupied will now be $\exp(-c \cdot 8(\lg n)^{2/3})$, this is much better than before! However, hash collisions may occur. The probability of hash collision will be $2^{-(\lg n)^{1/3}}$. The final rate of FPs and FNs will be the max of the two.

But wait, we can apply the same scheme again! We add hash values to the previous hash values that have collision. Then the number of FPs and FNs will be

$$2^{-(\lg n)^{2/3}}k.$$

Why stop here? Applying this scheme again, we will have $2^{-(\lg n)^{3/4}}k$. Eventually, we can achieve

$$2^{-(\lg n)^{1-1/\tau}}k. \tag{6.22}$$

6.3. Distributed Matrix Multiplication

Consider given the task of calculating the matrix product

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \cdot \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{bmatrix},$$

it requires 8 scalar multiplications. If we distribute this 8 tasks to 8 GPUs to compute and output the result only after all 8 GPUs finish, our computation will be vulnerable when some GPUs are slow. This may be due to some randomness, overheating, power outage, or our favorite, Microsoft system auto-update.

We can partition A into

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \Rightarrow \tilde{A}_{2/3} := \begin{bmatrix} A_1 \\ A_2 \\ A_1 + A_2 \end{bmatrix}.$$

If we compute the product $\tilde{A}B$, and distribute the task A_1B, A_2B, and $(A_1 + A_2)B$ to three separate processors, we can recover AB once any two finish. Since matrix addition is really cheap ($O(n)$) in comparison to multiplication ($O(n^3)$). We only care about reducing the multiplication count.

Let us continue on with this strategy, and define

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \Rightarrow \tilde{A}_{3/5} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_1 + A_2 + A_3 \\ A_1 + 2A_2 + 3A_3 \end{bmatrix},$$

this is a 3-out-of-5 system (3/5-system). By incorporating more redundant tasks, we can prevent our computation being affected by slow GPUs. In fact, by employing an $[n, k]$-Reed–Solomon code, we can implement a k/n-system. This is called the *polynomial code*, and will be discussed later in Section 6.3.5.

We can do better! Besides defining $\tilde{A}_{2/3}$, we also define $\tilde{B}_{2/3} = [B_1, B_2, B_1 + B_2]$. Then we can use 9 GPUs to compute $\tilde{A}_{2/3}\tilde{B}_{2/3}$, and requiring only 5 of them to obtain AB. Its extension will be a two-dimensional Reed–Solomon code.

In general, we find good matrices G and H as row and column operations, respectively, and compute

$$\tilde{A}\tilde{B} = (GA)(BH). \tag{6.23}$$

6.3.1. Polynomial Codes

Let us introduce the method of polynomial codes [YMA18]: For A and B being $n \times n$ matrices, consider computing

$$(\boldsymbol{g}_s \cdot A)(B \cdot \boldsymbol{h}_s) \tag{6.24}$$

at the sth GPU, with vectors

$$\boldsymbol{g}_s = \begin{bmatrix} 1 & \xi_s & \cdots & \xi_s^{n-1} \end{bmatrix}, \boldsymbol{h}_s = \begin{bmatrix} 1 \\ \xi_s^n \\ \vdots \\ \xi_s^{n^2-n} \end{bmatrix}, \tag{6.25}$$

and $\xi_1, \ldots, \xi_s, \ldots$ are all distinct.

Theorem 6.10: Polynomial Codes

Any n^2 GPUs recovers AB.

Proof. Observe that

$$\boldsymbol{g}_s \cdot AB \cdot \boldsymbol{h}_s = \operatorname{Tr}\{\boldsymbol{h}_s\boldsymbol{g}_s \cdot AB\} = \langle \boldsymbol{h}_s\boldsymbol{g}_s, AB \rangle,$$

where $\langle \cdot, \cdot \rangle$ is the Frobenius inner product. Since $\boldsymbol{h}_s\boldsymbol{g}_s$ forms a matrix basis X_s: we can compute n^2 of

$$\langle X_1, AB \rangle = \operatorname{Tr}\left\{X_1(AB)\right\}, \langle X_2, AB \rangle = \operatorname{Tr}\left\{X_2(AB)\right\}, \ldots$$

so as to finally obtain AB. ∎

6.3.2. MatDot

Let us introduce the method of MatDot [LMV22][2]: Let the sth CPU compute the product

$$A\boldsymbol{x}_s\boldsymbol{y}_s B, \tag{6.26}$$

with $\boldsymbol{x}_s = [1, \xi_s, \ldots, \xi_s^{n-1}]^\mathsf{T}$ and $\boldsymbol{y}_s = [\xi_s^{n-1}, \ldots, \xi_s, 1]$. Again, all the ξ_s's are distinct.

[2] To be frank, I think the polynomial codes is more related to "dot" product.

Theorem 6.11: MatDot

Any $(2n-1)$-subset of $A\boldsymbol{x}_s\boldsymbol{y}_sB$ recovers AB.

Proof. Observe that

$$\boldsymbol{x}_s\boldsymbol{y}_s = \begin{bmatrix} \xi_s^{n-1} & \cdots & \xi_s & 1 \\ \vdots & \ddots & & \xi_s \\ \xi_s^{2n-3} & & \ddots & \vdots \\ \xi_s^{2n-2} & \xi_s^{2n-3} & \cdots & \xi_s^{n-1} \end{bmatrix}$$

is a Toeplitz matrix. And any $(2n-1)$-subset of $A\boldsymbol{x}_s\boldsymbol{y}_sB$ spans all Toeplitz matrix, which includes the identity matrix $\mathbb{1}$. Henceforth, we can obtain $A\mathbb{1}B = AB$ from linear combinations of $2n-1$ $A\boldsymbol{x}_s\boldsymbol{y}_sB$'s. ■

6.3.3. Strassen's Method

The classical method for fast 2×2 matrix multiplication is the Strassen method, requiring only 7 scalar multiplications. See the terms below

$$\begin{aligned} S^{(1)} &= (A_{11}+A_{22})(B_{11}+B_{22}), \\ S^{(2)} &= (A_{21}+A_{22})B_{11}, \\ S^{(3)} &= A_{11}(B_{12}-B_{22}), \\ S^{(4)} &= A_{22}(-B_{11}+B_{21}), \\ S^{(5)} &= (A_{11}+A_{12})B_{22}, \\ S^{(6)} &= (-A_{21}+A_{11})(B_{11}+B_{12}), \\ S^{(7)} &= (A_{12}-A_{22})(B_{21}+B_{22}). \end{aligned}$$

Let $C = AB$, then

$$\begin{aligned} C_{11} &= S^{(1)} + S^{(4)} - S^{(5)} + S^{(7)}, \\ C_{12} &= S^{(3)} + S^{(5)}, \\ C_{21} &= S^{(2)} + S^{(4)}, \\ C_{22} &= S^{(1)} - S^{(2)} + S^{(3)} + S^{(6)}. \end{aligned}$$

If we want to extend Strassen's method to distributed computing, we should add some redundancy: consider

$$\begin{aligned} S^{(8)} &= (A_{11}+2A_{21})(-B_{11}+B_{12}), \\ S^{(9)} &= (A_{12}+2A_{22})(-B_{21}+B_{22}). \end{aligned}$$

Theorem 6.12

Any 8 of $S^{(i)}$'s recover $C = AB$.

Proof. Consider the product:

$$\text{linear combination of } C_{ij}\text{'s} = \begin{bmatrix} 1 & 2 \end{bmatrix} C \begin{bmatrix} -1 \\ 1 \end{bmatrix} = S^{(8)} + S^{(9)} = \text{linear combination of } S^{(1)} \text{ to } S^{(7)},$$

where the last equality is by knowing that the Strassen's method works. The linear combination is

$$(1, -4, 3, -3, 2, 2, -1) \cdot (S^{(1)}, \ldots, S^{(7)}) = S^{(8)} + S^{(9)}. \tag{6.27}$$

Henceforth, if any single one of the $S^{(i)}$'s is missing, we can use the above linear equation to recover it.

If we want to counter 2 missing data, we can construct

$$\begin{bmatrix} 3 & -1 \end{bmatrix} C \begin{bmatrix} 1 \\ 2 \end{bmatrix} = S^{(10)} + S^{(11)}.$$

Then we just need to solve

$$\left[\begin{array}{ccccccc:cccc} 1 & -4 & 3 & -3 & 2 & 2 & -1 & -1 & -1 & 0 & 0 \\ 1 & 1 & 4 & 2 & 3 & -2 & 3 & 0 & 0 & -1 & -1 \end{array}\right] \begin{bmatrix} S^{(1)} \\ \vdots \\ S^{(7)} \\ \hdashline S^{(8)} \\ \vdots \\ S^{(11)} \end{bmatrix} = 0 \tag{6.28}$$

to recover the missing terms. This is possible due to the fact that any 2×2 minor (containing at least one column from the first 7 columns) of the matrix on the left in Equation (6.28) is invertible. ■

This set of combinations is found by brute force via a computer.

Remark 6.13

It should be noted that Strassen's method is not optimal, as faster methods exist, such as the Coppersmith-Winograd method.

Remark 6.14

Interestingly enough, just a few days after the lecture, the team of Google Deepmind published a result on fast matrix multiplication from their AI AlphaTensor on 15th of May, 2025. The traditional method of multiplying a 4×4 matrix using the Strassen's method requires 49 scalar multiplications. Amazingly, the method proposed by Deepmind requires only 48 scalar multiplications! What a breakthrough.

6.3.4. Laderman's Method

The Laderman's method [Lad76] deals with fast 3×3 matrix multiplication, requiring only 23 scalar multiplications! I will not explicitly write out the 23 terms $L^{(1)}$ to $L^{(23)}$, however, we will discuss an extension of it to the distributed case.

Compared to the direct method requiring $3^3 = 27$ multiplications, we can add 3 more redundancies and still outperform direct computation. We can obtain $L^{(24)}$, $L^{(25)}$ and $L^{(26)}$ from the product

$$\begin{bmatrix} 2 & 1 & 3 \end{bmatrix} C \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} = L^{(24)} + L^{(25)} + L^{(26)}, \tag{6.29}$$

just like what we did for the distributed case of Strassen's method.

6.3.5. Polynomial Code

03 Jun.

Not to be confused with the *polynomial codes* in Section 6.3.1, this section is a continuation on the details of implementing a k/n-system for distributed matrix multiplication introduced at the start of Section 6.3.

6.4. Quantum Error Correcting Code

20 May

6.4.1. Mathematical Fundamentals to Quantum Information / Computation

First off, some basics to the mathematics of quantum theory. The following description to the quantum theory is an overly simplified one, with plenty of its historical backgrounds missing, leaving only the essential mathematical results. And as always, don't forget that "all models are wrong, but some are useful." The mathematical formulation to quantum mechanics has evolved for a hundred years, and it is, to this date, the most successful mathematical description we have of our observed world. It doesn't mean that is how the world actually works, but just that the mathematics checks out with the experiment that we cared about and can actually conduct.

In quantum information science, the most basic information-carrying unit is a qubit, which is described by a trace-1 positive semi-definite operator ρ termed the density operator acting on the vector space $\mathbb{C}^2$. With multiple qubits combined, we obtain a quantum system. When we have multiple qubits, say ρ_1 and ρ_2, the combined systems is described by a larger density operator $\rho_1 \otimes \rho_2$, acting on the combined vector space.

An experiment / measurement conducted on a quantum system is a partition of unity: each event i is described by a positive semi-definite operator E_i satisfying

$$\sum_i E_i = \mathbb{1}. \tag{6.30}$$

When conducting the experiment $E = \{E_i\}$ on a system ρ, the probability of seeing the outcome i is defined by the Born rule:

$$\Pr\{i\} = \operatorname{Tr}\{E_i \rho\} \in [0, 1]. \tag{6.31}$$

Some often used measurements are $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ and $\{|+\rangle\langle +|, |-\rangle\langle -|\}$, where a *bra* vector is $\langle x| := |x\rangle^\dagger$, and the *ket* vectors of the computational basis are defined as $|0\rangle := [1, 0]^\mathsf{T}$, $|1\rangle := [0, 1]^\mathsf{T}$, and $|\pm\rangle := \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$. If two vectors are tensored together, we simply denote it by concatenation, i.e., $|00\rangle = |0\rangle \otimes |0\rangle$.

The evolution of a quantum system is unitary, viz., for a state ρ evolving through a quantum system, there exists a unitary U such that

$$\rho \mapsto U \rho U^\dagger \tag{6.32}$$

represents the evolution.

For further understanding of the mathematics behind quantum information science, one should refer to any other book but this note.

6.4.2. Quantum Error Correcting Code

Our task is to protect our density operator ρ from close-to-identity unitary evolutions / noises. When applying the $\{|+\rangle\langle+|, |-\rangle\langle-|\}$-measurement, the probability of obtaining the outcome $+$ should be

$$\Pr\{+\} = \operatorname{Tr}\left\{|+\rangle\langle+|\rho\right\} = \langle+|\rho|+\rangle .$$

However, under a unitary noise U, the result will in turn be

$$\langle+|U\rho U^\dagger|+\rangle .$$

Our first attempt to correct this is to duplicate ρ, say into three copies. If U happens with a probability of 3%, then with the three copies, we can use majority voting and reduce the error probability to 1%.

But we can do much better by utilizing the *entanglement* nature of the quantum theory. Again given three qubits, we can thus construct density operators of size 8×8. For example, the state

$$\rho = \frac{1}{2}\begin{bmatrix} 1 & & & & & & & \\ & 0 & & & & & & \\ & & 0 & & & & & \\ & & & 0 & & & & \\ & & & & 0 & & & \\ & & & & & 0 & & \\ & & & & & & 0 & \\ & & & & & & & 1 \end{bmatrix} = \frac{1}{2}\left(|000\rangle\langle000| + |111\rangle\langle111|\right) \tag{6.33}$$

is a repetition code, whereas

$$\rho = \frac{1}{4}\begin{bmatrix} 1 & & & & & & & \\ & 0 & & & & & & \\ & & 0 & & & & & \\ & & & 1 & & & & \\ & & & & 0 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 0 \end{bmatrix} = \frac{1}{4}\left(|000\rangle\langle000| + |011\rangle\langle011| + |101\rangle\langle101| + |110\rangle\langle110|\right) \tag{6.34}$$

is a parity check code.

This is essentially what quantum error correcting code (QECC) is! We encode our usual codes within quantum states.

Of course, our encoding need not be in the $\{|0\rangle, |1\rangle\}$ basis, we can have

$$\rho = \frac{1}{4}\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = \frac{1}{2}\left(|+++\rangle\langle+++| + |---\rangle\langle---|\right) . \tag{6.35}$$

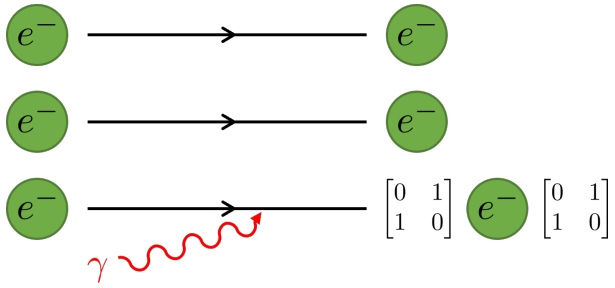


Figure 6.10. Bit flip of a quantum system. Each qubit is represented by an electron e^-. The third qubit is hit by an incident photon γ that changes the state by the unitary matrix X.

A kind of noise that often occurs is a bit flip by a photon, see Figure 6.10. The unitary gate caused by an incident photon acting on the third qubit is the Pauli-X gate:

$$X := \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, \tag{6.36}$$

which is also known as the NOT gate as it satisfies

$$X|0\rangle = |1\rangle\,,\ X|1\rangle = |0\rangle\,. \tag{6.37}$$

The total unitary that acts on the 3-qubit system is

$$U = \mathbb{1}\otimes\mathbb{1}\otimes X = \begin{bmatrix}0 & 1 & & & & \\ 1 & 0 & & & & \\ & & 0 & 1 & & \\ & & 1 & 0 & & \\ & & & & 0 & 1\\ & & & & 1 & 0\end{bmatrix}. \tag{6.38}$$

Assume the input state is the repetition code density operator as described in Equation (6.33), how, then, do we detect and fix the error induced? Consider the following quantum circuit in Figure 6.11.

Figure 6.11. Fixing a bit flip error.

If the first measurement turns out to be

6.5. Database

Here we discuss the *relational databases* $R(X,Y)$. Given 27 May

An interesting theorem regarding bipartite graphs and entropy is stated as follows. This is a generalization to the work by Suciu [Abo+24]:

Theorem 6.15: Entropy and Degree of Graph

Let $G(U,V,E)$ be a bipartite graph, where U and V are the two parts of vertices, and E is a subset of $U\times V$. Let $(X,Y)\in E$ be a pair of random variables. Then

$$aH(X|Y) + H(X,Y) + bH(Y|X) \le \log \sum_{(u,v)\in E} \deg(v)^a \deg(u)^b \tag{6.39}$$

for any $a, b > 0$. The bound is tight.

Proof.

$$aH(X|Y) + H(X,Y) + bH(Y|X) = \mathbb{E}\left[\log \frac{1}{p(X,Y)}\right] + a\mathbb{E}\left[\log \frac{1}{p(X|Y)}\right] + b\mathbb{E}\left[\log \frac{1}{p(Y|X)}\right],$$

where the expectations are taken with respect to the random variables $(X,Y) \in E$. Then, since for any probability measure P (over a finite set) and any other distribution Q subject to $Q = 0 \Rightarrow P = 0$ we have the positivity to the KL-divergence:

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_P\left[\log \frac{P}{Q}\right] \geq 0 \Rightarrow \mathbb{E}_P\left[\log \frac{1}{Q}\right] \geq \mathbb{E}_P\left[\log \frac{1}{P}\right].$$

Let us choose the distribution Q as the uniform distribution: set $q(X|Y) = 1/\deg(Y)$ and $q(Y|X) = 1/\deg(X)$, then we have

$$\begin{aligned} aH(X|Y) + H(X,Y) + bH(Y|X) &\leq \mathbb{E}\left[\log \frac{1}{p(X,Y)}\right] + a\mathbb{E}\left[\log \deg(Y)\right] + b\mathbb{E}\left[\log \deg(X)\right] \\ &= \mathbb{E}\left[\log \frac{\deg(Y)^a \deg(X)^b}{p(X,Y)}\right] \\ &\leq \log \mathbb{E}\left[\frac{\deg(Y)^a \deg(X)^b}{p(X,Y)}\right] = \log \sum_{(u,v)\in E} \deg(v)^a \deg(u)^b, \end{aligned}$$

where the last inequality is by Jensen's. ■

6.6. Determinant Code

[EM19]

6.7. Capacity-Achieving Gray Codes

03 Jun.

Bibliography

[Arı16] Erdal Arıkan. "On the Origin of Polar Coding". In: *IEEE Journal on Selected Areas in Communications* 34.2 (2016), pp. 209–223. DOI: 10.1109/JSAC.2015.2504300.

[Ari09] Erdal Arikan. "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels". In: *IEEE Transactions on Information Theory* 55.7 (2009), pp. 3051–3073. DOI: 10.1109/TIT.2009.2021379.

[AT09] Erdal Arikan and Emre Telatar. "On the rate of channel polarization". In: *2009 IEEE International Symposium on Information Theory*. 2009, pp. 1493–1495. DOI: 10.1109/ISIT.2009.5205856.

[Sha48] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[WD21] Hsin-Po Wang and Iwan M. Duursma. "Polar Codes' Simplicity, Random Codes' Durability". In: *IEEE Transactions on Information Theory* 67.3 (2021), pp. 1478–1508. DOI: 10.1109/TIT.2020.3041570.

[Dur19] Rick Durrett. *Probability: Theory and Examples*. Duke Mathematics Department, 2019.

[RU07] T. Richardson and R. Urbanke. *Modern Coding Theory*. Cambridge University Press, 2007.

[GR20] Naveen Goela and Maxim Raginsky. "Channel Polarization Through the Lens of Blackwell Measures". In: *IEEE Transactions on Information Theory* 66.10 (2020), pp. 6222–6241. DOI: 10.1109/TIT.2020.3016605.

[JA18] T. S. Jayram and Erdal Arıkan. "A Note on Some Inequalities Used in Channel Polarization and Polar Coding". In: *IEEE Transactions on Information Theory* 64.8 (2018), pp. 5767–5768. DOI: 10.1109/TIT.2017.2717598.

[Mur22] Jun Muramatsu. "Binary Polar Codes Based on Bit Error Probability". In: *2022 IEEE International Symposium on Information Theory (ISIT)*. 2022, pp. 2148–2153. DOI: 10.1109/ISIT50566.2022.9834407.

[MHU16] Marco Mondelli, S. Hamed Hassani, and Rüdiger L. Urbanke. "Unified Scaling of Polar Codes: Error Exponent, Scaling Exponent, Moderate Deviations, and Error Floors". In: *IEEE Transactions on Information Theory* 62.12 (2016), pp. 6698–6712. DOI: 10.1109/TIT.2016.2616117.

[Wan+23] Hsin-Po Wang et al. "Sub-4.7 Scaling Exponent of Polar Codes". In: *IEEE Transactions on Information Theory* 69.7 (2023), pp. 4235–4254. DOI: 10.1109/TIT.2023.3253074.

[HY13] Junya Honda and Hirosuke Yamamoto. "Polar Coding Without Alphabet Extension for Asymmetric Models". In: *IEEE Transactions on Information Theory* 59.12 (2013), pp. 7829–7838. DOI: 10.1109/TIT.2013.2282305.

[VS23] Atri Rudra Venkatesan Guruswami and Madhu Sudan. *Essential Coding Theory*. University at Buffalo, 2023.

[KS64] W. Kautz and R. Singleton. "Nonrandom binary superimposed codes". In: *IEEE Transactions on Information Theory* 10.4 (1964), pp. 363–377. DOI: 10.1109/TIT.1964.1053689.

[KY76] D. Knuth and A. Yao. "The complexity of nonuniform random number generation". In: *Algorithms and Complexity: New Directions and Recent Results*. Academic Press, 1976, pp. 357–428.

[CJZ22] Xiwei Cheng, Sidharth Jaggi, and Qiaoqiao Zhou. *Generalized Group Testing*. 2022. arXiv: 2102.10256 [cs.IT]. URL: https://arxiv.org/abs/2102.10256.

[Cai+13] Sheng Cai et al. *GROTESQUE: Noisy Group Testing (Quick and Efficient)*. 2013. arXiv: 1307.2811 [cs.IT]. URL: https://arxiv.org/abs/1307.2811.

[Gab+21] Ryan Gabrys et al. *AC-DC: Amplification Curve Diagnostics for Covid-19 Group Testing*. 2021. arXiv: 2011.05223 [q-bio.QM]. URL: https://arxiv.org/abs/2011.05223.

[LPR15] Kangwook Lee, Ramtin Pedarsani, and Kannan Ramchandran. *SAFFRON: A Fast, Efficient, and Robust Framework for Group Testing based on Sparse-Graph Codes*. 2015. arXiv: 1508.04485 [cs.IT]. URL: https://arxiv.org/abs/1508.04485.

[CN20] Mahdi Cheraghchi and Vasileios Nakos. *Combinatorial Group Testing and Sparse Recovery Schemes with Near-Optimal Decoding Time*. 2020. arXiv: 2006.08420 [cs.IT]. URL: https://arxiv.org/abs/2006.08420.

[LM25] Xiaxin Li and Arya Mazumdar. *Noisy Nonadaptive Group Testing with Binary Splitting: New Test Design and Improvement on Price-Scarlett-Tan's Scheme*. 2025. arXiv: 2410.14566 [cs.IT]. URL: https://arxiv.org/abs/2410.14566.

[WGG23] Hsin-Po Wang, Ryan Gabrys, and Venkatesan Guruswami. "Quickly-Decodable Group Testing with Fewer Tests: Price–Scarlett's Nonadaptive Splitting with Explicit Scalars". In: *2023 IEEE International Symposium on Information Theory (ISIT)*. 2023, pp. 1609–1614. DOI: 10.1109/ISIT54713.2023.10206843.

[Cha+11] Chun Lam Chan et al. "Non-adaptive probabilistic group testing with noisy measurements: Near-optimal bounds with efficient algorithms". In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2011, pp. 1832–1839. DOI: 10.1109/Allerton.2011.6120391.

[VW24] Venkatesan and Hsin-Po Wang. *Nonadaptive Noise-Resilient Group Testing with Order-Optimal Tests and Fast-and-Reliable Decoding*. 2024. arXiv: 2311.08283 [cs.IT]. URL: https://arxiv.org/abs/2311.08283.

[YMA18] Qian Yu, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. *Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication*. 2018. arXiv: 1705.10464 [cs.IT]. URL: https://arxiv.org/abs/1705.10464.

[LMV22] Hiram H. López, Gretchen L. Matthews, and Daniel Valvo. "Secure MatDot codes: a secure, distributed matrix multiplication scheme". In: *2022 IEEE Information Theory Workshop (ITW)*. 2022, pp. 149–154. DOI: 10.1109/ITW54588.2022.9965839.

[Lad76] Julian D. Laderman. "A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications". In: *Bulletin of the American Mathematical Society, Bull. Amer. Math. Soc.* 82.1 (1976), pp. 126–128. DOI: 10.1002/j.1538-7305.1948.tb01338.x. URL: https://www.ams.org/journals/bull/1976-82-01/S0002-9904-1976-13988-2/S0002-9904-1976-13988-2.pdf.

[Abo+24] Mahmoud Abo Khamis et al. "Join Size Bounds using lp-Norms on Degree Sequences". In: *Proc. ACM Manag. Data* 2.2 (May 2024). DOI: 10.1145/3651597. URL: https://doi.org/10.1145/3651597.

[EM19] Mehran Elyasi and Soheil Mohajer. "Determinant Codes With Helper-Independent Repair for Single and Multiple Failures". In: *IEEE Transactions on Information Theory* 65.9 (2019), pp. 5469–5483. DOI: 10.1109/TIT.2019.2904294.

[Gre76] Curtis Greene. "Weight Enumeration and Geoemtry of Linear Codes". In: (1976). DOI: https://doi.org/10.1002/sapm1976552119.

[Dwa69] Meyer Dwass. "The Total Progeny in a Branching Process and a Related Random Walk". In: *Journal of Applied Probability* 6.3 (1969), pp. 682–686. ISSN: 00219002. URL: http://www.jstor.org/stable/3212112 (visited on 05/19/2025).

A. Extra Proofs to Theorems and Lemmas

A.1. MacWilliams' Identity

The following proofs follows from the paper "Weight Enumeration and the Geometry of Linear Codes" [Gre76] by C. Greene.

A.1.1. Corank-Nullity Generating Function of the Dual Code

This is an alternate proof to Lemma 4.24.

Proof. Denote $E = \{1, 2, \ldots, n\}$ be the indices and S a subset of E. We denote $r(S)$ the rank of the columns S of a generator matrix of a code $\mathcal{C}$, and $r^*(S)$ the rank of the columns S of a generator matrix of the dual code $\mathcal{C}^\perp$.

A result from *matroid theory* states that

$$r^*(S) = |S| + r(E \backslash S) - r(E). \tag{A.1}$$

This equation is related to Tutte-Grothedieck invariance, and is in essence, derived from combinatorics arguments. Hence, this proof actually still uses counting. Nevertheless, let us assume that this equation is known and continue on.

Let us consider the corank-nullity generating function of the dual code:

$$\begin{aligned} W_{\mathcal{C}^\perp}(x, y) &= \sum_{S \subseteq E} x^{r^*(G) - r^*(S)} y^{|S| - r^*(S)} \\ &= \sum_{S \subseteq E} x^{(n-k) - (|S| + r(E\backslash S) - r(G))} y^{|S| - (|S| + r(E\backslash S) - r(G))} \\ &= \sum_{S \subseteq E} y^{r(G) - r(E\backslash S)} x^{(n - |S|) - r(E\backslash S)} = W_{\mathcal{C}}(y, x). \end{aligned}$$

Thus it is shown. ∎

A.1.2. Greene's Lemma

This is an alternate proof to Lemma 4.25.

Proof. Let $E = \{1, 2, \ldots, n\}$ be the indices for a codeword. The support of a codeword $c \in \mathcal{C}$ is defined as

$$\operatorname{supp}(c) := \{\, i \in E \,|\, c_i = 1 \,\}.$$

The weight enumerator can then be rewritten as

$$A_{\mathcal{C}}(x, y) = \sum_{S \subseteq E} \#\left\{\, c \in \mathcal{C} \,\middle|\, \operatorname{supp}(c) = S \,\right\} x^{|S|} y^{n - |S|}.$$

The RHS of the statement can then be reduced to

$$\begin{aligned}(y-x)^k x^{n-k} W_{\mathcal{C}}\left(\frac{2x}{y-x}, \frac{y-x}{x}\right) &= \sum_{T\subseteq E} 2^{k-r(T)} x^{n-|T|}(y-x)^{|T|} \\ &= \sum_{T\subseteq E}\sum_{m=0}^{|T|}(-1)^{|T|-m} 2^{k-r(T)}\binom{|T|}{m} x^{n-m} y^m \\ &= \sum_{T\subseteq E}\sum_{S\subseteq T}(-1)^{|T|-|S|} 2^{k-r(T)} x^{n-|S|} y^{|S|} \\ &= \sum_{S\subseteq E}\sum_{T\supseteq S}(-1)^{|T|-|S|} 2^{k-r(T)} x^{n-|S|} y^{|S|}.\end{aligned}$$

Next, we replace $S \mapsto S^{\mathrm{c}}$ and $T \mapsto T^{\mathrm{c}}$,

$$\begin{aligned}&= \sum_{S^{\mathrm{c}}\subseteq E}\sum_{T^{\mathrm{c}}\supseteq S^{\mathrm{c}}}(-1)^{|T^{\mathrm{c}}|-|S^{\mathrm{c}}|} 2^{k-r(T^{\mathrm{c}})} x^{n-|S^{\mathrm{c}}|} y^{|S^{\mathrm{c}}|} \\ &= \sum_{S\subseteq E}\left(\sum_{T\subseteq S}(-1)^{|S|-|T|} 2^{k-r(T^{\mathrm{c}})}\right) x^{|S|} y^{n-|S|}.\end{aligned}$$

We need to show that the term in the bracket is equal to $\#\left\{\, c\in\mathcal{C} \,\middle|\, \mathrm{supp}(c)=S \,\right\}$. This can easily be done by introducing the following lemma.

Lemma 1.1: Möbius Inversion

For set functions f and g, we have

$$f(S) = \sum_{T\subseteq S}(-1)^{|S|-|T|} g(T) \Leftrightarrow g(S) = \sum_{T\subseteq S} f(T). \tag{A.2}$$

Proof. (Möbius inversion) The lemma is simply proven below:

$$\begin{aligned}\sum_{T\subseteq S}(-1)^{|S|-|T|}\sum_{V\subseteq T} f(V) &= \sum_{V\subseteq S}\sum_{T\supseteq V}(-1)^{|S|-|T|} f(V) \\ &= \sum_{V\subseteq S}(-1)^{|S|-|V|} f(V)\sum_{m=0}^{|S|-|V|}\binom{|S|-|V|}{m}(-1)^m \\ &= \begin{cases}\displaystyle\sum_{V\subseteq S}(-1)^{|S|-|V|} f(V)(1-1)^{|S|-|V|} = 0, & S\neq V; \\ \displaystyle\sum_{V=S}(-1)^{|S|-|V|} f(V) = f(S), & S = V.\end{cases}\end{aligned}$$

□

With the lemma proven, let us continue with the proof of Greene's lemma. Let the set functions be $f(S) = \#\left\{\, c\in\mathcal{C} \,\middle|\, \mathrm{supp}(c)=S \,\right\}$ and $g(S) = 2^{k-r(S^{\mathrm{c}})}$. Since

$$\sum_{T\subseteq S}\#\left\{\, c\in\mathcal{C} \,\middle|\, \mathrm{supp}(c)=T \,\right\} = \#\left\{\, c\in\mathcal{C} \,\middle|\, \mathrm{supp}(c)\subseteq S \,\right\} = 2^{k-r(S^{\mathrm{c}})},$$

where the last equality is obtained since the generator matrix can be written in the full rank form below through column operations:

$$G = \left[\begin{array}{c:c} \mathbb{F}_2^{r(S^{\mathrm{c}})\times|S|} & \mathbb{F}_2^{r(S^{\mathrm{c}})\times(n-|S|)} \\ \hdashline \mathbb{F}_2^{r(\bar{S})\times|\bar{S}|} & 0 \end{array}\right] \in \mathbb{F}_2^{k\times n}.$$

The Greene's lemma is finally proven by applying Möbius inversion. ■

A.1.3. MacWilliam's Identity

This is an alternate proof to Theorem 4.20.

Proof. By directing applying the two lemmas above:

$$\begin{aligned}\frac{1}{|\mathcal{C}|}A_{\mathcal{C}}(y-x,y+x) &= \frac{1}{2^k}(2x)^k(y-x)^{n-k}W_{\mathcal{C}}\left(\frac{y-x}{x},\frac{2x}{y-x}\right)\\ &= x^k(y-x)^{n-k}W_{\mathcal{C}^\perp}\left(\frac{2x}{y-x},\frac{y-x}{x}\right)\\ &= A_{\mathcal{C}^\perp}(x,y).\end{aligned}$$

It is thus proven. ■

A.2. Catalan Number

The Catalan numbers show up in numerous combinatorial problems. This section gives a brief introduction to the Catalan number, derive its general formula, and derive its generating function, which was introduced back in Section 6.2.3. This section fills in the details missing in the proof of

A.3. Majorization