ECE 361: Lecture 12: Rate-Efficient Communication – Part III

12.1. Channel Capacity

12.1.1. Linear binary codes and binary modulation

In Lecture 11, we considered coded communication systems using binary codes of block length n and rate R and binary modulation, and found that averaged over the ensemble $\mathfrak{C}$ of all possible binary codes, or over the ensemble $\mathfrak{C}_L$ of all possible linear codes, the *average* error probability $\overline{P(E)}$ satisfies

$$\overline{P(E)} < 2^{n(R-R^*)} \tag{12.1}$$

where

$$R^* = 1 - \log_2(1 + \exp(-\mathsf{SNR}/2)) \text{ bits per channel use.} \tag{12.2}$$

Here, $\mathsf{SNR} = \mathcal{E}/\sigma^2$ where $\mathcal{E}$ is the energy received per channel use and σ^2 is the noise variance, and R^* is called the channel capacity. We noted that $R^* > 0$ for all $\mathsf{SNR} > 0$, and approaches 1 bit per channel use as $\mathsf{SNR} \to \infty$. Also, the right side of (12.1) is smaller than 1 if $R < R^*$ and can be made as small as we desire by choosing n to be suitably large. Thus, the average error probability of long linear codes is small provided that $R < R^*$. How large n must be to get a suitably small error probability is determined by how close the rate R is to R^*. If we wish to push the envelope and operate at rate R close to R^*, n must be larger than if we were more easy-going and chose to operate at rate R that is not so close to R^*. Large values of n increase the complexity and cost (and the delay) of the system, and this is the price to be paid for high performance at rates close to R^*. Now, at least one code in the ensemble has error probability no larger than $\overline{P(E)}$, and hence no larger than the right side of (12.1), and thus there exist linear binary codes of large block length n that allow the transmission of data with high reliability over the channel at all rates $R < R^* = 1 - \log_2(1 + \exp(-\mathsf{SNR}/2))$. Since $n\mathcal{E}$ is used to transmit nR bits so that the energy per bit is $\mathcal{E}/R$ is fixed and not dependent on n, linear binary coding and binary modulation *can* be used to devise a rate-efficient and energy-efficient scheme for communication over the discrete-time Gaussian channel. However, *which* of the 2^{n^2R} linear binary codes of block length n and rate R can provide such excellent performance is, unfortunately, not specified in this theory. Clearly, there must be many good codes because the Markov Inequality

$$P\{\mathbb{X} \geq \alpha\} \leq \mathsf{E}[\mathbb{X}]/\alpha \;\Leftrightarrow\; P\{\mathbb{X} < \alpha\} \geq 1 - \mathsf{E}[\mathbb{X}]/\alpha \;\Leftrightarrow\; P\{\mathbb{X} < \beta\mathsf{E}[\mathbb{X}]\} \geq 1 - \frac{1}{\beta}$$

assures us that if we simply pick a code at random from the ensemble $\mathfrak{C}_L$ of all linear codes, then there is an even (or better) chance that we will get a code $\mathcal{C}$ with error probability $P(E_{\mathcal{C}}) < 2\overline{P(E)}$. Now, we have chosen n so that $2^{n(R-R^*)}$ is suitably small and it might well be $2\overline{P(E)} < 2^{n(R-R^*)}$ also, in which case we have found a good code that meets our requirements. If not, as suggested in Lecture 10, we could discard $\mathcal{C}$ and pick a code at random again from $\mathfrak{C}_L$. If $2^{n(R-R^*)} = \beta\overline{P(E)}$ where $\beta > 1$, then randomly selecting a code from $\mathfrak{C}_L$ gives a good code (i.e. one for which $P(E_{\mathcal{C}}) < 2^{n(R-R^*)}$) with probability $1 - \beta^{-1}$ and thus on average, only $\beta/(\beta - 1)$ tries are required to find a good code. Thus, in a certain sense, it is not too hard to find a good code. The difficulty is that implementing the (maximum-likelihood) receiver for such a code is prohibitively expensive. Thus, there has been considerable research in designing codes that can be decoded relatively inexpensively with *suboptimum* receivers, and yet still allow us to communicate reliably over the channel at rates not too far from the capacity. As might have been expected, the search was none too successful. The codes that were found can operate only at rates considerably below capacity. However, some recent developments over the past fifteen years (described briefly in Section 12.2) have resulted in decoding methods for a particular class codes that allow for the design of communication systems operating at rates close to capacity.

12.1.2. Capacity with an average power constraint

The results described in Section 12.1.1. are for Gaussian channels in which binary codes are used with binary modulation. There is an implicit assumption in these results of a peak-power limitation on the channel. We also noted in Lecture 11 that if we have an *average power constraint* on the channel, *viz.* the *total energy* over n channel uses must not exceed $n\mathcal{E}$ though there is not a peak power limitation on any particular channel use, and we do not restrict the transmitter to use only binary modulation, then the discrete-time Gaussian channel has a larger capacity

$$C = \frac{1}{2}\log_2(1+\mathsf{SNR}) \text{ bits per channel use.} \tag{12.3}$$

Notice that as SNR increases, the channel capacity increases without bound rather than approaching a limiting value of one bit per channel use as in Section 12.1.1. Thus, the laxer average power constraint allows for higher rates of communication over the discrete-time Gaussian channel. However, this increase comes at a heavy price since the capacity increases only *logarithmically* with SNR. We have the following:

$$\text{If } \mathsf{SNR} \gg 1,\ C \approx \frac{1}{2}\log_2(\mathsf{SNR}), \text{ that is, } \mathsf{SNR} \approx 4^C.$$

Thus, the already large SNR must be *quadupled* to increase capacity by one bit per channel use.[1] Communication systems with large SNR use codes over large alphabets so as to able to transmit more than one bit per channel use. In contrast, binary modulation suffices at low SNR. Using the fact that $\ln(1+x) \approx x$ for small x, we have:

$$\text{If } \mathsf{SNR} \ll 1,\ C \approx \frac{1}{2\ln 2}\mathsf{SNR} = \frac{1}{2}\log_2(e)\mathsf{SNR}$$

and thus the capacity increases linearly with SNR when SNR is small. When SNR is low, the capacity R^* with binary modulation is $R^* \approx \frac{1}{4}\log_2(e)\mathsf{SNR} \approx C/2$ and also increases linearly with SNR.

12.2. Good Binary Codes

12.2.1. The Decoding Problem

Maximum-likelihood receivers are expensive to implement for binary (n,k) codes when k is large. The 2^k likelihoods that need to be computed require too many processors if they are computed in parallel, and too much time if they are computed sequentially on a single processor. It is also true that much of the computation is essentially wasted in the sense that the receiver is not interested in the actual numerical values of the likelihoods but only in which of the likelihoods is the largest. Now, if the code were known to have some *structure* to it from which the receiver could *deduce* from a given received vector $\underline{\mathbb{Y}}$ that only a few codewords could possibly have the largest likelihood, then it would suffice to compute the likelihoods for those few codewords only, and choose the codeword with the largest likelihood from among the few codewords. For example, we know that the codeword with the maximum likelihood is the one that is nearest in Euclidean distance to $\underline{\mathbb{Y}}$. Perhaps many codewords could be eliminated from consideration because they are "obviously" much farther away from $\underline{\mathbb{Y}}$ than a few others? The word *obviously* is in quotes in the previous sentence because the idea here is that this obviousness should be apparent with *far fewer* computations than computing the likelihoods of these many codewords and then discarding them. Of course, when the code is chosen by random selection from $\mathfrak{C}_L$, there is no guarantee that any such structure would exist in the code selected, or if structure did indeed exist, that the communications engineer – with a deadline looming – would be able to discover and exploit such structure by incorporating it into the receiver design in time to meet the delivery date. Thus, considerable effort was expended in finding codes with structure for which the receiver could eliminate many codewords from consideration based on computations far less onerous than actually computing the many likelihoods.

[1] We saw a similar result in Lecture 1 for a different channel model.

12.2.2. Decoding Algorithms

There is a huge literature describing a wide variety of ingenious and elegant designs for codes that people have devised and the *decoding algorithms* used to find the most likely transmitted codeword. In order to simplify the receiver computational burden, many of these designed coding schemes first *demodulate* each received symbol $\mathbb{Y}_i$ in to a bit $\hat{\mathbf{z}}_i \in \{0, 1\}$, that is, $\underline{\mathbb{Y}}$ is converted into a binary vector $\hat{\mathbf{z}}$ which is then processed by the decoding algorithm. This in effect parallels the decomposition (referred to in Lecture 10) of the transmitter into an *encoder* that maps the k-bit data vector $\mathbf{u}$ into the n-bit codeword $\mathbf{x}$ and a *modulator* that transforms $\mathbf{x}$ into a vector $\mathbf{X}$ of $\pm\sqrt{\mathcal{E}}$ symbols for transmission. At the receiver, we have a *demodulator* that maps $\underline{\mathbb{Y}}$ into $\hat{\mathbf{z}}$ followed by a *decoder* that processes $\hat{\mathbf{z}}$ to produce a most likely transmitted codeword $\hat{\mathbf{x}}$ and most likely transmitted data vector $\hat{\mathbf{u}}$. For almost all such codes, the decoding algorithms decode $\hat{\mathbf{z}}$ into a codeword $\mathbf{x}^{(\ell)}$ only if $d_H(\hat{\mathbf{z}}, \mathbf{x}^{(\ell)}) \leq \lfloor (d_{\min} - 1)/2 \rfloor$ where

$$d_{\min} = \min_{i \neq j} d_H(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

is the *minimum Hamming distance* between two different codewords in the code. The *triangle inequality*

$$d_H(\mathbf{x}^{(i)}, \hat{\mathbf{z}}) + d_H(\hat{\mathbf{z}}, \mathbf{x}^{(j)}) \geq d_H(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \geq d_{\min}$$

ensures that $\hat{\mathbf{z}}$ cannot be at distance $\lfloor (d_{\min} - 1)/2 \rfloor$ or less from two different codewords. If there is a codeword at distance $\leq \lfloor (d_{\min} - 1)/2 \rfloor$ from $\hat{\mathbf{z}}$, then that codeword is unique. On the hand, there may not be any such codeword, that is, $d_H(\hat{\mathbf{z}}, \mathbf{x}^{(j)}) > \lfloor (d_{\min} - 1)/2 \rfloor$ for *all* j, $0 \leq j \leq 2^k - 1$. In such cases, the decoding algorithm *fails to decode* and cannot determine the most likely transmitted codeword. Such decoding algorithms are said to be *bounded-distance* decoding algorithms: they decode $\hat{\mathbf{z}}$ (and thus $\underline{\mathbb{Y}}$) into the most likely transmitted codeword only if $\hat{\mathbf{z}}$ is within a bounded distance of that codeword. Note in comparison that a maximum-likelihood receiver *always* decodes $\underline{\mathbb{Y}}$ into the *most* likely (that is, nearest) codeword: failure is not an option, even in cases when $\underline{\mathbb{Y}}$ is at nearly equal distance from multiple codewords. While the maximum-likelihood receiver *does* make a decision even in such cases, it is clear that the decision is not very reliable, and the (conditional) probability of error is quite high. In contrast, bounded-distance decoders have quite small error probability – when they decode, their decisions are correct with high probability – but they have to contend with a significant *probability of decoding failure*.

Decoding failures are handled in two different ways. In some cases, when a *reverse channel* (sometimes called a *feedback channel* is available on which the receiver can send messages to the transmitter, the receiver can ask the transmitter to repeat the transmission. This is the approach taken in data transmission on the Internet. A very high rate code called a Cyclic Redundancy Check (CRC) code is used to protect packets during transmission. At the receiver, if $d_H(\hat{\mathbf{z}}, \mathbf{x}^{(j)}) = 0$ for some j, the packet is deemed to be received without error, and the data bits forwarded to the higher layers, while if $d_H(\hat{\mathbf{z}}, \mathbf{x}^{(j)}) > 0$ for all j, then a decoding failure is deemed to have occurred and a re-transmission is requested. An alternative strategy is to treat decoding failures as *erasures*, a subject that we shall consider in the next Lecture.

While partitioning the receiver into a demodulator (sometimes referred to as a *hard-decision* demodulator since each $\mathbb{Y}_i$ is irretrievably quantized into a 0 or 1) and a decoder certainly reduces the complexity of implementation, it is also true that a larger SNR is required to achieve any desired error probability that with a true maximum-likelihood receiver. This is due in part to the hard decisions (*cf.* the discussion of the suboptimum receiver for repetition coding in Lecture 6) and in part because in many cases, people tend to treat decoding failures the same as errors. Even if re-transmissions are possible and all packets are ultimately successfully decoded with very low (essentially ignorable) error rate, the re-transmissions do reduce the data rate. Thus, coded communication systems of the type discussed in this subsection operate at rates considerably below capacity. The failure to come up with good designed codes together with the fact that *most* codes in $\mathfrak{C}_L$ are good codes led to the tongue-in-cheek aphorism that

"Most codes are good, except for the ones we can think of"

Fortunately, in the past fifteen years or so, there have been notable exciting developments in coding theory leading to the development of codes that achieve small error probabilities while operating at rates close to capacity. In the next subsection, we take a brief look at the basic notions involved in these codes and their decoding algorithms.

12.2.3. Tanner Graphs and Message-Passing Algorithms

We have seen that a linear(n, k) binary code $\mathcal{C}$ can be defined as the set of codewords $\mathbf{x} = \mathbf{u}G$ where G is a $k \times n$ binary matrix called the generator matrix of the code. The codewords are the *row space* of the matrix G and constitute a k-dimensional vector subspace. An alternative description of $\mathcal{C}$ is as the *null space* of a $(n-k) \times n$ matrix H called the *parity-check* matrix of the code. Thus, a binary vector $\mathbf{z}$ is a codeword if and only if $H\mathbf{z}^T = \mathbf{0}$ where $\mathbf{0}$ denotes a $(n-k)$-bit column of zeroes. Since every row of G is a codeword, we have that $HG^T = \mathbf{0}$ where here $\mathbf{0}$ means a $(n-k) \times k$ matrix of zeroes!

Consider a $(6, 3)$ code defined by the parity-check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{12.4}$$

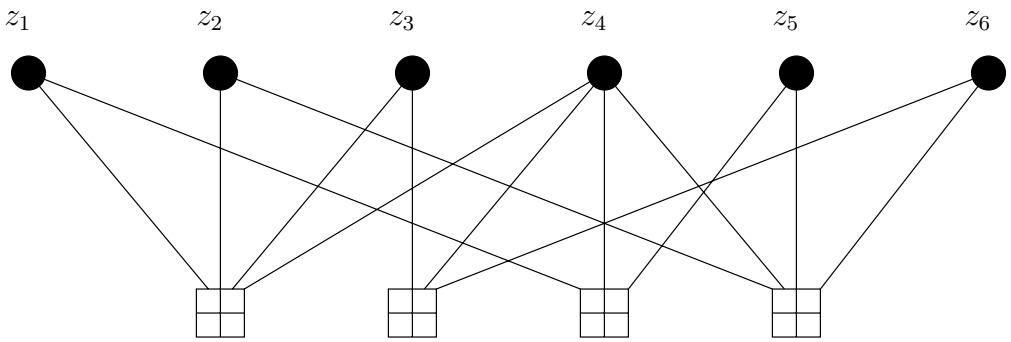
Thus, a vector $\mathbf{z} = [z_1, z_2, z_3, z_4, z_5, z_6]$ is a codeword if and only if

$$z_1 \oplus z_2 \oplus z_3 \oplus z_4 = 0; \quad z_3 \oplus z_4 \oplus z_6 = 0; \quad z_1 \oplus z_4 \oplus z_5 = 0 \tag{12.5}$$

which are called the parity-check equations satisfied by the code. Note that a sum of parity-check equations is another parity-check equation. For example,

$$(z_1 \oplus z_2 \oplus z_3 \oplus z_4) \oplus (z_3 \oplus z_4 \oplus z_6) \oplus (z_1 \oplus z_4 \oplus z_5) = z_2 \oplus z_4 \oplus z_5 \oplus z_6 = 0. \tag{12.6}$$

The relationships between the code symbols are often depicted in terms of a *Tanner graph*[2] (or factor graph) which is a graph with n vertices called code symbol nodes (black circles in the diagram below), one for each code symbol, and $m \geq n-k$ vertices called parity-check nodes (squares with a + sign in them in the diagram below), one for each parity-check equation. An edge connects a code symbol node to a parity-check node if the code symbol participates in that parity-check equation. For example, the Tanner graph shown below is for the code with parity-check matrix H in (12.4) with the parity-check equations shown in (12.5) and (12.6).



Message-passing algorithms for decoding a code work as follows. Suppose that receiver demodulates the received vector $\underline{\mathbb{Y}}$ into $\mathbf{z}$ as described earlier, and passes these n bits *together with their reliabilities* to the decoder. Here, reliability could, for example, be the value of $\mathbb{Y}^2$. The decoder can be thought of as carrying out Steps 1-4 below iteratively.

- Step 1: All the code symbol nodes pass their bit values z_i and the associated reliabilities to the parity-check nodes.
- Step 2: Each parity-check node determines if the z_i's satisfy the parity-check equation associated with that node. What happens at the next step depends on whether the parity-check fails or is satisfied.

[2]Dr. R. Michael Tanner, the inventor of what are now called Tanner graphs, is the Provost of the Chicago campus of the University of Illinois.

- Step 3: If the parity-check fails at a node, the parity-check node sends a message back to all *its* code symbol nodes that the check has failed. It also typically recommends that the *least reliable* of the z_i's connected to it change its value. (This *bit-flip* would ensure that the parity-check equation is satisfied.) The parity-check node also suggests new reliability levels to each of the z_i's connected to it. A typical message would suggest a reliability for the bit that was told to flip, and perhaps a small reduction in the reliability of the other, more reliable bit. After all, the parity-check *did* fail, and there is a small possibility that a z_i that has a high reliability is not so reliable after all.

  If the parity-check is satisfied at a node, the parity-check node sends a message back to all *its* code symbol nodes that the check was satisfied. It also recommends updates to the reliability of each of the z_i's connected to it. For example, each bit might be told to increase its reliability level a little

- Step 4: Each code symbol has received messages from the parity-check nodes connected to it, and these messages might well be in conflict. For example, some messages might demand that the bit be flipped while others may recommend that the bit stay the same and even increasing its reliability a little. Each code symbol node reconciles these messages, possibly flipping its bit value, and selecting a new reliability level for the bit.

  At this point, the process starts again with Step 1, with the newly updated bit values and reliabilities being sent from the code symbol nodes to the parity-check nodes, etc.

Message-passing algorithms employ distributed computation since each parity-check node sees only the bit values involved in its parity-check equation, and there is no global decision as to what each bit value or reliability ought to be. Information *does* percolate slowly between parity-check nodes through their shared bit values and reliabilties, and the algorithms do usually converge to fixed bit values for the z_i's, but it is also possible for the message-passing algorithm to get stuck in a loop in which code symbol bits keep flipping back and forth instead of converging to one value. Because of this, practical implementations stop the iterations of Steps 1-4 after a fixed number of iterations (chosen so that the message-passing algorithms have *usually* converged by then, or are stuck in a loop and are never going to converge. Message-passing decoding algorithms provide coding schemes that achieve low error probabilities at rates close to the channel capacity.

It is, of course, obvious, that message-passing algorithms require a fair amount of computation. Practical schemes use codes from a class called *low density parity check* (LDPC) codes. As the name implies, such codes have *sparse* parity-check matrices with relatively few 1s in them. Thus, there are relatively few edges in the Tanner graph which reduces the computaional effort required. These were invented in the early 1960s but were impractical to implement with the technology available at that time, and thus were put aside and disregarded for many years. However, extensive research over the past fifteen years together with technological improvements arising from Moore's law has made them quite practical for use, and LDPC codes are being adopted in various standards. They are definitely the wave of the future.