CS3236 Lecture Notes #6: Practical Channel Codes

Jonathan Scarlett

April 10, 2023

Useful references:

- Blog post on Hamming codes[1]
- Cover/Thomas Section 7.11
- MacKay Sections 11.4–11.5, Chapters 13–14
- (Beyond the scope of this course) Survey article on coding,[2] and/or the advanced textbook "Modern Coding Theory" (Richardson and Urbanke)

1 Recap of Parity Checks and the Hamming Code

Parity check.

- A *parity check* of a sequence of bits $b_1, \dots, b_m$ is an additional bit equaling 1 if the number of 1's in $b_1 \dots, b_m$ is odd, and 0 if the number of 1's is even.
  - Hence, in either case, there is an even number of 1's in the sequence $b_1, \dots, b_m c$, where c is the parity check bit
- We can express this via modulo-2 arithmetic: For $a, b \in \{0, 1\}$, let the $\oplus$ operator be defined as

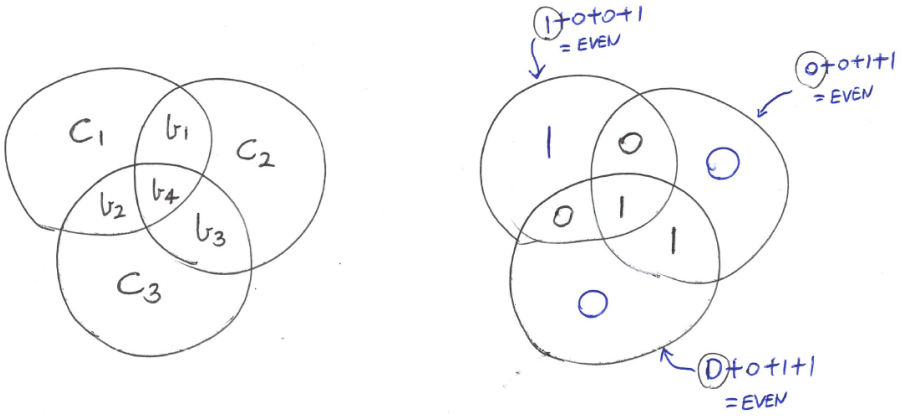
$$0 \oplus 0 = 1 \oplus 1 = 0$$
$$0 \oplus 1 = 1 \oplus 0 = 1.$$

  Then the parity check of $b_1, \dots, b_m$ is $c = b_1 \oplus \dots \oplus b_m$.
- If we transmit the length-$(m+1)$ sequence $b_1 \dots b_m c$ across a channel, and one of the bits is flipped, we will notice that the total number of 1's is no longer even. Hence, we can *detect* a single bit flip. However, a little thought reveals that we cannot *correct* it.
- The idea that permits error correction: *Send multiple parity checks applied to different groups of bits*

[1] https://jeremykun.com/2015/03/02/hammings-code/
[2] https://arxiv.org/pdf/1908.09903.pdf

Simple examples.

- In this first lecture, we introduced the *repetition code*, which (for example) encodes 1010 into 111000111000. By a majority vote rule, this permits the *correction* of one bit flip in each of the groups of 3 bits.
- We also introduced the Hamming code, which maps 4 bits to 7 bits and can correct a single bit flip:



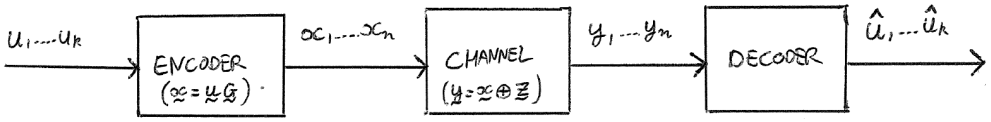
We described the check bits $c_1c_2c_3$ as being chosen to make the number of 1's per circle even. In the above terminology, each c_i is a *parity check* of the b_i's in the corresponding circle.

  - Note: When we talk about being able to correct a certain number of bit flips, this may include flips in both the uncoded bits b_i and the check bits c_i.
  - How to correct one bit flip: Observe that each possible single bit flip changes a unique combination of circles from "even number of 1s" to "odd number of 1s" (e.g., the middle bit changes all 3 circles; the top-left bit only changes the top-left circle). Therefore, the decoder can check which circles have an odd number of 1s, and un-flip the corresponding bit. If the decoder sees all 3 circles already having an even number of 1s, then no changes are made.

2 Linear Codes

Notation.

- We will now switch notation a little, and consider the general procedure of mapping k bits $\mathbf{u} = (u_1, \ldots, u_k)$ to n bits $\mathbf{x} = (x_1, \ldots, x_n)$, where $n \geq k$.
- We will mostly consider the transmission of $\mathbf{x} = (x_1, \ldots, x_n)$ across a binary symmetric channel (BSC) to produce $\mathbf{y} = (y_1, \ldots, y_n)$; these output bits are used to construct an estimate $\hat{\mathbf{u}} = (\hat{u}_1, \ldots, \hat{u}_k)$ of the original k bits.



The channel can be described as $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$, where $\mathbf{z} \in \{0,1\}^n$ indicates which bits are flipped, and the operator $\oplus$ is applied bit-by-bit.

- Notice that this is the channel coding setup of the previous lectures specialized to the BSC. The generic "message" $m \in \{1, \ldots, M\}$ is now replaced by "message bits" $(u_1, \ldots, u_k) \in \{0,1\}^k$, so that $M = 2^k$. Previously we defined the rate as $R = \frac{1}{n} \log_2 M$, and substituting $M = 2^k$ gives

$$R = \frac{k}{n}.$$

  This is intuitive: If we map k bits to $3k$ bits (say), then the rate is $\frac{1}{3}$.

Definition and generator matrix.

- For reasons to be made clear shortly, we say that any code comprised of parity checks is a *linear code*.
- We distinguish between the following two cases:
  - A **systematic parity-check code** is one in which the first k (out of n) bits of $\mathbf{x}$ are always precisely the original k bits, and the remaining $n-k$ bits are parity checks:

$$x_i = u_i, \quad i = 1, \ldots, k,$$
$$x_i = \bigoplus_{j=1}^{k} u_j g_{j,i}, \quad i = k+1, \ldots, n$$

    where $g_{j,i} = 1$ if the parity check in location i includes u_j, and $g_{j,i} = 0$ otherwise. For instance, the Hamming code described above is a systematic parity-check code.
  - A **general parity-check code** is one in which all n codeword bits may be arbitrary parity checks:

$$x_i = \bigoplus_{j=1}^{k} u_j g_{j,i}, \quad i = 1, \ldots, n.$$

    Clearly a systematic code is a special case of this, since it corresponds to setting $g_{j,i} = \mathbf{1}\{j = i\}$ for $i = 1, \ldots, k$.
- The above (general) formula for generating each x_i from $u_1, \ldots, u_k$ can be succinctly summarized as a (modulo-2) vector-matrix multiplication:

$$\mathbf{x} = \mathbf{uG}, \quad \text{(in modulo-2 arithmetic)}$$

  where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{u} = (u_1, \ldots, u_k)$ are the suitable row vectors, and

$$\mathbf{G} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{bmatrix}$$

  is known as the *generator matrix*.
  - Interpretation: The 1's in each column indicate which bits are included in the parity check

- In the special case of a systematic code, this simplifies to

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & \dots & 0 & g_{1,k+1} & \dots & g_{1,n} \\ 0 & 1 & \dots & 0 & g_{2,k+1} & \dots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & g_{k,k+1} & \dots & g_{k,n} \end{bmatrix}$$

  with the left-most $k \times k$ sub-matrix being the identity matrix.

- With the mapping from $\mathbf{u}$ to $\mathbf{x}$ being described by the matrix multiplication $\mathbf{x} = \mathbf{uG}$ (in modulo-2 arithmetic), we can now justify the terminology *linear code*: If $\mathbf{u}$ and $\mathbf{u}'$ are two different message sequences, and their corresponding codewords are $\mathbf{x} = \mathbf{uG}$ and $\mathbf{x}' = \mathbf{uG}'$, then

$$\begin{aligned} \mathbf{x} \oplus \mathbf{x}' &= \mathbf{uG} \oplus \mathbf{u}'\mathbf{G} \\ &= (\mathbf{u} \oplus \mathbf{u}')\mathbf{G}, \end{aligned}$$

  which means that $\mathbf{x} \oplus \mathbf{x}'$ is also a codeword (corresponding to message $\mathbf{u} \oplus \mathbf{u}'$). In other words, the (modulo-2) sum of any two valid codewords is another valid codeword.

  – Note: We have extended the $\oplus$ notation to vectors/sequences, which is done via a bit-by-bit application of the definition above, e.g., $0001 \oplus 1101 = 1100$ and $101 \oplus 010 = 111$.

- **Examples.** With $k = 4$, the generator matrices for the single-parity-check code and Hamming code (described at the start of the lecture) are given by

$$\mathbf{G}_{\text{parity}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \qquad \mathbf{G}_{\text{Hamming}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1}$$

  Sometimes you might see the latter with the last 3 columns in a different order, but re-ordering columns just amounts to re-labeling the indices of parity check bits.

  – Pre-multiplying $\mathbf{G}_{\text{Hamming}}$ by all 16 possible $\mathbf{u}$ sequences, we can list all the codewords of the Hamming code:

$$\begin{array}{cccc} 0000000 & 0001111 & 0010011 & 0011100 \\ 0100101 & 0101010 & 0110110 & 0111001 \\ 1000110 & 1001001 & 1010101 & 1011010 \\ 1100011 & 1101100 & 1110000 & 1111111. \end{array}$$

  – For instance, the codeword 1100011 is obtained from $\mathbf{u} = 1100$ by taking the modulo-2 sum of the first two rows of $\mathbf{G}_{\text{Hamming}}$. For $\mathbf{u} = 1111$, we take the modulo-2 sum of all 4 rows.

- As we will see in the tutorial, any general code can be reduced to an "equivalent" systematic code by applying a technique similar to Gaussian elimination on $\mathbf{G}$.

Parity-check matrix.

- A matrix closely related to $\mathbf{G}$, but which will be more directly useful when it comes to decoding, is called the *parity-check matrix* $\mathbf{H}$. It is an $n \times (n-k)$ matrix that satisfies

$$\mathbf{xH} = \mathbf{0} \iff \mathbf{x} \text{ is a valid codeword.}$$

  Notice the distinction:

  - $\mathbf{G}$ is used to *generate* $\mathbf{x}$ from $\mathbf{u}$;
  - $\mathbf{H}$ is used to *check* if $\mathbf{x}$ can be generated from *any* $\mathbf{u}$ (doing this naively using $\mathbf{G}$ by testing all 2^k possible $\mathbf{u}$ would be grossly inefficient).

- While such check matrices exist for all generator matrices, we will focus our attention on the systematic case, as it is much simpler. Recall the two formulas we gave for x_i (in terms of the $\{u_j\}$ and $\{g_{j,i}\}$) in the systematic case; substituting the first into the second gives

$$x_i = \bigoplus_{j=1}^{k} x_j g_{j,i}, \quad i = k+1, \ldots, n.$$

  Adding x_i (modulo 2) to both sides, the left-hand side gives $x_i \oplus x_i = 0$, and we are left with

$$\left(\bigoplus_{j=1}^{k} x_j g_{j,i} \right) \oplus x_i = 0, \quad i = k+1, \ldots, n.$$

  Converting to matrix form reveals that the following choice of $\mathbf{H}$ indeed gives $\mathbf{xH} = \mathbf{0}$:

$$\mathbf{H} = \begin{bmatrix} g_{1,k+1} & g_{1,k+2} & \cdots & g_{1,n} \\ g_{2,k+1} & g_{2,k+2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,k+1} & g_{k,k+2} & \cdots & g_{k,n} \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}.$$

  It is also not hard to establish the opposite, i.e., if $\mathbf{xH} = \mathbf{0}$ then $\mathbf{x}$ is indeed a valid codeword.

- Stated more succinctly, we have

$$\mathbf{G} = [\mathbf{I}_k \;\; \mathbf{P}] \quad \Longrightarrow \quad \mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix},$$

  where $\mathbf{I}_m$ is the $m \times m$ identity matrix, and $\mathbf{P}$ is the remaining $k \times (n-k)$ submatrix of $\mathbf{G}$.

- **Examples.** The check matrices corresponding to (1) are

$$\mathbf{H}_{\text{parity}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \qquad \mathbf{H}_{\text{Hamming}} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

- To give a flavor of why the check matrix is useful for decoding, notice that if $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$ with $\mathbf{z}$ indicating which bits got flipped, then

$$\begin{aligned} \mathbf{yH} &= (\mathbf{x} \oplus \mathbf{z})\mathbf{H} \\ &= \big(\mathbf{xH}\big) \oplus \big(\mathbf{zH}\big) \\ &= \mathbf{zH}, \end{aligned} \tag{2}$$

  where we first used linearity, and then the fact that $\mathbf{xH} = \mathbf{0}$ for any valid codeword $\mathbf{x}$.

  - In particular, if $\mathbf{z}$ only contains a single 1 (i.e., only one bit got flipped), then $\mathbf{yH}$ is simply the i-th row of $\mathbf{H}$, where i is the index of the flipped bit.
  - But notice that in $\mathbf{H}_{\text{Hamming}}$, all the rows are distinct! This means that by looking at $\mathbf{H}_{\text{Hamming}}$, we can immediately identify which bit got flipped and therefore correct it.

Notes on storage and computation.

- Notice that the code is fully specified by $\mathbf{G}$ (or $\mathbf{H}$), which only requires nk bits of storage – much smaller than the exponential storage requirement for the random coding technique used to prove the channel coding theorem!
- However, efficient decoding (getting from the noisy channel output $\mathbf{y}$ back to $\mathbf{u}$) is still challenging, designing a good choice of $\mathbf{G}$ is also difficult (but do-able!).
- In fact, at this stage it is unclear whether any good choices of G exist! This is addressed in Chapter 14 of MacKay's book (optional reading), where it is shown that a *random generator matrix* G (i.e., each entry is 1 or 0 with equal probability) achieves arbitrarily small error probability on the BSC at all rates below capacity, when used in conjunction with joint typicality decoding. The proof is very similar to the standard (non-linear) random coding proof. In summary, **very good linear codes exist**, at least if we ignore computational constraints on the decoder.

3 Distance Properties

We will only touch on the basics of distance properties here; if you are interested in knowing more, see Chapter 13 of MacKay's book.

Definition and properties.

- **Definition 1.** The *Hamming distance* between two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ an $\mathbf{x}' = (x'_1, \ldots, x'_n)$ (having the same length n) is the number of positions in which they differ:

$$d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \mathbb{1}\{x_i \neq x'_i\}.$$

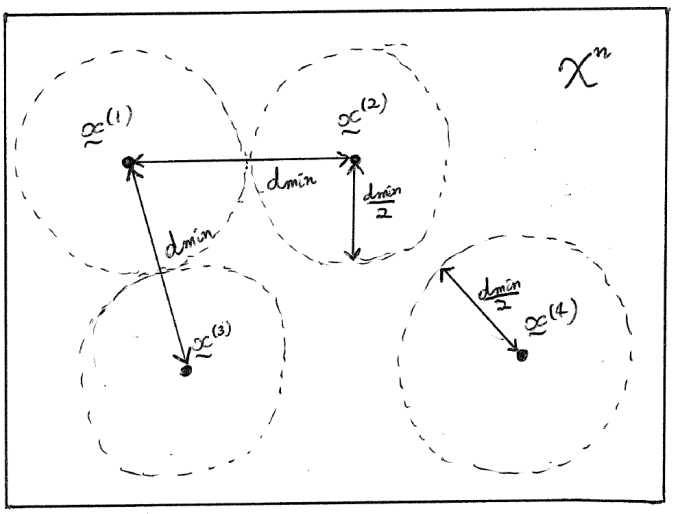
  For instance, the Hamming distance between 00110011 and 00010111 is 2.

- **Definition 2.** The *minimum distance* of a codebook $\mathcal{C}$ of length-n codewords is

$$d_{\min} = \min_{\mathbf{x} \in \mathcal{C}, \mathbf{x}' \in \mathcal{C} \,:\, \mathbf{x} \neq \mathbf{x}'} d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}').$$

  Intuitively, we should expect higher $d_{\min}$ to mean better robustness to noise in the channel.

- The following illustration shows that if the minimum distance is $d_{\min}$, then (at least given enough computation time) it is possible to *correct up to* $d_{\min} - 1$ *erasures* and *correct up to* $\frac{d_{\min}-1}{2}$ *bit flips*:



  - For correcting erasures: Simply note that if $d_{\min} - 1$ code symbols are replaced by an erasure symbol '?', then there is only one way to fill them in to get a valid codeword (otherwise, the minimum distance would be $d_{\min} - 1$ or less!)
  - For correcting flips: Note that the balls of radius $\frac{d_{\min}-1}{2}$ around each codeword cannot overlap. This means that if we decode each $\mathbf{y}$ to the codeword in its closest ball, we will always be correct if at most $\frac{d_{\min}-1}{2}$ flips occurred.

- **Claim.** If $\mathcal{C}$ is the set of codewords formed by a given linear code with $d_{\min} > 0$,[3] then

$$d_{\min} = \min_{\mathbf{x} \in \mathcal{C} \,:\, \mathbf{x} \neq \mathbf{0}} w(\mathbf{x}),$$

  where $w(\mathbf{x}) = \sum_{i=1}^{n} \mathbb{1}\{x_i = 1\}$ is the *weight* of the codeword $\mathbf{x}$. Hence, for linear codes, the minimum distance equals the minimum weight.

[3] A code with $d_{\min} = 0$ is a terrible idea – it means two different vectors of information bits $\mathbf{u}, \mathbf{u}'$ lead to the same codeword!

– Proof: Let $\mathbf{x}', \mathbf{x}''$ be the two codewords at a minimum distance from each other. Then by linearity, $\mathbf{x}' \oplus \mathbf{x}''$ is also a valid codeword, and its weight is precisely $d_{\mathrm{H}}(\mathbf{x}', \mathbf{x}'') = d_{\min}$. In addition, the assumption $d_{\min} > 0$ implies that it is not the all-zero codeword.

Conversely, since $\mathbf{x} = \mathbf{0}$ is always a valid codeword (corresponding to $\mathbf{u} = \mathbf{0}$), any codeword weight also corresponds to a distance (to the all-zero codeword).

- **Example.** Recall the 16 codewords we listed earlier for the Hamming code. The minimum non-zero weight (and hence minimum distance) is 3, which is consistent with the fact that the Hamming code can correct $\frac{3-1}{2} = 1$ bit flip.

(Optional) Distance vs. capacity.

- While a high minimum distance seems like a nice property to have, it corresponds to a fundamentally different modeling assumption compared to the channel capacity:
  - The minimum distance goal is aligned with *worst-case errors*, where (for instance) we need to be able to correct *arbitrary patterns* of up to δn bit flips introduced by the channel.
  - The channel capacity goal is aligned with *random errors*, where (for instance) we need to be able to correct bit flips that occur *independently* with probability δ each.

  Neither of these goals should be viewed as "better" than the other in general; either may be preferable depending on the application.

- Of course, it is worth asking whether achieving capacity and attaining good distance properties are actually equivalent goals. However, as argued in Chapter 13 of MacKay's book, this is not the case:
  - The best possible rate with minimum distance δn is at most the channel capacity with *double* the noise level, 2δ;
  - The channel capacity is positive for any $\delta \in \left(0, \frac{1}{2}\right)$, but no positive rate can be achieved for $\delta > \frac{1}{4}$ if one insists on a minimum distance δn;
  - Examples are known where the channel capacity is achieved despite a "bad" minimum distance.

- Although distinct from achieving capacity, the design of codes with good distance properties is a very important problem in its own right. Examples of such codes include BCH codes (a generalization of Hamming codes), and Reed-Solomon codes (commonly used for CDs and DVDs). These are based on more advanced mathematical algebraic methods, namely, polynomials on finite fields.

4 Minimum Distance Decoding

Maximum-likelihood decoding over general channels.

- Recall that in the general channel coding setup, the message m is uniform on $\{1, \ldots, M\}$, the encoder maps each m to a codeword $\mathbf{x}^{(m)}$, the channel produces $\mathbf{y}$, and the error probability is given by

$$P_{\mathrm{e}} = \mathbb{P}[\hat{m} \neq m]. \tag{3}$$

  where the probability is with respect to both the randomly-chosen message and the channel.

– Note: Below, we will sometimes use the message notation $m \in \{1, \ldots, M\}$ for linear codes even though it is perhaps more natural to think of the information bits $\mathbf{u} \in \{0,1\}^k$. A one-to-one correspondence between the two can be formed by considering $M = 2^k$.

- **Theorem.** For any channel $P_{\mathbf{Y}|\mathbf{X}}$ and any codebook $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$, the decoding rule that minimizes the error probability P_{e} is the maximum-likelihood (ML) decoder:

$$\hat{m} = \arg\max_{j=1,\ldots,M} P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}).$$

  – Proof outline: It is a standard result in Bayesian probability that the optimal estimate is the *maximum a posteriori* (MAP) estimate $\arg\max_{j=1,\ldots,M} P_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}^{(j)}|\mathbf{y})$. But by Bayes' rule, we have $P_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}^{(j)}|\mathbf{y}) = \frac{P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}) P_{\mathbf{X}}(\mathbf{x}^{(j)})}{P_{\mathbf{Y}}(\mathbf{y})}$. Since neither $P_{\mathbf{X}}(\mathbf{x}^{(j)})$ nor $P_{\mathbf{Y}}(\mathbf{y})$ depend on j (the former being because we assumed a uniformly chosen message, so all of the prior probabilities are $\frac{1}{M}$), the MAP rule is equivalent to the ML rule maximizing $P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)})$.

  – (Note: Strictly speaking, the above outline implicitly assumes that all the codewords are distinct, but the result holds more generally, and such an assumption is very mild anyway)

Application to the BSC – Minimum distance decoding.

- For a memoryless channel, we have $P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^n P_{Y|X}(y_i|x_i)$, or equivalently $\log P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^n \log P_{Y|X}(y_i|x_i)$. For the binary symmetric channel (BSC), $P_{Y|X}(y|x_i)$ is equal to $1-\delta$ if $y = x_i$, and δ otherwise. Hence,

$$\begin{aligned}
\log P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) &= \sum_{i=1}^n \log P_{Y|X}(y_i|x_i) \\
&= \sum_{i\,:\,y_i = x_i} \log(1-\delta) + \sum_{i\,:\,y_i \neq x_i} \log \delta \\
&= (n - d_{\mathrm{H}}(\mathbf{x},\mathbf{y})) \log(1-\delta) + d_{\mathrm{H}}(\mathbf{x},\mathbf{y}) \log \delta \\
&= n \log(1-\delta) - d_{\mathrm{H}}(\mathbf{x},\mathbf{y}) \log \frac{1-\delta}{\delta}.
\end{aligned}$$

  Assuming that $\delta < \frac{1}{2}$, we have $\log \frac{1-\delta}{\delta} > 0$, and we conclude that

$$\arg\max_{j=1,\ldots,M} P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}^{(j)}) = \arg\min_{j=1,\ldots,M} d_{\mathrm{H}}(\mathbf{x}^{(j)}, \mathbf{y}).$$

  That is, maximum-likelihood decoding is equivalent to *minimum (Hamming) distance* decoding.

Application to *linear codes* for the BSC – Syndrome decoding.

- We repeat the evaluation of $\mathbf{yH}$ given above:

$$\begin{aligned}
\mathbf{yH} &= (\mathbf{x} \oplus \mathbf{z})\mathbf{H} \\
&= \big(\mathbf{xH}\big) \oplus \big(\mathbf{zH}\big) \\
&= \mathbf{zH}, \qquad (4)
\end{aligned}$$

  with the middle step using the fact that $\mathbf{xH} = \mathbf{0}$ for any valid codeword $\mathbf{x}$.

- The quantity

$$\mathbf{S} = \mathbf{z}\mathbf{H} = \mathbf{y}\mathbf{H}$$

is called the *syndrome*, and we have just shown that it can immediately be computed from the check matrix $\mathbf{H}$ given the channel output $\mathbf{y}$.

- **Claim.** For a linear code, if the syndrome is $\mathbf{S}$, then the minimum-distance codeword to $\mathbf{y}$ can be obtained by finding

$$\hat{\mathbf{z}} = \underset{\tilde{\mathbf{z}}\,:\,\tilde{\mathbf{z}}\mathbf{H}=\mathbf{S}}{\arg\min}\, w(\tilde{\mathbf{z}})$$

(where $w(\tilde{\mathbf{z}})$ is the number of 1's in $\tilde{\mathbf{z}}$) and then computing

$$\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}.$$

Provided that the code is systematic, the estimate $\hat{\mathbf{u}}$ of the original information bits is then formed by taking the first k entries of $\hat{\mathbf{x}}$.

  - Proof: Minimum-distance decoding searches for the codeword $\mathbf{x}^{(j)}$ $(j = 1, \ldots, M)$ having the fewest disagreements with $\mathbf{y}$. For each codeword $\mathbf{x}^{(j)}$, we can define

$$\mathbf{z}^{(j)} = \mathbf{x}^{(j)} \oplus \mathbf{y}$$

and this clearly yields $w(\mathbf{z}^{(j)}) = d_{\mathrm{H}}(\mathbf{x}^{(j)}, \mathbf{y})$. Hence, the minimum-distance codeword is the one indexed by

$$\hat{m} = \underset{j=1,\ldots,M}{\arg\min}\, w(\mathbf{z}^{(j)}).$$

The claim now follows since

$$\begin{aligned}
\left\{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}\right\} &\overset{(a)}{=} \left\{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}}\mathbf{H} = \mathbf{y}\mathbf{H}\right\} \\
&\overset{(b)}{=} \left\{\tilde{\mathbf{z}} \,:\, (\tilde{\mathbf{z}} \oplus \mathbf{y})\mathbf{H} = \mathbf{0}\right\} \\
&\overset{(c)}{=} \left\{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}} \oplus \mathbf{y} \text{ is a valid codeword}\right\} \\
&\overset{(d)}{=} \left\{\mathbf{x}^{(j)} \oplus \mathbf{y} \,:\, j = 1, \ldots, M\right\} \\
&\overset{(e)}{=} \left\{\mathbf{z}^{(j)} \,:\, j = 1, \ldots, M\right\}
\end{aligned}$$

where (a) uses $\mathbf{S} = \mathbf{y}\mathbf{H}$, (b) uses the definition of $\oplus$ and linearity, (c) uses the fact that $\mathbf{x}$ is a valid codeword if and only if $\mathbf{x}\mathbf{H} = \mathbf{0}$, (d) uses the fact that the valid codewords are $\{\mathbf{x}^{(j)}\}_{j=1}^{M}$ and the fact that $\mathbf{x}^{(j)} = \tilde{\mathbf{z}} \oplus \mathbf{y}$ is equivalent to $\tilde{\mathbf{z}} = \mathbf{x}^{(j)} \oplus \mathbf{y}$, and (e) uses $\mathbf{z}^{(j)} = \mathbf{x}^{(j)} \oplus \mathbf{y}$.

- The equivalent optimization formulation in the claim can help us reduce the computational complexity of minimum-distance decoding:
  - If $\mathbf{S} = \mathbf{0}$, then simply output $\hat{\mathbf{x}} = \mathbf{y}$;
  - Otherwise, look for a $\tilde{\mathbf{z}}$ of weight 1 such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$, and if one is found, output $\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}$;
  - Otherwise, look for a $\tilde{\mathbf{z}}$ of weight 2 such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$, and if one is found, output $\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}$;
  - etc.

If very few bit flips occurred during transmission, then we may avoid searching over an exponentially large number of codewords – there are only n vectors $\tilde{\mathbf{z}} \in \{0,1\}^n$ of weight 1, $\binom{n}{2}$ of weight 2, and so on. However, if a significant number of bit flips occur during transmission, this decoding method still typically requires far too much computation to be practical.

- **Example.** For the Hamming code (see the check matrix $\mathbf{H}_{\text{Hamming}}$ above), we could traverse the following list (starting from the top) until we find a $\tilde{\mathbf{z}}$ such that $\tilde{\mathbf{z}}\mathbf{H} = \mathbf{S}$:
  - $\tilde{\mathbf{z}} = 0000000 \implies \tilde{\mathbf{z}}\mathbf{H} = 000$
  - $\tilde{\mathbf{z}} = 1000000 \implies \tilde{\mathbf{z}}\mathbf{H} = 110$
  - $\tilde{\mathbf{z}} = 0100000 \implies \tilde{\mathbf{z}}\mathbf{H} = 101$
  - $\tilde{\mathbf{z}} = 0010000 \implies \tilde{\mathbf{z}}\mathbf{H} = 011$
  - $\tilde{\mathbf{z}} = 0001000 \implies \tilde{\mathbf{z}}\mathbf{H} = 111$
  - $\tilde{\mathbf{z}} = 0000100 \implies \tilde{\mathbf{z}}\mathbf{H} = 100$
  - $\tilde{\mathbf{z}} = 0000010 \implies \tilde{\mathbf{z}}\mathbf{H} = 010$
  - $\tilde{\mathbf{z}} = 0000001 \implies \tilde{\mathbf{z}}\mathbf{H} = 001$
  - $\tilde{\mathbf{z}} = 1100000 \implies \ldots$ (actually, in this example this case would never be reached, since the previous cases already cover all 8 possible syndromes)

  If only one bit flip occurred, the resulting $\mathbf{y} \oplus \tilde{\mathbf{z}}$ will correctly recover the transmitted codeword.
- We will not cover more advanced decoding techniques in this course; some relevant concepts are very briefly introduced in the next (optional) section, and a comprehensive reference is the book "Modern Coding Theory". MacKay's book is also a very good reference, whereas Cover/Thomas focuses almost entirely on fundamental limits rather than practical codes.

5 (Optional) Advanced Coding Methods

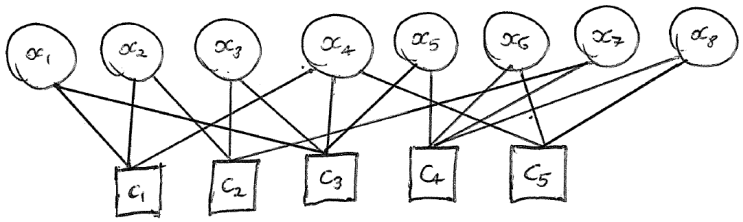
History.

- The channel coding theorem was published in 1948. This spawned decades of research on practical coding techniques, but none were seen to come close to achieving capacity for a long time.
- In the early 1990s, *turbo codes* were (empirically) seen to be very close to achieving capacity.
- *Low density parity check* (LDPC) codes were invented in the 1960s, but perhaps due to limited computing power, they remained unused for a long time. After the invention of turbo codes, they were rediscovered and also shown to be nearly capacity-achieving.
- *Polar codes* are a more recent development (2008). Although they usually don't perform as well as turbo/LDPC codes in practice, they are much nicer to analyze theoretically, in particular to rigorously establish that they are capacity-achieving.

(Very) Brief overview of LDPC codes.

- As a rough definition, an LDPC code is a (typically non-systematic) linear code such that the check matrix $\mathbf{H}$ contains mostly 0's and relatively few 1's.

- We can picture this via a *factor graph* representation:



  Here circles correspond to the n codeword bits ("variable nodes"), and squares correspond to the $n-k$ parity equations ("check nodes"), i.e., if there are no errors, there should be an even number of 1's connected to each square. The *low density* assumption amounts to saying that the above graph is *sparse* (has few edges).

- The reason sparsity / low density is useful is that it permits effective *belief propagation* decoding, in which information is shared between variable nodes and check nodes until accurate beliefs (posterior probability estimates) on the transmitted bits are obtained.

- Strictly speaking, this is only guaranteed to end up with accurate posterior estimates when the underlying graph has no cycles (i.e., one can never follow any path of edges and end up back at the starting node). Although sparse graphs like the ones above do have cycles, they have *very few short cycles*, and for practical purposes this is often nearly as good as having no cycles.

- For a much better introduction to LDPC codes, see Chapter 47 of MacKay's book (see also `https://www.youtube.com/watch?v=RWUxtGh-guY` for a brief introduction in Layman's terms).

(Very) Brief overview of polar codes.

- The rough idea of polar codes is as follows:
  - First consider 2 uses of the channel, $(X_1, X_2) \to (Y_1, Y_2)$.
  - The idea: Apply some pre-processing to (X_1, X_2) to create a related channel $(\widetilde{X}_1, \widetilde{X}_2) \to (Y_1, Y_2)$ such that (i) $I(X_1, X_2; Y_1, Y_2) = I(\widetilde{X}_1, \widetilde{X}_2; Y_1, Y_2)$ (no information is lost); (ii) Comparing the new channel to the original channel, one "channel use" is less noisy, and the other is more noisy.
  - Repeat this procedure several times, moving from 2 to 4 uses of the channel, then 8 uses of the channel, then 16, etc., with many "channel uses" getting less and less noisy, and the rest getting more and more noisy.
  - *For a symmetric binary channel with capacity $C \in (0,1)$, repeating this procedure eventually leads to a fraction C of "near-perfect" channel uses, and a fraction $1-C$ of "near-useless" channel uses.*
  - Perfect and useless channels are very easy to handle! If it were truly perfect, we could just send a bit and receive it without noise. If it were truly useless, we would simply avoid using it at all. Polar codes do something similar, sending information only over the "near-perfect" uses.

- Extensions to non-binary and non-symmetric channels were established later.

- For a much better introduction to polar codes, see the following lecture by Emre Telatar: `https://www.youtube.com/watch?v=VhyoZSB9g0w`