Chapter 0

REVIEW OF DISCRETE PROBABILITY THEORY

The *sample space*, Ω, is the set of possible outcomes of the random experiment in question. In discrete probability theory, the sample space is finite (i.e., $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$) or at most countably infinite (which we will indicate by $n = \infty$). An *event* is any subset of Ω, including the *impossible event*, $\varnothing$ (the empty subset of Ω) and the certain event, Ω. An event occurs when the outcome of a random experiment is a sample point in that event. The *probability measure*, P, assigns to each event a real number (which is the probability of that event) between 0 and 1 inclusive such that

$$P(\Omega) = 1 \tag{1}$$

and

$$P(A \cup B) = P(A) + P(B) \;\text{ if } A \cap B = \varnothing. \tag{2}$$

Notice that (2) implies

$$P(\varnothing) = 0$$

as we see by choosing $A = \varnothing$ and $B = \Omega$. The *atomic events* are the events that contain a single sample point. It follows from (2) that the numbers

$$p_i = P(\{\omega_i\}) \qquad i = 1, 2, \ldots, n \tag{3}$$

(i.e., the probabilities that the probability measure assigns to the atomic events) completely determine the probabilities of all events.

A *discrete random variable* is a mapping from the sample space into a specified finite or countably infinite set. For instance, on the sample space $\Omega = \{\omega_1, \omega_2, \omega_3\}$ we might define the random variables X, Y and Z as

ω	$X(\omega)$
ω_1	- 5
ω_2	0
ω_3	+ 5

ω	$Y(\omega)$
ω_1	yes
ω_2	yes
ω_3	no

ω	$Z(\omega)$
ω_1	[1, 0]
ω_2	[0, 1]
ω_3	[1, 1]

Note that the *range* (i.e., the set of possible values of the random variable) of these random variables are

$$X(\Omega) = \{-5, 0, +5\}$$
$$Y(\Omega) = \{\text{yes, no}\}$$
$$Z(\Omega) = \{[1, 0], [0, 1], [1, 1]\}.$$

The *probability distribution* (or "frequency distribution") of a random variable X, denoted P_X, is the mapping from $X(\Omega)$ into the interval $[0, 1]$ such that

$$P_X(x) = P(X = x). \tag{4}$$

Here $P(X = x)$ denotes the probability of the event that X takes on the value x, i.e., the event $\{\omega : X(\omega) = x\}$, which we usually write more simply as $\{X = x\}$. When X is real-valued, P_X is often called the *probability mass function* for X. It follows immediately from (4) that

$$P_X(x) \geq 0, \qquad \text{all } x \in X(\Omega) \tag{5}$$

and

$$\sum_x P_X(x) = 1 \tag{6}$$

where the summation in (6) is understood to be over all x in $X(\Omega)$. Equations (5) and (6) are the only mathematical requirements on a probability distribution, i.e., any function which satisfies (5) and (6) is the probability distribution for some suitably defined random variable on some suitable sample space.

In discrete probability theory, there is no mathematical distinction between a single random variable and a "random vector" (see the random variable Z in the example above.). However, if $X_1, X_2, \ldots, X_N$ are random variables on Ω, it is often convenient to consider their *joint probability distribution* defined as the mapping from $X_1(\Omega) \times X_2(\Omega) \times \cdots \times X_N(\Omega)$ into the interval $[0, 1]$ such that

$$P_{X_1X_2\ldots X_N}(x_1, x_2, \ldots, x_N) = P(\{X_1 = x_1\} \cap \{X_2 = x_2\} \cap \ldots \cap \{X_N = x_N\}). \tag{7}$$

It follows again immediately that

$$P_{X_1X_2\ldots X_N}(x_1, x_2, \ldots, x_N) \geq 0 \tag{8}$$

and that

$$\sum_{x_1}\sum_{x_2}\cdots\sum_{x_N} P_{X_1X_2\ldots X_N}(x_1, x_2, \ldots, x_N) = 1. \tag{9}$$

More interestingly, it follows from (7) that

$$\sum_{x_i} P_{X_1X_2\ldots X_N}(x_1, x_2, \ldots, x_N) = P_{X_1\ldots X_{i-1}X_{i+1}\ldots X_N}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N). \tag{10}$$

The random variables $x_1, x_2, \ldots x_N$ are said to be *statistically independent* when

$$P_{X_1X_2\ldots X_N}(x_1, x_2, \ldots, x_N) = P_{X_1}(x_1)P_{X_2}(x_2)\cdots P_{X_N}(x_N) \tag{11}$$

for all $x_1 \in X_1(\Omega), x_2 \in X_2(\Omega), \ldots, x_N \in X_N(\Omega)$.

Suppose that F is a real-valued function whose domain includes $X(\Omega)$. Then, the *expectation* of $F(X)$, denoted $\mathrm{E}[F(X)]$ or $\overline{F(X)}$, is the real number

$$\mathrm{E}[F(X)] = \sum_x P_X(x)F(x). \tag{12}$$

Note that the values of X need not be real numbers. The term *average* is synonymous with expectation. Similarly, when F is a real-valued function whose domain includes $X_1(\Omega) \times X_2(\Omega) \times \cdots \times X_N(\Omega)$, one defines

$$\mathrm{E}[F(X_1, X_2, \ldots, X_N)] = \sum_{x_1}\sum_{x_2}\cdots\sum_{x_N} P_{X_1X_2\ldots X_N}(x_1, x_2, \ldots, x_N)F(x_1, x_2, \ldots, x_N). \tag{13}$$

It is often convenient to consider *conditional probability distributions.* If $P_X(x) > 0$, then one defines

$$P_{Y|X}(y|x) = \frac{P_{XY}(x,y)}{P_X(x)}. \tag{14}$$

It follows from (14) and (10) that

$$P_{Y|X}(y|x) \geq 0 \qquad \text{all } y \in Y(\Omega) \tag{15}$$

and

$$\sum_y P_{Y|X}(y|x) = 1. \tag{16}$$

Thus, mathematically, there is no difference between a conditional probability distribution for Y (given, say, a value of X) and the (unconditioned) probability distribution for Y. When $P_X(x) = 0$, we cannot of course use (14) to define $P_{Y|X}(y|x)$. It is often said that $P_{Y|X}(y|x)$ is "undefined" in this case, but it is better to say that $P_{Y|X}(y|x)$ can be arbitrarily specified, provided that (15) and (16) are satisfied by the specification. This latter practice is often done in information theory to avoid having to treat as special cases those uninteresting situations where the conditioning event has zero probability.

If F is a real-valued function whose domain includes $X(\Omega)$, then the *conditional expectation* of $F(X)$ given the occurrence of the event A is defined as

$$\mathrm{E}[F(X)|A] = \sum_x F(x)P(X = x|A). \tag{17}$$

Choosing $A = \{Y = y_0\}$, we see that (17) implies

$$\mathrm{E}[F(X)|Y = y_0] = \sum_x F(x)P_{X|Y}(x|y_0). \tag{18}$$

More generally, when F is a real-valued function whose domain includes $X(\Omega) \times Y(\Omega)$, the definition (17) implies

$$\mathrm{E}[F(X,Y)|A] = \sum_x \sum_y F(x,y)P(\{X = x\} \cap \{Y = y\}|A). \tag{19}$$

Again nothing prevents us from choosing $A = \{Y = y_0\}$ in which case (19) reduces to

$$\mathrm{E}[F(X,Y)|Y = y_0] = \sum_x F(x, y_0)P_{X|Y}(x|y_0) \tag{20}$$

as follows from the fact that $P(\{X = x\} \cap \{Y = y\}|Y = y_0)$ vanishes for all y except $y = y_0$ in which case it has the value $P(X = x|Y = y_0) = P_{X|Y}(x|y_0)$. Multiplying both sides of the equation (20) by $P_Y(y_0)$ and summing over y_0 gives the relation

$$\mathrm{E}[F(X,Y)] = \sum_y \mathrm{E}[F(X,Y)|Y = y]P_Y(y) \tag{21}$$

where we have changed the dummy variable of summation from y_0 to y for clarity. Similarly, conditioning on an event A, we would obtain

$$\mathrm{E}[F(X,Y)|A] = \sum_y \mathrm{E}[F(X,Y)|\{Y = y\} \cap A]P(Y = y|A) \tag{22}$$

which in fact reduces to (21) when one chooses A to be the certain event Ω. Both (21) and (22) are referred to as statements of the *theorem on total expectation*, and are exceedingly useful in the calculation of expectations.

A sequence $Y_1, Y_2, Y_3, \ldots$ of real-valued random variables is said to *converge in probability* to the random variable Y, denoted

$$Y = \operatorname*{plim}_{N\to\infty} Y_N,$$

if for every positive ε it is true that

$$\lim_{N\to\infty} P(|Y - Y_N| < \varepsilon) = 1.$$

Roughly speaking, the random variables $Y_1, Y_2, Y_3, \ldots$ converge in probability to the random variable Y if, for every large N, it is virtually certain that the random variable Y_N will take on a value very close to that of Y. Suppose that $X_1, X_2, X_3, \ldots$ is a sequence of statistically independent and identically-distributed (i.i.d.) real-valued random variables, let m denote their common expectation, and let

$$Y_N = \frac{X_1 + X_2 + \cdots + X_N}{N}.$$

Then the *weak law of large numbers* asserts that

$$\operatorname*{plim}_{N\to\infty} Y_N = m,$$

i.e., that this sequence $Y_1, Y_2, Y_3, \ldots$ of random variables converges in probability to (the constant random variable whose values is always) m. Roughly speaking, the law of large numbers states that, for every large N, it is virtually certain that $\frac{X_1+X_2+\cdots+X_N}{N}$ will take on a value close to m.

Chapter 1

SHANNON'S MEASURE OF INFORMATION

1.1 Hartley's Measure

In 1948, Claude E. Shannon, then of the Bell Telephone Laboratories, published one of the most remarkable papers in the history of engineering. This paper ("A Mathematical Theory of Communication", *Bell System Tech. Journal*, Vol. 27, July and October 1948, pp. 379 - 423 and pp. 623 - 656) laid the groundwork of an entirely new scientific discipline, "information theory", that enabled engineers for the first time to deal quantitatively with the elusive concept of "information".

Perhaps the only precedent of Shannon's work in the literature is a 1928 paper by R.V.L. Hartley ("Transmission of Information", *Bell Syst. Tech. J.*, Vol. 3, July 1928, pp. 535 - 564). Hartley very clearly recognized certain essential aspects of information. Perhaps most importantly, he recognized that reception of a symbol provides information only if there had been other possibilities for its value besides that which was received. To say the same thing in more modern terminology, a symbol can give information only if it is the value of a random variable. This was a radical idea, which communications engineers were slow to grasp. Communication systems should be built to transmit random quantities, not to reproduce sinusoidal signals.

Hartley then went on to propose a quantitative measure of information based on the following reasoning. Consider a single symbol with D possible values. The information conveyed by n such symbols ought to be n times as much as that conveyed by one symbol, yet there are D^n possible values of the n symbols. This suggests that $\log(D^n) = n \log D$ is the appropriate measure of information where "the base selected (for the logarithm) fixes the size of the unit of information", to use Hartley's own words.

We can therefore express Hartley's measure of the amount of information provided by the observation of a discrete random variable X as:

$$I(X) = \log_b L \tag{1.1}$$

where

$$L = \text{number of possible values of } X.$$

When $b = 2$ in (1.1), we shall call Hartley's unit of information the "bit", although the word "bit" was not used until Shannon's 1948 paper. Thus, when $L = 2^n$, we have $I(X) = n$ bits of information – a single binary digit always gives exactly one bit of information according to Hartley's measure.

Hartley's simple measure provides the "right answer" to many technical problems. If there are eight telephones in some village, we could give them each a different three-binary-digit telephone number since 000, 001, 010, 011, 100, 101, 110, and 111 are the $8 = 2^3$ possible such numbers; thus, requesting a telephone number X in that village requires giving the operator 3 bits of information. Similarly, we need to use 16 binary digits to address a particular memory location in a memory with $2^{16} = 65536$ storage locations; thus, the address gives 16 bits of information.

Perhaps this is the place to dispel the notion that one bit is a small amount of information (even though Hartley's equation (1.1) admits no smaller non-zero amount). There are about $6 \times 10^9 \approx 2^{32.5}$ people living in the world today; thus, only 32.5 bits of information suffice to identify any person on the face of the earth today!

To see that there is something "wrong" with Hartley's measure of information, consider the random experiment where X is the symbol inscribed on a ball drawn "randomly" from a hat that contains some balls inscribed with 0 and some balls inscribed with 1. Since $L = 2$, Hartley would say that the observation of X gives 1 bit of information whether the hat was that shown in Figure 1.1a or that shown in Figure 1.1b. But, because for the hat of Fig. 1.1b we are rather sure in advance that $X = 0$ will occur, it seems intuitively clear that we get less information from observing this X that we would get had X come from the hat of Fig. 1.1a. The weakness of Hartley's measure of information is that it *ignores the probabilities* of the various values of X.

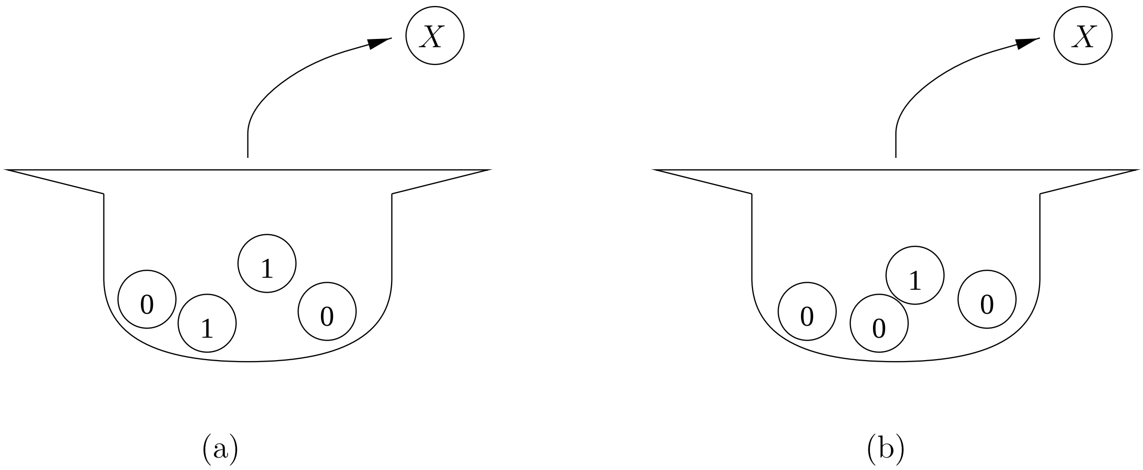


Figure 1.1: Two random experiments that give 1 bit of information by Hartley's measure.

Hartley's pioneering work seems to have had very little impact. He is much more remembered for his electronic oscillator than for his measure of information. The only trace of this latter contribution lies in the fact that information theorists have agreed to call Shannon's unit of information "the Hartley" when the base 10 is used for the logarithms. This is a questionable honor since no one is likely to use that base and, moreover, it is inappropriate because Hartley clearly recognized the arbitrariness involved in the choice of the base used with information measures. *Sic transit gloria mundi.*

1.2 Shannon's Measure

The publication, twenty years after Hartley's paper, of Shannon's 1948 paper, which proposed a new measure of information, touched off an explosion of activity in applying Shannon's concepts that continues still today. Shannon had evidently found something that Hartley had missed and that was essential to the general application of the theory.

Rather than stating Shannon's measure at the outset, let us see how Hartley might, with only a small additional effort, have been led to the same measure. Referring again to Hartley's hat of Figure 1.1b, we see that there is only one chance in four of choosing the ball marked "1". Thus choosing this ball is in a sense equivalent to choosing one out of 4 possibilities and should thus by Hartley's measure provide

$$\log_2(4) = 2 \text{ bits}$$

of information. But there are three chances out of four of choosing a ball marked "0". Thus, choosing such a ball is in a sense equivalent to choosing one out of only $\frac{4}{3}$ possibilities (whatever $\frac{4}{3}$ possibilities might mean!) and thus again by Hartley's measure provides only

$$\log_2(\frac{4}{3}) = 0.415 \text{ bits}$$

of information. But how do we reconcile these two quite different numbers? It seems obvious that we should weight them by their probabilities of occurrence to obtain

$$\frac{1}{4}(2) + \frac{3}{4}(0.415) = 0.811 \text{ bits}$$

which we could also write as

$$-\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = 0.811 \text{ bits}$$

of information as the amount provided by X. In general, if the i-th possible value of X has probability p_i, then the Hartley information $\log(1/p_i) = -\log p_i$ for this value should be weighted by p_i to give

$$-\sum_{i=1}^{L} p_i \log p_i \tag{1.2}$$

as the amount of information provided by X. This is precisely Shannon's measure, which we see can in a sense be considered the "average Hartley information".

There is a slight problem of what to do when $p_i = 0$ for one or more choices of i. From a practical viewpoint, we would conclude that, because the corresponding values of X never occur, they should not contribute to the information provided by X. Thus, we should be inclined to omit such terms from the sum in (1.2). Alternatively, we might use the fact that

$$\lim_{p \to 0_+} p \log p = 0 \tag{1.3}$$

as a mathematical justification for ignoring terms with $p_i = 0$ in (1.2). In any case, we have now arrived at the point where we can formally state Shannon's measure of information. It is convenient first to introduce some terminology and notation. If f is any real-valued function, then the *support* of f is defined as that subset of its domain where f takes on non-zero values, and is denoted $\text{supp}(f)$. Thus, if P_X is the probability distribution for the discrete random variable X, then $\text{supp}(P_X)$ is just the subset of possible values of X that have non-zero probability.

Definition 1.1: The *uncertainty* (or *entropy*) of a discrete random variable X is the quantity

$$H(X) = -\sum_{x \in \text{supp}(P_X)} P_X(x) \log_b P_X(x). \tag{1.4}$$

The choice of the base b (which of course must be kept constant in a given problem) determines the unit of information. When $b = 2$, the unit is called the *bit* (a word suggested to Shannon by J.W. Tukey as the contraction of "binary digit" – Tukey is better known for his work with fast Fourier transforms.)

When $b = e$, the only other base commonly used today in information theory, the unit is called the *nat*. Because $\log_2(e) \approx 1.443$, it follows that one nat of information equals about 1.443 bits of information.

In our definition of Shannon's measure, we have not used the word "information". In fact, we should be careful *not* to confuse "information" with "uncertainty". For Shannon, *information is what we receive when uncertainty is reduced.* The reason that the information we receive from observing the value of X equals $H(X)$ is that $H(X)$ is our *a priori* uncertainty about the value of X whereas *our a posteriori* uncertainty is *zero.* Shannon is ruthlessly consistent in defining information in all contexts as the difference between uncertainties. Besides the physically suggestive name "uncertainty" for $H(X)$, Shannon also used the name "entropy" because in statistical thermodynamics the formula for entropy is that of (1.4). Shannon also borrowed the symbol H from thermodynamics – but it would do no harm if we also thought of H as a belated honor to Hartley.

It should come as no surprise, in light of our discussion of Hartley's measure, that $H(X)$ can be expressed as an average or expected value, namely

$$\boxed{H(X) = \mathrm{E}[-\log P_X(X)] \qquad (1.5)}$$

provided that we adopt the *convention* (as we do here and hereafter) that *possible values with zero probability are to be ignored in taking the expectation of a real-valued function of a discrete random variable*, i.e., we define $\mathrm{E}[F(X)]$ to mean

$$\mathrm{E}[F(X)] = \sum_{x \in \mathrm{supp}(P_X)} P_X(x)F(x). \qquad (1.6)$$

Because there is no mathematical distinction between discrete random variables and discrete random vectors (we might in fact have $X = [Y, Z]$), it follows that (1.4) or (1.5) also defines the uncertainty of random vectors. The common tradition is to denote the uncertainty of $[Y, Z]$ as $H(YZ)$, but the notation $H(Y, Z)$ is also in use. We shall always use the former notation so that, for instance,

$$\boxed{H(XY) = \mathrm{E}[-\log P_{XY}(X, Y)] \qquad (1.7)}$$

which means

$$H(XY) = -\sum_{(x,y) \in \mathrm{supp}(P_{XY})} P_{XY}(x, y) \log P_{XY}(x, y). \qquad (1.8)$$

Example 1.1: Suppose that X has only two possible values x_1 and x_2 and that $P_X(x_1) = p$ so that $P_X(x_2) = 1 - p$. Then the uncertainty of X in bits, provided that $0 < p < 1$, is

$$H(X) = -p \log_2 p - (1 - p) \log_2(1 - p).$$

Because the expression on the right occurs so often in information theory, we give it its own name (the *binary entropy function*) and its own symbol ($h(p)$). The graph of $h(p)$ is given in Figure 1.2, as well as a table of useful numbers.

1.3 Some Fundamental Inequalities and an Identity

The following simple inequality is so often useful in information theory that we shall call it the "Information Theory (IT) Inequality":

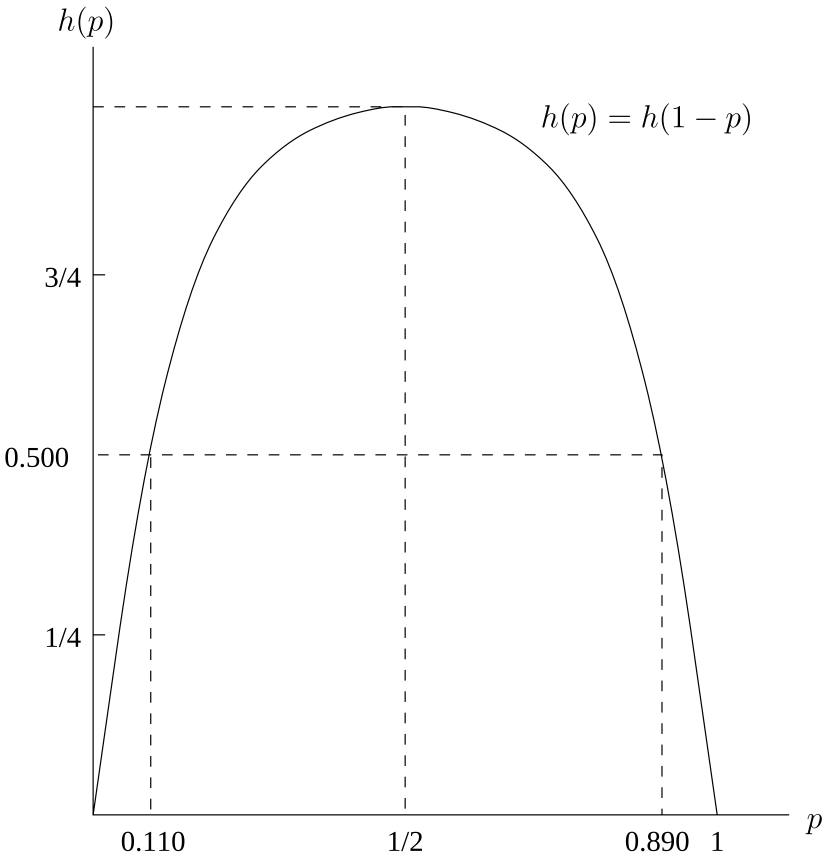


Figure 1.2: The Binary Entropy Function

IT-Inequality: For a positive real number r,

$$\log r \leq (r-1)\log e \tag{1.9}$$

with equality if and only if $r = 1$.

Proof: The graphs of $\ln r$ and of $r - 1$ coincide at $r = 1$ as shown in Figure 1.3. But

$$\frac{d}{dr}(\ln r) = \frac{1}{r}\begin{cases} > 1 & \text{for } r < 1 \\ < 1 & \text{for } r > 1 \end{cases}$$

so the graphs can never cross. Thus $\ln r \leq r - 1$ with equality if and only if $r = 1$. Multiplying both sides of this inequality by $\log e$ and noting that $\log r = (\ln r)(\log e)$ then gives (1.9). □

We are now ready to prove our first major result which shows that Shannon's measure of information coincides with Hartley's measure when and only when the possible values of X are all equally likely. It also shows the intuitively pleasing facts that the uncertainty of X is greatest when its values are equally likely and is least when one of its values has probability 1.

p	$-p\log_2 p$	$-(1-p)\log_2(1-p)$	$h(p)$ (bits)
0	0	0	0
.05	.216	.070	.286
.10	.332	.137	.469
.11	.350	.150	.500
.15	.411	.199	.610
.20	.464	.258	.722
.25	.500	.311	.811
.30	.521	.360	.881
1/3	.528	.390	.918
.35	.530	.404	.934
.40	.529	.442	.971
.45	.518	.474	.993
.50	1/2	1/2	1

Table 1.1: Values of the binary entropy function $h(p)$.

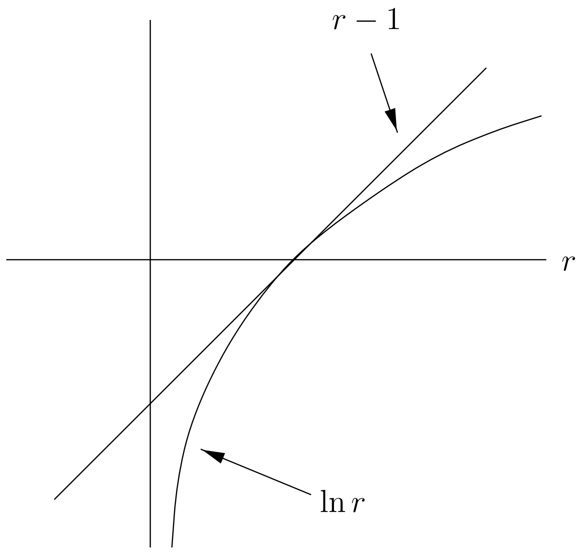


Figure 1.3: The graphs used to prove the IT-inequality.

Theorem 1.1: If the discrete random variable X has L possible values, then

$$0 \le H(X) \le \log L \tag{1.10}$$

with equality on the left if and only if $P_X(x) = 1$ for some x, and with equality on the right if and only if $P_X(x) = 1/L$ for all x.

Proof: To prove the left inequality in (1.10), we note that if $x \in \text{supp}\,(P_X)$, i.e., if $P_X(x) > 0$, then

$$-P_X(x)\log P_X(x) \begin{cases} = 0 & \text{if } P_X(x) = 1 \\ > 0 & \text{if } 0 < P_X(x) < 1. \end{cases}$$

Thus, we see immediately from (1.4) that $H(X) = 0$ if and only if $P_X(x)$ equals 1 for every x in supp(P_X), but of course there can be only one such x.

To prove the inequality on the right in (1.10), we use a common "trick" in information theory, namely we first write the quantity that we hope to prove is less than or equal to zero, then manipulate it into a form where we can apply the IT-inequality.

Thus, we begin with

$$\begin{aligned}
H(X) - \log L &= \left[- \sum_{x \in \operatorname{supp}(P_X)} P(x) \log P(x) \right] - \log L \\
&= \sum_{x \in \operatorname{supp}(P_X)} P(x) \left[\log \frac{1}{P(x)} - \log L \right] \\
&= \sum_{x \in \operatorname{supp}(P_X)} P(x) \log \frac{1}{LP(x)} \\
(IT-Inequality) \qquad &\leq \sum_{x \in \operatorname{supp}(P_X)} P(x) \left[\frac{1}{LP(x)} - 1 \right] \log e = \\
&= \left[\sum_{x \in \operatorname{supp}(P_X)} \frac{1}{L} - \sum_{x \in \operatorname{supp}(P_X)} P(x) \right] \log e \\
&\leq (1-1) \log e = 0
\end{aligned}$$

where equality holds at the place where we use the IT-inequality if and only if $LP(x) = 1$, for all x in $\operatorname{supp}(P_X)$, which in turn is true if and only if $LP(x) = 1$ for all L values of X. □

In the above proof, we have begun to *drop subscripts from probability distributions, but only when the subscript is the capitalized version of the argument.* Thus, we will often write simply $P(x)$ or $P(x, y)$ for $P_X(x)$ or $P_{XY}(x, y)$, respectively, but we would never write $P(0)$ for $P_X(0)$ or write $P(x_1)$ for $P_X(x_1)$. This convention will greatly simplify our notation with no loss of precision, provided that we are not careless with it.

Very often we shall be interested in the behavior of one random variable when another random variable is specified. The following definition is the natural generalization of uncertainty to this situation.

Definition 1.2: The *conditional uncertainty* (or *conditional entropy*) of the discrete random variable X, *given that the event* $Y = y$ occurs, is the quantity

$$H(X|Y = y) = - \sum_{x \in \operatorname{supp}\left(P_{X|Y}(.|y)\right)} P(x|y) \log P(x|y). \tag{1.11}$$

We note that (1.11) can also be written as a *conditional expectation*, namely

$$H(X|Y = y) = \mathrm{E}\left[-\log P_{X|Y}(X|Y) | Y = y \right]. \tag{1.12}$$

To see this, recall that the conditional expectation, given $Y = y$, of a real-valued function $F(X, Y)$ of the discrete random variables X and Y is defined as

$$\mathrm{E}[F(X, Y)|Y = y] = \sum_{x \in \operatorname{supp}\left(P_{X|Y}(.|y)\right)} P_{X|Y}(x|y) F(x, y). \tag{1.13}$$

Notice further that (1.6), (1.13) and the fact that $P_{XY}(x, y) = P_{X|Y}(x|y) P_Y(y)$ imply that the unconditional expectation of $F(X, Y)$ can be calculated as

$$\mathrm{E}[F(X, Y)] = \sum_{y \in \operatorname{supp}(P_Y)} P_Y(y) \mathrm{E}[F(X, Y)|Y = y]. \tag{1.14}$$

We shall soon have use for (1.14).

From the mathematical similarity between the definitions (1.4) and (1.11) of $H(X)$ and $H(X|Y = y)$, respectively, we can immediately deduce the following result.

Corollary 1 to Theorem 1.1: If the discrete random variable X has L possible values, then

$$0 \leq H(X|Y = y) \leq \log L \tag{1.15}$$

with equality on the left if and only if $P(x|y) = 1$ for some x, and with equality on the right if and only if $P(x|y) = 1/L$ for all x. (Note that y is a fixed value of Y in this corollary.)

When we speak about the "uncertainty of X given Y", we will mean the conditional uncertainty of X given the event $Y = y$, *averaged* over the possible values y of Y.

Definition 1.3: The *conditional uncertainty* (or *conditional entropy*) of the discrete random variable X *given the* discrete *random variable* Y is the quantity

$$H(X|Y) = \sum_{y \in \operatorname{supp}(P_Y)} P_Y(y) H(X|Y = y). \tag{1.16}$$

The following is a direct consequence of (1.16).

Corollary 2 to Theorem 1.1: If the discrete random variable X has L possible values, then

$$0 \leq H(X|Y) \leq \log L \tag{1.17}$$

with equality on the left if and only if for every y in $\operatorname{supp}(P_Y)$, $P_{X|Y}(x|y) = 1$ for some x (i.e., when Y *essentially determines* X), and with equality on the right if and only if, for every y in $\operatorname{supp}(P_Y)$, $P_{X|Y}(x|y) = 1/L$ for all x.

We now note, by comparing (1.12) and (1.16) to (1.14), that we can also write

$$H(X|Y) = \mathrm{E}\left[-\log P_{X|Y}(X|Y)\right] \tag{1.18}$$

which means

$$H(X|Y) = -\sum_{(x,y) \in \operatorname{supp}(P_{XY})} P_{XY}(x, y) \log P_{X|Y}(x|y). \tag{1.19}$$

For theoretical purposes, we shall find (1.18) most useful. However, (1.11) and (1.16) often provide the most convenient way to calculate $H(X|Y)$.

We now introduce the last of the uncertainty definitions for discrete random variables, which is entirely analogous to (1.12).

Definition 1.4: The *conditional uncertainty* (or *conditional entropy*) of the discrete random variable X *given* the discrete *random variable* Y *and given* that *the event* $Z = z$ occurs is the quantity

$$H(X|Y, Z = z) = \mathrm{E}\left[-\log P_{X|YZ}(X|YZ)|Z = z\right]. \tag{1.20}$$

Equivalently, we may write

$$H(X|Y, Z = z) = -\sum_{(x,y)\in\mathrm{supp}\left(P_{XY|Z}(.,.|z)\right)} P_{XY|Z}(x, y|z) \log P_{X|YZ}(x|y, z). \tag{1.21}$$

It follows further from (1.14), (1.20) and (1.18) that

$$H(X|Y, Z) = \sum_{z\in\mathrm{supp}(P_Z)} P_Z(z) H(X|Y, Z = z), \tag{1.22}$$

where again in invoking (1.18) we have again relied on the fact that there is no essential difference in discrete probability theory between a single random variable Y and a random vector (Y, Z). Equation (1.22), together with (1.21), often provides the most convenient means for computing $H(X|Y, Z)$.

1.4 Informational Divergence

We now introduce a quantity that is very useful in proving inequalities in information theory. Essentially, this quantity is a measure of the "distance" between two probability distributions.

Definition 1.4: If X and $\tilde{X}$ are discrete random variables with the same set of possible values, i.e. $X(\Omega) = \tilde{X}(\Omega)$, then the *information divergence between* P_X *and* $P_{\tilde{X}}$ is the quantity

$$D(P_X||P_{\tilde{X}}) = \sum_{x\in\mathrm{supp}(P_X)} P_X(x) \log \frac{P_X(x)}{P_{\tilde{X}}(x)}. \tag{1.23}$$

The informational divergence between P_X and $P_{\tilde{X}}$ is also known by several other names, which attests to its usefulness. It is sometimes called the *relative entropy between* X *and* $\tilde{X}$ and it is known in statistics as the *Kullbach-Leibler distance between* P_X *and* $P_{\tilde{X}}$.

Whenever we write $D(P_X||P_{\tilde{X}})$, we will be tacitly assuming that $X(\Omega) = \tilde{X}(\Omega)$, as without this condition the informational divergence is not defined. We see immediately from the definition (1.23) of $D(P_X||P_{\tilde{X}})$ that if there is an x in supp(P_X) but not in supp$(P_{\tilde{X}})$, i.e., for which $P_X(x) \neq 0$ but $P_{\tilde{X}}(x) = 0$, then

$$D(P_X||P_{\tilde{X}}) = \infty.$$

It is thus apparent that in general

$$D(P_X||P_{\tilde{X}}) \neq D(P_{\tilde{X}}||P_X)$$

so that informational divergence does not have the symmetry required for a true "distance" measure.

From the definition (1.23) of $D(P_X||P_{\tilde{X}})$, we see that the informational divergence may be written as an expectation, which we will find to be very useful, in the following manner:

$$D(P_X||P_{\tilde{X}}) = \mathrm{E}\left[\log \frac{P_X(X)}{P_{\tilde{X}}(X)}\right]. \tag{1.24}$$

The following simple and elegant inequality is the key to the usefulness of the informational divergence and supports its interpretation as a kind of "distance."

The Divergence Inequality:

$$D(P_X||P_{\tilde{X}}) \geq 0 \tag{1.25}$$

with equality if and only if $P_X = P_{\tilde{X}}$, i.e., if and only if $P_X(x) = P_{\tilde{X}}(x)$ for all $x \in X(\Omega) = \tilde{X}(\Omega)$.

Proof: We begin by writing

$$\begin{aligned}
-D(P_X||P_{\tilde{X}}) &= \sum_{x\in\mathrm{supp}(P_X)} P_X(x) \log \frac{P_{\tilde{X}}(x)}{P_X(x)} \\
\text{(IT-inequality)} &\leq \sum_{x\in\mathrm{supp}(P_X)} P_X(x) \left[\frac{P_{\tilde{X}}(x)}{P_X(x)} - 1\right] \log e = \\
&= \left[\sum_{x\in\mathrm{supp}(P_X)} P_{\tilde{X}}(x) - \sum_{x\in\mathrm{supp}(P_X)} P_X(x)\right] \log e \\
&\leq (1-1)\log e \\
&= 0.
\end{aligned}$$

Equality holds at the point where the IT-inequality was applied if and only if $P_X(x) = P_{\tilde{X}}(x)$ for all $x \in \mathrm{supp}(P_X)$, which is equivalent to $P_X(x) = P_{\tilde{X}}(x)$ for all $x \in X(\Omega) = \tilde{X}(\Omega)$ and hence also gives equality in the second inequality above. □

The following example illustrates the usefulness of the informational divergence.

Example 1.2: Suppose that X has L possible values, i.e., $\#(X(\Omega)) = L$. Let $\tilde{X}$ have the probability distribution $P_{\tilde{X}}(x) = 1/L$ for all $x \in X(\Omega)$. Then

$$\begin{aligned}
D(P_X||P_{\tilde{X}}) &= \mathrm{E}\left[\log \frac{P_X(X)}{P_{\tilde{X}}(X)}\right] \\
&= \mathrm{E}\left[\log \frac{P_X(X)}{1/L}\right] \\
&= \log L - \mathrm{E}\left[-\log P_X(X)\right] \\
&= \log L - H(X).
\end{aligned}$$

It thus follows from the Divergence Inequality that $H(X) \leq \log L$ with equality if and only if $P_X(x) = 1/L$ for all $x \in X(\Omega)$, which is the fundamental inequality on the right in Theorem 1.1.

1.5 Reduction of Uncertainty by Conditioning

We now make use of the Divergence Inequality to prove a result that is extremely useful and that has an intuitively pleasing interpretation, namely that knowing Y reduces, in general, our uncertainty about X.

Theorem 1.2 [The Second Entropy Inequality]: For any two discrete random variables X and Y,

$$H(X|Y) \le H(X)$$

with equality if and only if X and Y are independent random variables.

Proof: Making use of (1.5) and(1.18), we have

$$\begin{aligned} H(X) - H(X|Y) &= \mathrm{E}[-\log P_X(X)] - \mathrm{E}\left[-\log P_{X|Y}(X|Y)\right] \\ &= \mathrm{E}\left[\log \frac{P_{X|Y}(X|Y)}{P_X(X)}\right] \\ &= \mathrm{E}\left[\log \frac{P_{X|Y}(X|Y)P_Y(Y)}{P_X(X)P_Y(Y)}\right] \\ &= \mathrm{E}\left[\log \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)}\right] \\ &= D(P_{XY}||P_{\tilde{X}\tilde{Y}}) \end{aligned}$$

where

$$P_{\tilde{X}\tilde{Y}}(x,y) = P_X(x)P_Y(y)$$

for all $(x,y) \in X(\Omega) \times Y(\Omega)$. It now follows from the Divergence Inequality that $H(X|Y) \le H(X)$ with equality if and only if $P_{XY}(x,y) = P_X(x)P_Y(y)$ for all $(x,y) \in X(\Omega) \times Y(\Omega)$, which is the definition of independence for discrete random variables. □

We now observe that (1.19) and (1.21) differ only in the fact that the probability distributions in the latter are further conditioned on the event $Z = z$. Thus, because of this mathematical similarity, we can immediately state the following result.

Corollary 1 to Theorem 1.2: For any three discrete random variables X, Y and Z,

$$H(X|Y, Z = z) \le H(X|Z = z)$$

with equality if and only if $P_{XY|Z}(x,y|z) = P_{X|Z}(x|z)P_{Y|Z}(y|z)$ for all x and y. (Note that z is a fixed value of Z in this corollary.)

Upon multiplying both sides of the inequality in the above corollary by $P_Z(z)$ and summing over all z in supp(P_Z), we obtain the following useful and intuitively pleasing result, which is the last of our inequalities relating various uncertainties and which again shows how conditioning reduces uncertainty.

Corollary 2 to Theorem 1.2 [The Third Entropy Inequality]: For any three discrete random variables X, Y and Z,

$$H(X|YZ) \le H(X|Z)$$

with equality if and only if, for every z in supp(P_Z), the relation $P_{XY|Z}(x,y|z) = P_{X|Z}(x|z)P_{Y|Z}(y|z)$ holds for all x and y [or, equivalently, if and only if X and Y are independent when conditioned on knowing Z].

We can summarize the above inequalities by saying that *conditioning on random variables can only decrease uncertainty* (more precisely, can never increase uncertainty). This is again an intuitively pleasing property of Shannon's measure of information. The reader should note, however, that *conditioning on an event can increase uncertainty*, e.g., $H(X|Y=y)$ can exceed $H(X)$. It is only the average of $H(X|Y=y)$ over all values y of Y, namely $H(X|Y)$, that cannot exceed $H(X)$. That this state of affairs is not counter to the intuitive notion of "uncertainty" can be seen by the following reasoning. Suppose that X is the color (yellow, white or black) of a "randomly selected" earth-dweller and that Y is his nationality. "On the average", telling us Y would reduce our "uncertainty" about X ($H(X|Y) < H(X)$). However, because there are many more earth-dwellers who are yellow than are black or white, our "uncertainty" about X would be increased if we were told that the person selected came from a nation y in which the numbers of yellow, white and black citizens are roughly equal ($H(X|Y=y) > H(X)$ in this case). See also Example 1.3 below.

1.6 The Chain Rule for Uncertainty

We now derive one of the simplest, most intuitively pleasing, and most useful identities in information theory. Let $[X_1, X_2, \ldots, X_N]$ be a discrete random vector with N component discrete random variables. Because discrete random vectors are also discrete random variables, (1.5) gives

$$H(X_1X_2\ldots X_N) = \mathrm{E}\left[-\log P_{X_1X_2,\ldots,X_N}(X_1, X_2, \ldots, X_N)\right], \tag{1.26}$$

which can also be rewritten via the multiplication rule for probability distributions as

$$H(X_1X_2\ldots X_N) = \mathrm{E}[-\log(P_{X_1}(X_1)P_{X_2|X_1}(X_2|X_1)\ldots P_{X_N|X_1,\ldots,X_{N-1}}(X_N|X_1,\ldots,X_{N-1}))]. \tag{1.27}$$

We write this identity more compactly as

$$\begin{aligned} H(X_1X_2\ldots X_N) &= \mathrm{E}\left[-\log\prod_{n=1}^{N} P_{X_n|X_1\ldots X_{n-1}}(X_n|X_1,\ldots,X_{n-1})\right] \\ &= \sum_{n=1}^{N}\mathrm{E}\left[-\log P_{X_n|X_1\ldots X_{n-1}}(X_n|X_1,\ldots,X_{n-1})\right] \\ &= \sum_{n=1}^{N} H(X_n|X_1\ldots X_{n-1}). \end{aligned} \tag{1.28}$$

In less compact, but more easily read, notation, we can rewrite this last expansion as

$$\boxed{H(X_1X_2\ldots X_N) = H(X_1) + H(X_2|X_1) + \cdots + H(X_N|X_1\ldots X_{N-1}).} \tag{1.29}$$

This identity, which is sometimes referred to as the *chain rule for uncertainty*, can be phrased as stating that "the uncertainty of a random vector equals the uncertainty of its first component, plus the uncertainty of its second component when the first is known, ..., plus the uncertainty of its last component when all previous components are known." This is such an intuitively pleasing property that it tempts us to conclude, before we have made a single application of the theory, that Shannon's measure of information is *the* correct one.

It should be clear from our derivation of (1.29) that the order of the components is arbitrary. Thus, we can expand, for instance $H(XYZ)$, in any of the six following ways:

$$\begin{aligned}
H(XYZ) &= H(X)+H(Y|X)+H(Z|XY)\\
&= H(X)+H(Z|X)+H(Y|XZ)\\
&= H(Y)+H(X|Y)+H(Z|XY)\\
&= H(Y)+H(Z|Y)+H(X|YZ)\\
&= H(Z)+H(X|Z)+H(Y|XZ)\\
&= H(Z)+H(Y|Z)+H(X|YZ).
\end{aligned}$$

Example 1.3: Suppose that the random vector $[X, Y, Z]$ is equally likely to take any of the following four values: $[0, 0, 0], [0, 1, 0], [1, 0, 0]$ and $[1, 0, 1]$. Then $P_X(0) = P_X(1) = 1/2$ so that

$$H(X) = h(1/2) = 1 \text{ bit.}$$

Note that $P_{Y|X}(0|1) = 1$ so that

$$H(Y|X=1) = 0.$$

Similarly, $P_{Y|X}(0|0) = 1/2$ so that

$$H(Y|X=0) = h(1/2) = 1 \text{ bit.}$$

Using (1.16) now gives

$$H(Y|X) = \frac{1}{2}(1) = 1/2 \text{ bit.}$$

From the fact that $P(z|xy)$ equals 1, 1 and 1/2 for (x, y, z) equal to $(0, 0, 0), (0, 1, 0)$ and $(1, 0, 0)$, respectively, it follows that

$$\begin{aligned}
H(Z|X=0, Y=0) &= 0\\
H(Z|X=0, Y=1) &= 0\\
H(Z|X=1, Y=0) &= 1 \text{ bit.}
\end{aligned}$$

Upon noting that $P(x, y)$ equals 1/4, 1/4 and 1/2 for (x, y) equal to $(0, 0), (0, 1)$ and $(1, 0)$, respectively, we find from (1.16) that

$$H(Z|XY) = \frac{1}{4}(0) + \frac{1}{4}(0) + \frac{1}{2}(1) = 1/2 \text{ bit.}$$

We now use (1.29) to obtain

$$H(XYZ) = 1 + 1/2 + 1/2 = 2 \text{ bits.}$$

Alternatively, we could have found $H(XYZ)$ more simply by noting that $[X, Y, Z]$ is equally likely to take on any of 4 values so that

$$H(XYZ) = \log 4 = 2 \text{ bits.}$$

Because $P_Y(1) = 1/4$, we have

$$H(Y) = h(1/4) = .811 \text{ bits.}$$

Thus, we see that

$$H(Y|X) = 1/2 < H(Y) = .811$$

in agreement with Theorem 1.1. Note, however, that

$$H(Y|X=0) = 1 > H(Y) = .811 \text{ bits.}$$

We leave as exercises showing that conditional uncertainties can also be expanded analogously to (1.29), i.e., that

$$\begin{aligned}
H(X_1X_2\dots X_N|Y=y) &= H(X_1|Y=y) + H(X_2|X_1, Y=y)\\
&\quad + \cdots + H(X_N|X_1\dots X_{N-1}, Y=y),
\end{aligned}$$

$$
\begin{aligned}
H(X_1X_2\dots X_N|Y) &= H(X_1|Y) + H(X_2|X_1Y) \\
&\quad + \cdots + H(X_N|X_1\dots X_{N-1}Y),
\end{aligned}
$$

and finally

$$
\begin{aligned}
H(X_1X_2\dots X_N|Y, Z=z) &= H(X_1|Y, Z=z) + H(X_2|X_1Y, Z=z) \\
&\quad + \cdots + H(X_N|X_1\dots X_{N-1}Y, Z=z).
\end{aligned}
$$

Again, we emphasize that the order of the component random variables $X_1, X_2, \dots, X_N$ appearing in the above expansions is entirely arbitrary. Each of these conditional uncertainties can be expanded in $N!$ ways, corresponding to the $N!$ different orderings of these random variables.

1.7 Mutual information

We have already mentioned that uncertainty is the basic quantity in Shannon's information theory and that, for Shannon, "information" is always a difference of uncertainties. How much information does the random variable Y give about the random variable X? Shannon's answer would be "the amount by which Y reduces the uncertainty about X", namely $H(X) - H(X|Y)$.

Definition 1.5: The mutual information between the discrete random variables X and Y is the quantity

$$I(X;Y) = H(X) - H(X|Y). \tag{1.30}$$

The reader may well wonder why we use the term "mutual information" rather than "information provided by Y about X" for $I(X;Y)$. To see the reason for this, we first note that we can expand $H(XY)$ in two ways, namely

$$
\begin{aligned}
H(XY) &= H(X) + H(Y|X) \\
&= H(Y) + H(X|Y)
\end{aligned}
$$

from which it follows that

$$H(X) - H(X|Y) = H(Y) - H(Y|X)$$

or, equivalently, that

$$I(X;Y) = I(Y;X). \tag{1.31}$$

Thus, we see that X cannot avoid giving the same amount of information about Y as Y gives about X. The relationship is entirely symmetrical. The giving of information is indeed "mutual".

Example 1.4: (Continuation of Example 1.3). Recall that $P_Y(1) = 1/4$ so that

$$H(Y) = h(1/4) = .811 \text{ bits.}$$

Thus,

$$
\begin{aligned}
I(X;Y) &= H(Y) - H(Y|X) \\
&= .811 - .500 = .311 \text{ bits.}
\end{aligned}
$$

In words, the first component of the random vector $[X, Y, Z]$ gives .311 bits of information about the second component, and vice versa.

It is interesting to note that Shannon in 1948 used neither the term "mutual information" nor a special symbol to denote it, but rather always used differences of uncertainties. The terminology "mutual information" (or "average mutual information" as it is often called) and the symbol $I(X;Y)$ were introduced later by Fano. We find it convenient to follow Fano's lead, but we should never lose sight of Shannon's idea that information is nothing but a change in uncertainty.

The following two definitions should now seem natural.

Definition 1.6: The conditional mutual information between the discrete random variables X and Y, *given* that *the event* $Z = z$ occurs, is the quantity

$$I(X;Y|Z=z) = H(X|Z=z) - H(X|Y,Z=z). \tag{1.32}$$

Definition 1.7: The conditional mutual information between the discrete random variables X and Y *given the* discrete *random variable* Z is the quantity

$$I(X;Y|Z) = H(X|Z) - H(X|YZ). \tag{1.33}$$

From (1.16) and (1.26), it follows that

$$I(X;Y|Z) = \sum_{z \in \text{supp}(P_Z)} P_Z(z) I(X;Y|Z=z). \tag{1.34}$$

From the fact that $H(XY|Z=z)$ and $H(XY|Z)$ can each be expanded in two ways, namely

$$\begin{aligned} H(XY|Z=z) &= H(X|Z=z) + H(Y|X,Z=z) \\ &= H(Y|Z=z) + H(X|Y,Z=z) \end{aligned}$$

and

$$\begin{aligned} H(XY|Z) &= H(X|Z) + H(Y|XZ) \\ &= H(Y|Z) + H(X|YZ), \end{aligned}$$

it follows from the definitions (1.32) and (1.33) that

$$I(X;Y|Z=z) = I(Y;X|Z=z) \tag{1.35}$$

and that

$$I(X;Y|Z) = I(Y;X|Z). \tag{1.36}$$

We consider next the fundamental inequalities satisfied by mutual information. Definition (1.30), because of (1.17) and (1.31), immediately implies the following result.

Theorem 1.3: For any two discrete random variables X and Y,

$$0 \leq I(X;Y) \leq \min[H(X), H(Y)] \tag{1.37}$$

with equality on the left if and only if X and Y are independent random variables, and with equality on the right if and only if either Y essentially determines X, or X essentially determines Y, or both.

Similarly, definitions (1.32) and (1.33) lead to the inequalities

$$0 \leq I(X;Y|Z=z) \leq \min[H(X|Z=z), H(Y|Z=z)] \tag{1.38}$$

and

$$0 \leq I(X;Y|Z) \leq \min[H(X|Z), H(Y|Z)], \tag{1.39}$$

respectively. We leave it to the reader to state the conditions for equality in the inequalities (1.38) and (1.39).

We end this section with a caution. Because conditioning reduces uncertainty, one is tempted to guess that the following inequality should hold:

$$I(X;Y|Z) \leq I(X;Y)$$

This, however, is not true in general. That this does not contradict good intuition can be seen by supposing that X is the plaintext message in a secrecy system, Y is the encrypted message, and Z is the secret key. In fact, both $I(X;Y|Z) < I(X;Y)$ and $I(X;Y|Z) > I(X;Y)$ are possible.

1.8 Suggested Readings

N. Abramson, *Information Theory and Coding.* New York: McGraw-Hill, 1963. (This is a highly readable, yet precise, introductory text book on information theory. It demands no sophisticated mathematics but it is now somewhat out of date.)

R. E. Blahut, *Principles and Practice of Information Theory.* Englewood Cliffs, NJ: Addison-Wesley, 1987. (One of the better recent books on information theory and reasonably readable.)

T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991. (This is the newest good book on information theory and is written at a sophisticated mathematical level.)

R. G. Gallager, *Information Theory and Reliable Communication.* New York: Wiley, 1968. (This is by far the best book on information theory, but it is written at an advanced level and beginning to be a bit out of date. A new edition is in the writing process.)

R. J. McEliece, *The Theory of Information and Coding.* (Encyclopedia of Mathematics and Its Applications, vol. 3, Sec. Probability.) Reading, Mass.: Addison-Wesley, 1977. (This is a very readable small book containing many recent results in information theory.)

C. E. Shannon and W. Weaver, *A Mathematical Theory of Communication.* Urbana, Ill., Illini Press, 1949. (This inexpensive little paperback book contains the reprint of Shannon's original 1948 article in

the Bell System Technical Journal. The appearance of this article was the birth of information theory. Shannon's article is lucidly written and should be read by anyone with an interest in information theory.)

Key Papers in the Development of Information Theory (Ed. D. Slepian). New York: IEEE Press, 1973. Available with cloth or paperback cover and republished in 1988. (This is a collection of many of the most important papers on information theory, including Shannon's 1948 paper and several of Shannon's later papers.)

Chapter 2

EFFICIENT CODING OF INFORMATION

2.1 Introduction

In the preceding chapter, we introduced Shannon's measure of information and showed that it possessed several intuitively pleasing properties. However, we have not yet shown that it is the "correct" measure of information. To do that, we must show that the answers to practical problems of information transmittal or storage can be expressed simply in terms of Shannon's measure. In this chapter, we show that Shannon's measure does exactly this for the problem of coding a digital information source into a sequence of letters from a given alphabet. We shall also develop some efficient methods for performing such a coding.

We begin, however, in a much more modest way by considering how one can efficiently code a single random variable.

2.2 Coding a Single Random Variable

2.2.1 Prefix-Free Codes and the Kraft Inequality

We consider the conceptual situation shown in Figure 2.1 in which a single random variable U is encoded into D-ary digits.

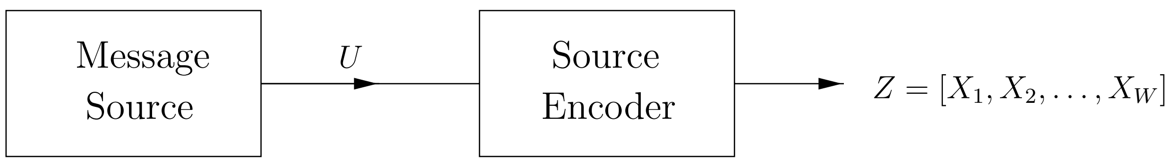


Figure 2.1: A Variable-Length Coding Scheme

More precisely, for the quantities shown in Figure 2.1:

(1) U is a random variable taking values in the alphabet $\{u_1, u_2, \ldots, u_K\}$.

(2) Each X_i takes on values in the D-ary alphabet, which we will usually take to be $\{0, 1, \ldots, D-1\}$.

(3) W is a random variable, i.e., the values of the random variable $Z = [X_1, X_2, \ldots, X_W]$ are D-ary sequences of varying length.

By a code for U, we mean a list $(z_1, z_2, \ldots, z_K)$ of D-ary sequences where we understand z_i to be the codeword for u_i, i.e., $U = u_i$ implies $Z = z_i$. We take the smallness of $\mathrm{E}[W]$, the *average codeword length*, as the measure of goodness of the code. If $z_i = [x_{i1}, x_{i2}, \ldots, x_{iw_i}]$ is the codeword for u_i and w_i is the length of this codeword, then we can write the average codeword length as

$$\mathrm{E}[W] = \sum_{i=1}^{K} w_i P_U(u_i), \tag{2.1}$$

since W is a real-valued function of the random variable U. Note that $X_1, X_2, \ldots, X_W$ are in general only conditionally defined random variables since X_i takes on a value only when $W \geq i$.

We will say that the D-ary sequence z is a *prefix* of the D-ary sequence z' if z has length n $(n \geq 1)$ and the first n digits of z' form exactly the sequence z. Note that a sequence is trivially a prefix of itself.

We now place the following requirement on the type of coding schemes for U that we wish to consider:

No codeword is the prefix of another codeword.

This condition assures that a codeword can be recognized as soon as its last digit is received, even when the source encoder is used repeatedly to code a succession of messages from the source. This condition also ensures that no two codewords are the same and hence that $H(Z) = H(U)$. A code for U which satisfies the above requirement is called a *prefix-free* code (or an *instantaneously decodable* code.)

Example 2.1:

U	Z
u_1	0
u_2	10
u_3	11

This is a prefix-free code.

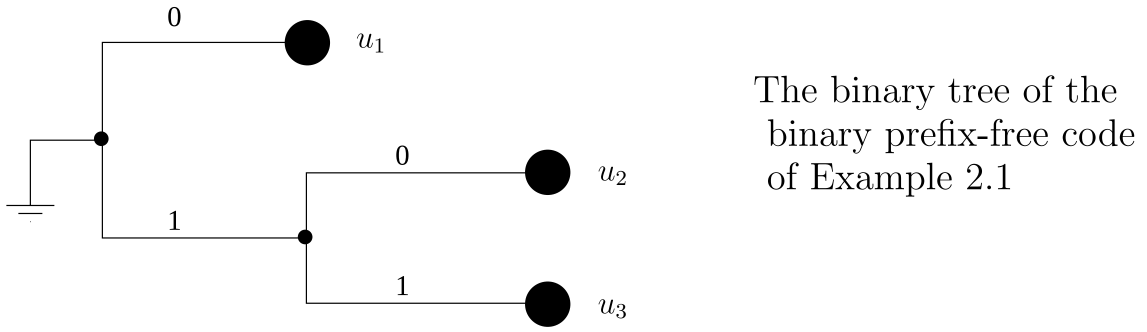
U	Z
u_1	1
u_2	00
u_3	11

This is not a prefix-free code since z_1 is a prefix of z_3.

Why it makes sense to consider only prefix-free codes for a random variable U can be seen by reflecting upon what could happen if, in the small village of Chapter 1, one subscriber had the phone number 011 and another 0111. We will see other reasons later for the prefix-free requirement.

To gain insight into the nature of a prefix-free code, it is useful to show the digits in the codewords of some code as the labels on the branches of a rooted tree. In Figure 2.2, we show the "binary trees" corresponding to the two binary codes of Example 2.1. Note that the codewords (shown by the large dark circles at the end of the branch labelled with the last digit of the codeword) of the prefix-free code all correspond to *leaves* (i.e., vertices with no outgoing branches) in its tree, but that the non-prefix code also has a codeword corresponding to a *node* (i.e., a vertex with outgoing branches), where direction is taken away from the specially labelled vertex called the *root* (and indicated in the diagrams by an electrical ground.)

To make our considerations more precise, we introduce some definitions.



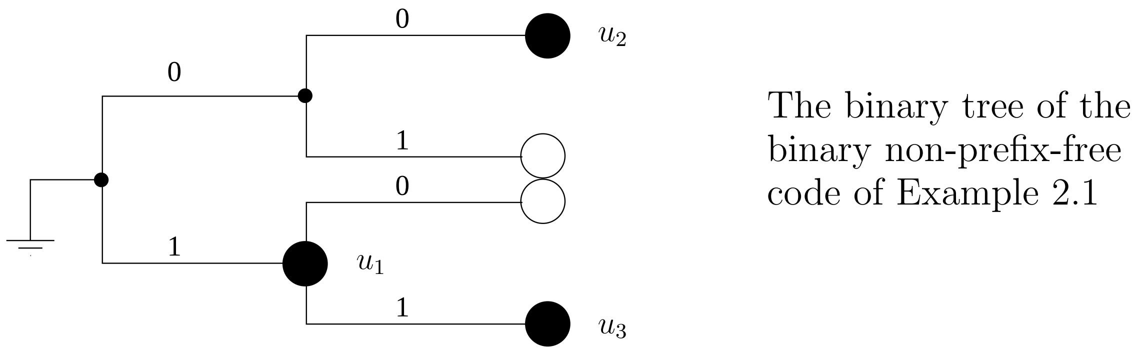


Figure 2.2: The trees of two binary codes.

Definition 2.1: A *D-ary tree* is a finite (or semi-infinite) rooted tree such that D branches stem outward (i.e., in the direction away from the root) from each node.

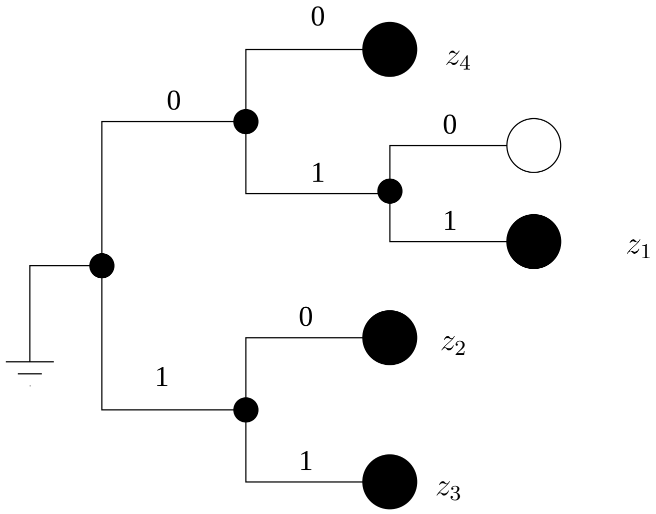
We shall label the D branches stemming outward from a node with the D different D-ary letters, $0, 1, \ldots, D-1$ in our coding alphabet.

Definition 2.2: The *full D-ary tree of length N* is the D-ary tree with D^N leaves, each at depth N from the root.

See Figure 2.3 for examples of D-ary trees.

It should now be self-evident that every D-ary prefix-free code can be identified with a set of leaves in a D-ary tree, and that, conversely, any set of leaves in a D-ary tree defines a D-ary prefix-free code. (Hereafter we show in our D-ary trees those leaves used as codewords as darkened circles and those not used as codewords as hollow circles.) We make the D-ary tree for a given prefix-free code unique by "pruning" the tree at every node from which no codewords stem.

Example 2.2: The prefix-free code $z_1 = [011]$ $z_2 = [10]$ $z_3 = [11]$ and $z_4 = [00]$ has the tree



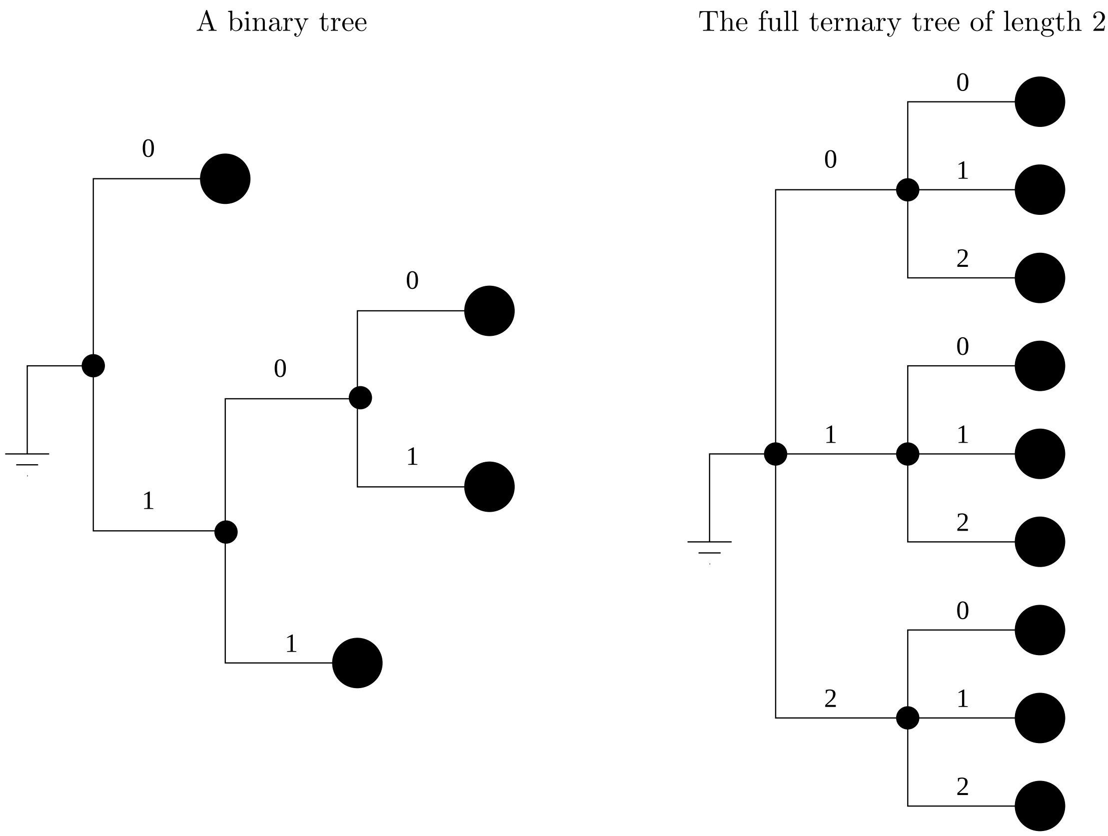


Figure 2.3: Examples of D-ary trees.

The following result tells us exactly when a given set of codeword lengths can be realized by a D-ary prefix-free code.

The Kraft Inequality: There exists a D-ary prefix-free code whose codeword lengths are the positive integers $w_1, w_2, \ldots, w_K$ if and only if

$$\sum_{i=1}^{K} D^{-w_i} \leq 1. \tag{2.2}$$

When (2.2) is satisfied with equality, the corresponding prefix-free code has no unused leaves in its D-ary tree.

Proof: We shall use the fact that, in the full D-ary tree of length N, D^{N-w} leaves stem from each node at depth w where $w < N$.

Suppose first that there does exist a D-ary prefix-free code whose codeword lengths are $w_1, w_2, \ldots, w_K$. Let $N = \max_i w_i$ and consider constructing the tree for the code by pruning the full D-ary tree of length N at all vertices corresponding to codewords. Beginning with $i = 1$, we find the node (at leaf) z_i in what is left of the original full D-ary tree of length N and, if $w_i < N$, we prune the tree there to make this vertex a leaf at depth w_i. By this process, we delete D^{N-w_i} leaves from the tree, since none of these leaves could have previously been deleted because of the prefix-free condition. But there are only D^N leaves that can be deleted so we must find after all codewords have been considered that

$$D^{N-w_1} + D^{N-w_2} + \cdots + D^{N-w_K} \leq D^N.$$

Dividing by D^N gives the inequality (2.2) as a necessary condition for a prefix-free code.

Suppose, conversely, that $w_1, w_2, \ldots, w_K$ are positive integers such that (2.2) is satisfied. Without loss of essential generality, we may assume that we have *ordered* these *lengths* so that $w_1 \leq w_2 \leq \cdots \leq w_K$. Consider then the following algorithm: (Let $N = \max_i w_i$ and start with the full D-ary tree of length N.)

(1) $i \leftarrow 1$.

(2) Choose z_i as any surviving node or leaf at depth w_i (not yet used as a codeword) and, if $w_i < N$, prune the tree at z_i so as to make z_i a leaf. Stop if there is no such surviving node or leaf.

(3) If $i = K$, stop. Otherwise, $i \leftarrow i + 1$ and go to step (2).

If we are able to choose z_K in step (2), then we will have constructed a prefix-free D-ary code with the given codeword length. We now show that we can indeed choose z_i in step (2) for all $i \leq K$. Suppose that $z_1, z_2, \ldots, z_{i-1}$ have been chosen. The number of surviving leaves at depth N not stemming from any codeword is

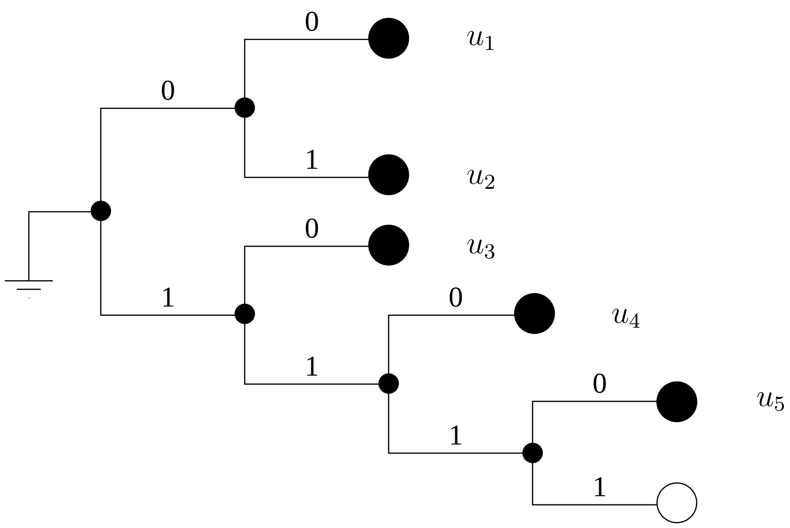
$$D^N - \left(D^{N-w_1} + D^{N-w_2} + \cdots + D^{N-w_{i-1}}\right) = D^N \left(1 - \sum_{j=1}^{i-1} D^{-w_j}\right).$$

Thus, if $i \leq K$, condition (2.2) shows that the number of surviving leaves at depth N is greater than zero. But if there is a surviving leaf at depth N, then there must also be (unused) surviving nodes at depth $w_i < N$. Since $w_1 \leq w_2 \leq \cdots \leq w_{i-1} \leq w_i$, no already chosen codeword can stem outward from such a surviving node and hence this surviving node may be chosen as z_i. Thus condition (2.2) is also sufficient for the construction of a D-ary prefix-free code with the given codeword lengths. □

Note that our proof of the Kraft inequality actually contained an *algorithm for constructing a D-ary prefix-free code given the codeword lengths*, $w_1, w_2, \ldots, w_K$, wherever such a code exists, i.e. whenever (2.2) is satisfied. There is no need, however, to start from the full-tree of length $N = \max_i w_i$ since the algorithm given in the proof is fully equivalent to growing the tree from the root and selecting any leaf at depth w_i for the codeword z_i *provided that the codewords with smaller length are chosen sooner.*

Example 2.3: Construct a binary prefix-free code with lengths $w_1 = 2, w_2 = 2, w_3 = 2, w_4 = 3, w_5 = 4$.

Since $\sum_{i=1}^{5} 2^{-w_i} = 1/4 + 1/4 + 1/4 + 1/8 + 1/16 = 15/16 < 1$, we know such a prefix-free code exists. We "grow" such a code to obtain

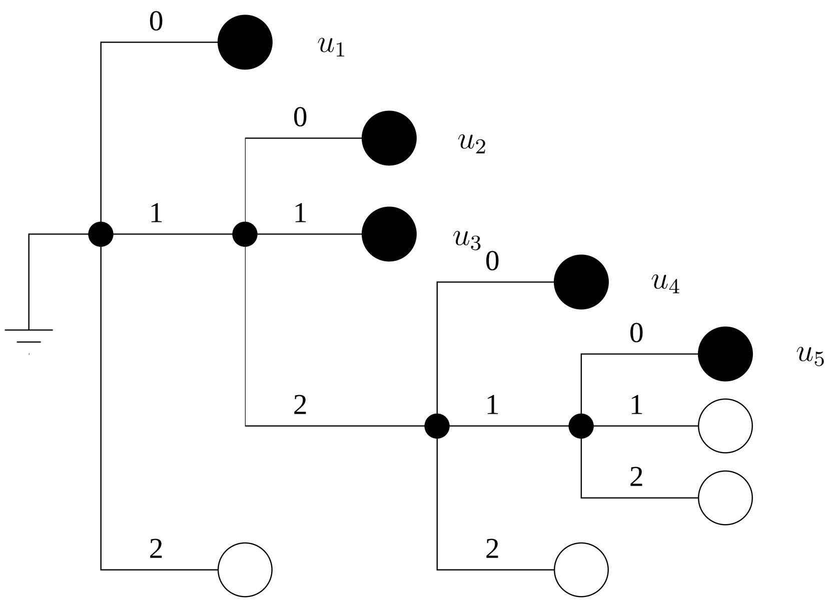


U	Z
u_1	00
u_2	01
u_3	10
u_4	110
u_5	1110

Example 2.4: Construct a binary prefix-free code with lengths $w_1 = 1, w_2 = 2, w_3 = 2, w_4 = 3, w_5 = 4$.

Since $\sum_{i=1}^{5} 2^{-w_i} = 1/2 + 1/4 + 1/4 + 1/8 + 1/16 = 19/16 > 1$, the Kraft inequality shows that no such code exists!

Example 2.5: Construct a ternary prefix-free code with codeword lengths $w_1 = 1, w_2 = 2, w_3 = 2, w_4 = 3, w_5 = 4$. Since $\sum_{i=1}^{5} 3^{-w_i} = 1/3 + 1/9 + 1/9 + 1/27 + 1/81 = 49/81 < 1$, we know such a prefix-free code exists. We "grow" such a code to obtain



U	Z
u_1	0
u_2	10
u_3	11
u_4	120
u_5	1210

The reader should reflect on the fact that we have yet to concern ourselves with the probabilities of the codewords in the prefix-free codes that we have constructed, although our goal is to code U so as to minimize $\mathrm{E}[W]$. From (2.1), it is obvious that we should assign the shorter codewords to the more probable values of U. But how do we know what codeword lengths to use? And what is the smallest $\mathrm{E}[W]$ that we can achieve? We shall return to these questions shortly, but first we shall look more carefully into the information-theoretic aspects of rooted trees.

2.2.2 Rooted Trees with Probabilities – Path Length and Uncertainty

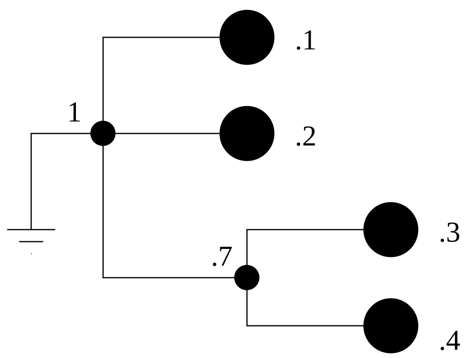
By a *rooted tree with probabilities* we shall mean a finite rooted tree with nonnegative numbers (probabilities) assigned to each vertex such that:

(1) the root is assigned probability 1, and

(2) the probability of every node (including the root) is the sum of the probabilities of the nodes and leaves at depth 1 in the subtree stemming from this intermediate node.

Note that we do not here require the tree to be "D-ary", i.e., to have the same number of branches stemming from all intermediate nodes.

Example 2.6:



This is a rooted tree with probabilities, but it is neither a ternary tree nor a binary tree.

Notice that, in a rooted tree with probabilities, the sum of the probabilities of the leaves must be one.

> *Path Length Lemma:* In a rooted tree with probabilities, the average depth of the leaves is equal to the sum of the probabilities of the nodes (including the root).

Proof: The probability of each node equals the sum of the probabilities of the leaves in the subtree stemming from that node. But a leaf at depth d is in the d such subtrees corresponding to the d nodes (including the root) on the path from the root to that leaf. Thus, the sum of the probabilities of the nodes equals the sum of the products of each leaf's probability and its depth, but this latter sum is just the average depth of the leaves according to (2.1). □

In the preceding example, the average depth of the leaves is $1 + .7 = 1.7$ by the Path Length Lemma. As a check, note that $1(.1) + 1(.2) + 2(.3) + 2(.4) = 1.7$.

We can think of the probability of each vertex in a rooted tree with probabilities as the *probability that we could reach that vertex in a random one-way journey* through the tree, starting at the root and ending on some leaf. Given that one is at a specified node, the conditional probability of choosing each outgoing branch as the next leg of the journey is just the probability on the node or leaf at the end of that branch divided by the probability of the node at its beginning (i.e., of the specified node itself). For instance, in Example 2.6, the probability of moving to the leaves with probability .3 and .4, given that one is at the node with probability .7, is 3/7 and 4/7, respectively.

We now consider some uncertainties that are naturally defined for a rooted tree with probabilities. Suppose that the rooted tree has T leaves whose probabilities are $p_1, p_2, \ldots, p_T$. Then we define the *leaf entropy* of the rooted tree as the quantity

$$H_{leaf} = -\sum_{i:p_i \neq 0} p_i \log p_i. \tag{2.3}$$

Notice that H_{leaf} can be considered as the uncertainty $H(U)$ of the random variable U whose value specifies the leaf reached in the random journey described above.

Suppose next that the rooted tree has N nodes (including the root) whose probabilities are $P_1, P_2, \ldots, P_N$. (Recall from the Path Length Lemma that $P_1 + P_2 + \cdots + P_N$ equals the average length, in branches, of the random journey described above.) We now wish to define the *branching* entropy at each of these nodes so that it equals the uncertainty of a random variable that specifies the branch followed out of that node, given that we had reached that node on our random journey. Suppose that $q_{i1}, q_{i2}, \ldots, q_{iL_i}$ are the probabilities of the nodes and leaves at the ends of the L_i branches stemming outward from the node whose probability is P_i. Then the branching entropy H_i, at this node is

$$H_i = -\sum_{j:q_{ij} \neq 0} \frac{q_{ij}}{P_i}\ log \frac{q_{ij}}{P_i}, \tag{2.4}$$

because q_{ij}/P_i is the conditional probability of choosing the j^{th} of these branches as the next leg on our journey given that we are at this node.

Example 2.7: Suppose that the $T = 4$ leaves and the $N = 2$ nodes for the second rooted tree of Example 2.6 have been numbered such that $p_1 = .1, p_2 = .2, p_3 = .3, p_4 = .4, P_1 = 1, P_2 = .7$.

Then

$$H_{leaf} = -\sum_{i=1}^{4} p_i \log p_i = 1.846 \text{ bits}.$$

We see that $L_1 = 3$ and $q_{11} = .1, q_{12} = .2, q_{13} = .7$. Thus

$$H_1 = -.1 \log .1 - .2 \log .2 - .7 \log .7 = 1.157 \text{ bits}.$$

Finally, we see that $L_2 = 2$ and $q_{21} = .3$ and $q_{13} = .7$. Thus

$$H_2 = -\frac{3}{7} \log \frac{3}{7} - \frac{4}{7} \log \frac{4}{7} = h\left(\frac{3}{7}\right) = .985 \text{ bits}.$$

Because of part (2) of the definition of a rooted tree with probabilities, we see that

$$P_i = \sum_{j=1}^{L_i} q_{ij}. \tag{2.5}$$

Using (2.5) together with $\log(q_{ij}/P_i) = \log q_{ij} - \log P_i$ in (2.4), we obtain for the product $P_i H_i$

$$P_i H_i = -\sum_{j:q_{ij} \neq 0} q_{ij} \log q_{ij} + P_i \log P_i. \tag{2.6}$$

We shall make use of (2.6) to prove the following fundamental result.

Theorem 2.1 (Leaf Entropy Theorem): The leaf entropy of a rooted tree with probabilities equals the sum over all nodes (including the root) of the branching entropy of that node weighted by the node probability, i.e.

$$H_{leaf} = \sum_{i=1}^{N} P_i H_i. \tag{2.7}$$

Example 2.8: Continuing Example 2.7, we calculate H_{leaf} by (2.7) to obtain

$$\begin{aligned} H_{leaf} &= (1)H_1 + (.7)H_2 \\ &= 1.157 + (.7)(.985) \\ &= 1.846 \text{ bits} \end{aligned}$$

in agreement with the direct calculation of Example 2.7.

Proof of Theorem 2.1: From (2.6), we see that the k-th node, if it is not the root, will contribute $+P_k \log P_k$ to the term in the sum in (2.7) with $i = k$, but will contribute $-P_k \log P_k$ to the term for i such that $q_{ij} = P_k$ (i.e., to the term for i such that node k is at the end of a branch leaving node i). Hence, the total contribution of all nodes, except the root, to the sum in (2.7) is 0. The root, say $i = 1$, contributes only the term $P_1 \log P_1$ to the sum in (2.7), but this is also 0 since $P_1 = 1$. Finally, from (2.6), we see that the k-th leaf contributes $-p_k \log p_k$ to the sum in (2.7), as it affects only the term for that i such that $q_{ij} = p_k$. Hence, we have proved that

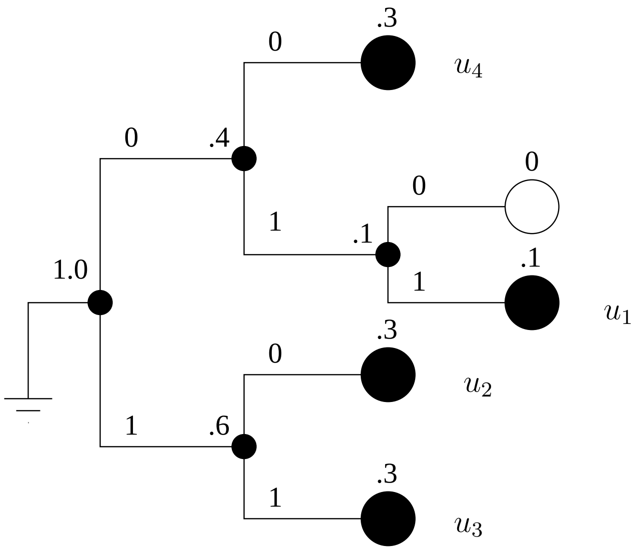
$$\sum_{i=1}^{N} P_i H_i = -\sum_{k=1}^{T} p_k \log p_k, \tag{2.8}$$

as was to be shown. □

2.2.3 Lower Bound on E[$\boldsymbol{W}$] for Prefix-Free Codes

We now use the results of the previous section to obtain a fundamental lower bound on the average codeword length, E[W], of a D-ary prefix-free code for a K-ary random variable U. We have already noted that such a prefix-free code defines a D-ary rooted tree in which each codeword corresponds to a leaf. The probability distribution P_U assigns probabilities to the codewords, and hence to the leaves, whose sum is 1. By way of convention, we assign probability 0 to any leaf that does not correspond to a codeword. We further assign to all nodes a probability equal to the sum of the probabilities of the nodes at depth 1 in the subtree stemming from this node. This now creates a D-ary rooted tree with probabilities.

Example 2.9: Taking $P_U(u_1) = .1$ and $P_U(u_2) = P_U(u_3) = P_U(u_4) = .3$ for the binary prefix-free code in Example 2.2 results in the following binary rooted tree with probabilities:



For the D-ary rooted tree with probabilities created by the construction just described, we see that

$$H_{leaf} = H(U), \tag{2.9}$$

i.e., the leaf entropy equals the uncertainty of the random variable being coded. Moreover, because D branches leave each node, it follows from the Corollary to Theorem 1.1 that the branching entropy at each node satisfies

$$H_i \leq \log D \tag{2.10}$$

with equality if and only if the next code digit is equally likely to be any of the D possibilities given that the previous code digits are those on the path to that node. Using (2.9) and (2.10) in (2.7) now gives

$$H(U) \leq \log D \sum_{i=1}^{N} P_i. \tag{2.11}$$

But, by the Path Length Lemma, the sum on the right of (2.11) is recognized to be the average depth of the leaves, i.e., the average codeword length $\mathrm{E}[W]$. We have thus proved the following fundamental bound.

Converse Part of the Coding Theorem for a Single Random Variable: The average codeword length, $\mathrm{E}[W]$, of any D-ary prefix-free code for a K-ary random variable U satisfies

$$\mathrm{E}[W] \geq \frac{H(U)}{\log D}, \tag{2.12}$$

where equality holds if and only if the code is optimal and the probability of each value of U is a negative integer power of D.

The bound (2.12) could possibly have been anticipated on intuitive grounds. It takes $H(U)$ bits of information to specify the value of U. But each D-ary digit of the codeword can, according to Theorems 1.1 and 1.3, provide at most $\log_2 D$ bits of information about U. Thus, we surely will need at least

$H(U)/\log_2 D$ code digits, on the average, to specify U. Such intuitive arguments are appealing and give insight; however, they are not substitutes for proofs.

The above theorem is our first instance where the answer to a technical question turns out to be naturally expressed in terms of Shannon's measure of information, but it is not yet a full justification of that measure as it specifies only a *lower bound* on $\mathrm{E}[W]$. (Trivially, $\mathrm{E}[W] \geq 1$ is a valid lower bound also, but we would not claim this bound as a justification for anything!) Only if the lower bound (2.12) is, in some sense, the best possible lower bound will Shannon's measure be justified. To show this, we need to show that there exist codes whose $\mathrm{E}[W]$ is arbitrarily close to the lower bound of (2.12).

2.2.4 Shannon-Fano Prefix-Free Codes

We now show how to construct "good", but in general non-optimum prefix-free codes. Perhaps the most insightful way to do this is to return for a moment to our discussion of Hartley's information measure in Section 1.1. We argued that, when the event $U = u_i$ occurs, it is as if one of $1/P_U(u_i)$ equally likely possibilities occurs. But to code L equally likely possibilities with equal length D-ary codewords, we need a codeword length of $\lceil \log_D L \rceil$ digits – where $\lceil x \rceil$ denotes the smallest integer equal to or greater than x. This suggests that the length, w_i, of the codeword for u_i ought to be chosen as

$$\begin{aligned} w_i &= \lceil \log_D \frac{1}{P_U(u_i)} \rceil \\ &= \lceil -\log_D P_U(u_i) \rceil \\ &= \lceil \frac{-\log P_U(u_i)}{\log D} \rceil. \end{aligned} \tag{2.13}$$

If $P_U(u_i) = 0$, (2.13) becomes meaningless. We resolve this quandry by agreeing here and hereafter that *we do not bother to provide codewords for values of U that have probability zero.*

Two questions now stand before us. First, does there exist a prefix-free code whose codeword lengths are given by (2.13)? If so, how small is $\mathrm{E}[W]$?

To answer these questions, we first note the obvious inequality

$$x \leq \lceil x \rceil < x + 1. \tag{2.14}$$

From (2.13), we can thus conclude that

$$w_i \geq -\log_D P_U(u_i) \tag{2.15}$$

so that, if U is a K-ary random variable,

$$\begin{aligned} \sum_{i=1}^{K} D^{-w_i} &\leq \sum_{i=1}^{K} D^{\log_D P_U(u_i)} \\ &= \sum_{i=1}^{K} P_U(u_i) = 1. \end{aligned} \tag{2.16}$$

Thus, Kraft's inequality ensures us that we can indeed find a D-ary prefix-free code whose codeword lengths are given by (2.13). To see how good our code is, we observe next that (2.13) and (2.14) imply that

$$w_i < \frac{-\log P_U(u_i)}{\log D} + 1. \tag{2.17}$$

Multiplying by $P_U(u_i)$ and summing over i gives, because of (2.1),

$$\mathrm{E}[W] < \frac{H(U)}{\log D} + 1. \tag{2.18}$$

We see that this method of coding, which is called *Shannon-Fano coding* since the technique is implicit in Shannon's 1948 paper but was first made explicit by Fano, gives us a prefix-free code whose average codeword length is within 1 digit of the lower bound (2.13) satisfied by *all* prefix-free codes. Thus, we have proved the following result.

The Coding Theorem for a K-ary Random Variable: The average codeword length of an optimum D-ary prefix-free code for a K-ary random variable U satisfies

$$\frac{H(U)}{\log D} \leq \mathrm{E}[W] < \frac{H(U)}{\log D} + 1 \tag{2.19}$$

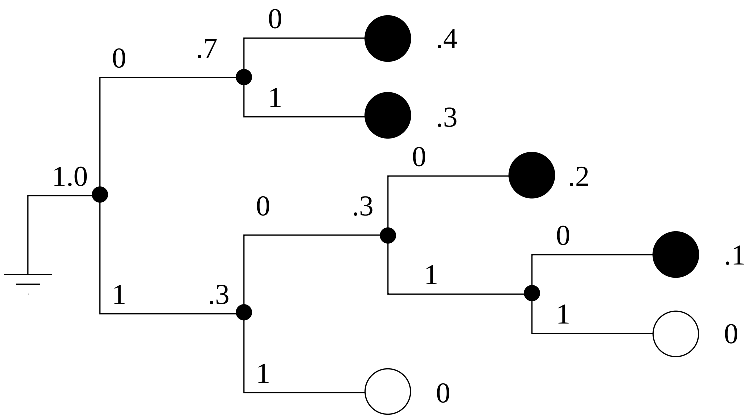
with equality on the left if and only if the probability of each value of U is some negative integer power of D. Moreover, $\mathrm{E}[W]$ for Shannon-Fano coding (which is generally non-optimum) also satisfies inequality (2.19) with equality if and only if the probability of each value of U is some negative integer power of D.

This coding theorem still does not quite give a complete justification of Shannon's measure of information because the upper bound cannot be made arbitrarily close to the lower bound. The complete justification must await our introduction in Section 2.3 of coding for an information source that emits a sequence of random variables.

Example 2.10: Consider binary Shannon-Fano coding for the 4-ary random variable U for which $P_U(u_i)$ equals .4, .3, .2 and .1 for i equal to 1, 2, 3 and 4, respectively. We first use (2.13) with $D = 2$ to compute the codeword lengths as

$$\begin{aligned}
w_1 &= \lceil \log_2 \frac{1}{.4} \rceil = 2 \\
w_2 &= \lceil \log_2 \frac{1}{.3} \rceil = 2 \\
w_3 &= \lceil \log_2 \frac{1}{.2} \rceil = 3 \\
w_4 &= \lceil \log_2 \frac{1}{.1} \rceil = 4.
\end{aligned}$$

We then "grow the code" by the algorithm given in Section (2.2.1) to obtain the code whose binary tree is



By the Path Length Lemma, we see that

$$\mathrm{E}[W] = 1 + .7 + .3 + .3 + .1 = 2.4$$

and a direct calculation gives

$$H(U) = 1.846 \text{ bits.}$$

We see indeed that (2.19) is satisfied. We note that our code, however, is clearly non-optimal. Had we simply used the 4 possible codewords of length 2, we would have had a better code ($\mathrm{E}[W] = 2$). Nonetheless, we know from our coding theorem that no code can beat the $\mathrm{E}[W]$ of the Shannon-Fano code by more than 1 digit. When $H(U)$ is large, Shannon-Fano coding is nearly optimal. But when $H(U)$ is small, we can generally do much better than Shannon-Fano coding.

2.2.5 Huffman Codes – Optimal Prefix-Free Codes

We now show how to construct an optimum D-ary prefix-free code for a K-ary random value U. We assume that

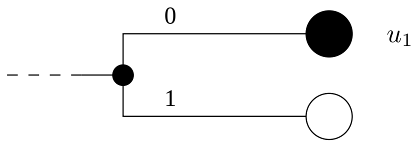
$$P_U(u_i) > 0 \qquad i = 1, 2, \ldots, K$$

so that we must assign a codeword to every possible value of U.

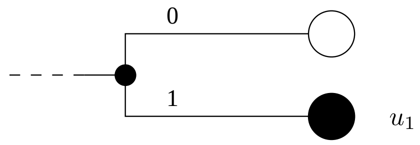
We first consider binary coding, i.e., $D = 2$. Two simple lemmas will give us the key to the construction of an optimum code.

Lemma 2.1: The binary tree of an optimum binary prefix-free code for U has no unused leaves.

Proof: Supposed that the tree has unused leaves. Because the code is optimal, these unused leaves must be at maximum depth in the tree. Then, for at least one value u_i of U, we have the situation



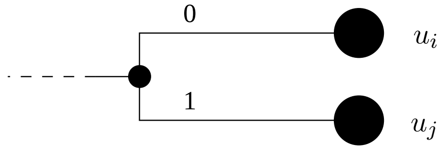
or



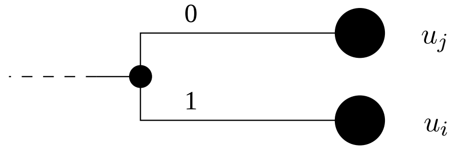
In either case, we can delete the last digit of the codeword for u_i (without changing the other codewords) and still have a prefix-free code. But the new code has smaller E[W] and thus the original code could not have been optimal. □

> *Lemma 2.2:* There is an optimal binary prefix-free code for U such that the two least likely codewords, say those for u_{K-1} and u_K, differ only in their last digit.

Proof: Assume $P_U(u_{K-1}) \geq P_U(u_K)$. Let z_i be one of the *longest* codewords in an optimum code for U. Then, because by Lemma 2.1 there can be no unused leaves in the code tree, we must have the situation

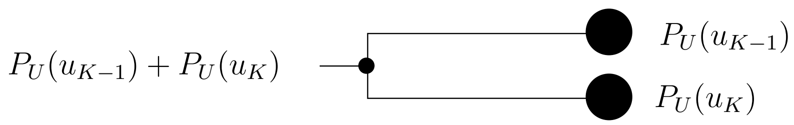


or



where u_j is some other value of the random variable U being coded. Now, if $j \neq K$, we switch the leaves for u_j and u_K. This cannot increase E[W] since $P_U(u_j) \geq P_U(u_K)$ and $w_j \geq w_K$. If $i \neq K-1$, we also switch the nodes for u_i and u_{K-1}. This cannot increase E[W] by a similar argument. Because the original code was optimal, so is the new code. But the new optimum code has its two least likely codewords differing only in their last digit. □

Because of Lemma 2.1 and the Path Length Lemma, we see that the construction of an optimum binary prefix-free code for the K-ary random variable U is equivalent to *constructing a binary tree with K leaves such that the sum of the probabilities of the nodes is minimum* when the leaves are assigned the probabilities $P_U(u_i)$ for $i = 1, 2, \ldots, K$. But Lemma 2.2 tells us how we may choose one node in an optimum code tree, namely as

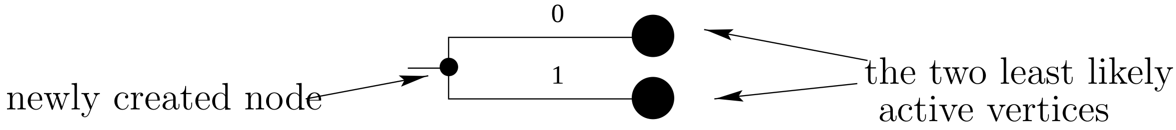


where u_{K-1} and u_K are the two least likely values of U. But, if we should now prune our binary tree at this node to make it a leaf with probability $P_U(u_{K-1}) + P_U(u_K)$, it would become one of $K-1$ leaves in a new tree. Completing the construction of the optimum code would then be equivalent to constructing a binary tree with these $K-1$ leaves such that the sum of the probabilities of the nodes is minimum. Again Lemma 2.2 tells us how to choose the one node in this new tree. Etc. We have thus proved the validity of the following:

Algorithm, due to Huffman, for constructing an optimal binary prefix-free code for a K-ary random variable U such that $P(u) \neq 0$, all u.

Step 0: Designate K vertices (which will be the leaves in the final tree) as $u_1, u_2, \ldots, u_K$ and assign probability $P_U(u_i)$ to vertex u_i for $i = 1, 2, \ldots, K$. Designate these K vertices as "active".

Step 1: Create a node that ties together the two least likely active vertices with binary branches in the manner



Deactivate these two active vertices, activate the new node and give it probability equal to the sum of the probabilities of the two vertices just deactivated.

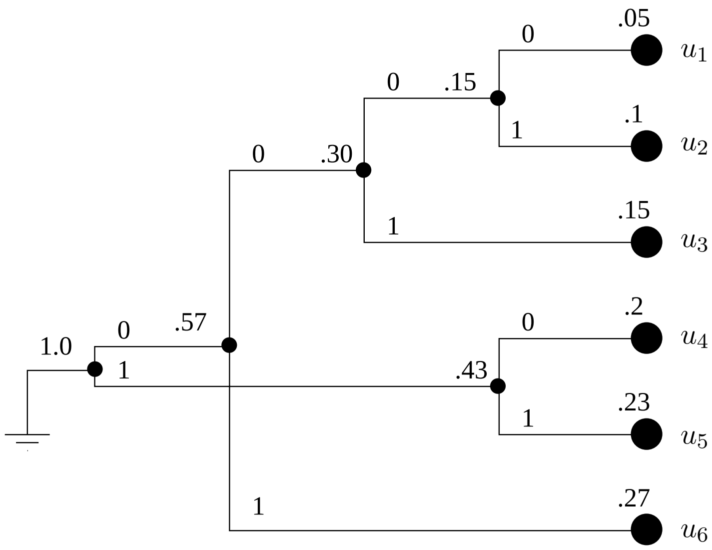
Step 2: If there is now only one active vertex, then ground this vertex (i.e., make it the root) and stop. Otherwise, go to step 1.

Example 2.11: Consider the random variable U such that

u	u_1	u_2	u_3	u_4	u_5	u_6
$P_U(u)$	.05	.1	.15	.2	.23	.27

.

An optimal binary prefix-free code for U is the following:



U	Z
u_1	0000
u_2	0001
u_3	001
u_4	10
u_5	11
u_6	01

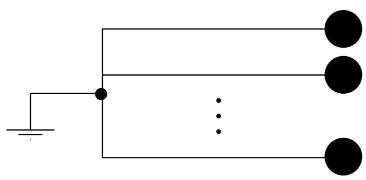
Notice that $\mathrm{E}[W] = 2(.2 + .23 + .27) + 3(.15) + 4(.1 + .05) = 1.4 + .45 + .6 = 2.45$.

Also, $H(U) = -\sum_{i=1}^{6} P_U(u) \log_2 P_U(u) = 2.42$ bits, so that (2.18) is indeed satisfied.

We now consider the generalization of the previous results for $D = 2$ to the case of arbitrary D. First, we prove an often useful result.

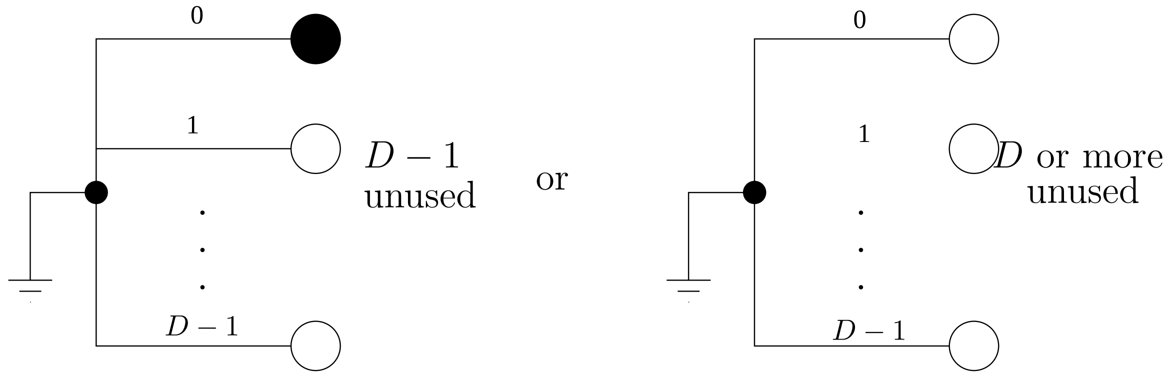
> *Node-Counting Lemma:* The number of leaves in a finite D-ary tree is always given by $D+q(D-1)$ where q is the number of nodes not counting the root.

Proof: Consider building such a tree from the root. At the first step, we get



which has D leaves. At each step thereafter, when we "extend" any leaf we get D new leaves but lose one old one (the one extended) for a net gain of $D-1$ leaves, and we gain one non-root node. The lemma follows. □

Now consider the tree for an optimal D-ary code for U. Such a tree can have no unused leaves except at the maximum length, because if there were such a leaf we could decrease $\mathrm{E}[W]$ by transferring one of the codewords at maximum length to this leaf. Moreover, if there are $D-1$ or more unused leaves at the maximum length, we could collect $D-1$ or D of these unused nodes on one node as



Thus, we can shorten the codeword by removing its last digit in the former case, or we can create an unused node at length less than maximum in the latter case. But, in either case, the code could not have been optimal. We conclude:

> *Lemma 2.0':* There are at most $D-2$ unused leaves in the tree of an optimal prefix-free D-ary code for U and all are at maximum length. Moreover, there is an optimal D-ary code for U that has all the unused leaves stemming from the same previous node.

Let r be the number of unused leaves. Then if U has K values, we have

$$r = \begin{bmatrix} \text{number of leaves in the} \\ D\text{-ary tree of the code} \end{bmatrix} - K.$$

From Lemma 2.0', we know that $0 \le r < D-1$ if the code is optimal. It follows from the node-counting lemma that

$$r = [D + q(D-1)] - K$$

or

$$D - K = -q(D-1) + r \qquad \text{where } 0 \le r < D-1. \tag{$*$}$$

It follows then, from Euclid's division theorem for integers that r is the *remainder* when $D-K$ is divided by $D-1$. (The *quotient* is $-q$.) This is not a convenient characterization of r since $D-K$ will usually be negative. However, adding $(K-D)(D-1)$ to both sides in $(*)$ gives

$$(K-D)(D-2) = (K-D-q)(D-1) + r \quad \text{where } 0 \le r < D-1$$

so Euclid tells us that r is also the remainder when $(K-D)(D-2)$ is divided by $D-1$.

Thus, we have proved the following result.

Lemma 2.1': The number of unused leaves in the tree of an optimal D-ary prefix-free code for a random variable U with K possible values, $K \geq D$, is the remainder when $(K - D)(D - 2)$ is divided by $D - 1$.

Arguments entirely similar to those we used in the binary case would give, *mutatis mutandis*, the next result.

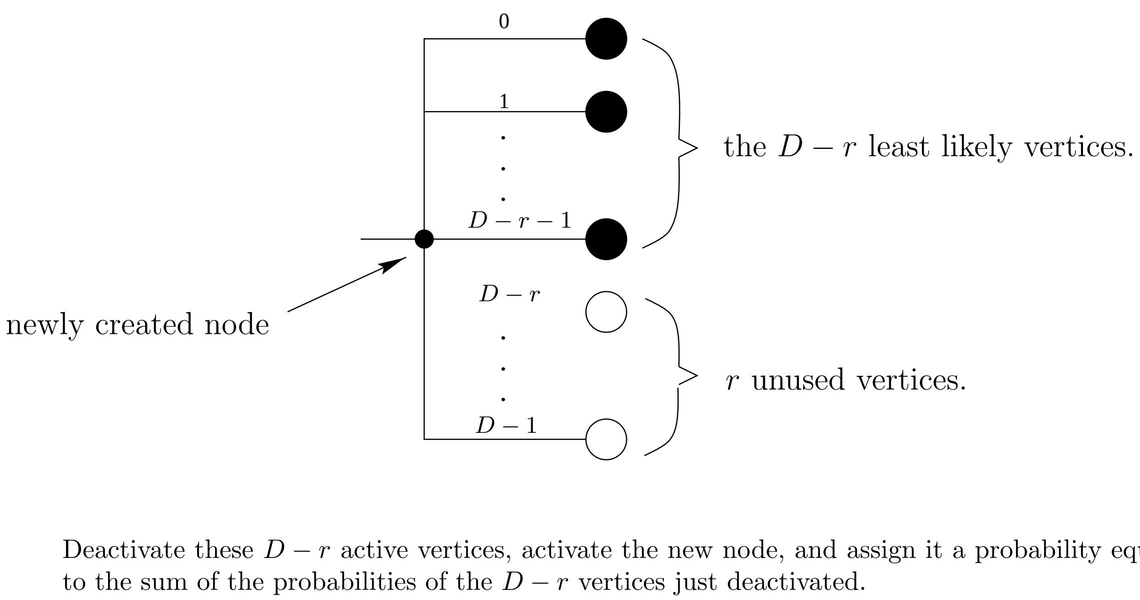
Lemma 2.2': There is an optimal D-ary prefix-free code for a random variable U with K possible values such that the $D - r$ least likely codewords differ only in their last digit, where r is the remainder when $(K - D)(D - 2)$ is divided by $D - 1$.

Lemma 2.2' tells us how to choose one node in the D-ary tree of an optimum prefix-free code for U. If we prune the tree at this node, Lemma 2.0' tells us that there will be no unused leaves in the resulting D-ary tree. The Path Length Lemma then tells us that constructing an optimum code is equivalent to constructing a D-ary tree with these $K + r - (D - 1)$ leaves that minimizes the sum of probabilities of the nodes. Lemma 2.2' now tells us how to choose one node in this new tree with $K + r - D + 1$ leaves, namely by creating a node from which stem the D least likely of these leaves. Etc. We have just justified the following:

Algorithm, due to Huffman, for constructing an optimal D-ary prefix-free code ($D \geq 3$) for a K-ary random variable U such that $P(u) \neq 0$ for all u.

Step 0: Designate K vertices (which will be the used leaves in the final tree) as $u_1, u_2, \ldots, u_K$ and assign probability $P_U(u_i)$ to vertex u_i for $i = 1, 2, \ldots, K$. Designate these K vertices as "active". Compute r as the remainder when $(K - D)(D - 2)$ is divided by $D - 1$.

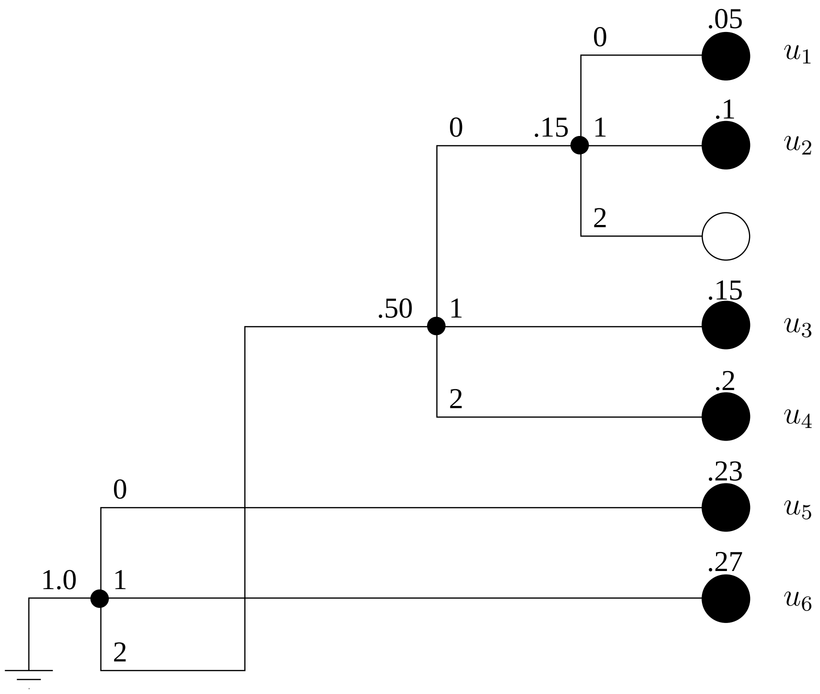
Step 1: Tie together the $D - r$ least likely active vertices with $D - r$ branches of a D-ary branch in the manner



Deactivate these $D - r$ active vertices, activate the new node, and assign it a probability equal to the sum of the probabilities of the $D - r$ vertices just deactivated.

Step 2: If there is only one active vertex, then ground this vertex (i.e., make it the root) and stop. Otherwise, set $r = 0$ and go to step 1.

Example 2.12: For the same U as in the previous example and $D = 3$, we have r = remainder when $(K - D)(D - 2) = (6 - 3)(3 - 2) = 3$ is divided by $D - 1 = 2$, i.e., $r = 1$. We construct an optimum code as



U	Z
u_1	200
u_2	201
u_3	21
u_4	22
u_5	0
u_6	1

$$\begin{aligned} \text{Note: } \mathrm{E}[W] &= 3(.16) + 2(.35) + 1(.50) = 1.65 \\ \frac{H(U)}{\log D} &= \frac{2.42}{1.59} = 1.52 \end{aligned}$$

2.3 Coding an Information Source

2.3.1 The Discrete Memoryless Source

Most "information sources" do not emit a single random variable, but rather a *sequence* of random variables. In fact, we implicitly recognized this fact when we required our codes for random variables to be prefix-free, because this requirement ensures us that we could immediately recognize the end of a codeword when it appeared in a sequence of codewords. We now make precise the first type of "sequential" information source that we will consider.

> *Definition:* A K-ary *discrete memoryless source* (or DMS) is a device whose output is a semi-infinite sequence $U_1, U_2, \ldots$ of statistically independent and identically distributed (i.i.d.) K-ary random variables.

A DMS is mathematically the simplest kind of information source to consider and is an appropriate model for certain practical situations. However, many actual information sources have memory in the sense that successive output letters are *not* statistically independent of the previous output letters. Before taking on sources with memory, it seems wise to become familiar with the simpler memoryless source.

2.3.2 Parsing an Information Source

The first question that arises in coding any information source, memoryless or not, is: what should one actually encode, i.e., to what objects should one assign codewords? We obviously cannot wait forever to see the entire semi-infinite source output sequence $U_1, U_2, \ldots$ before we begin our coding. The answer to this question is that we should first collect some source output digits and assign a codeword to this sequence, then collect some further source letters and assign them a codeword, etc. In Figure 2.4, we show the two devices that correspond to this manner of source encoding, namely the *source parser* and the *message encoder.*

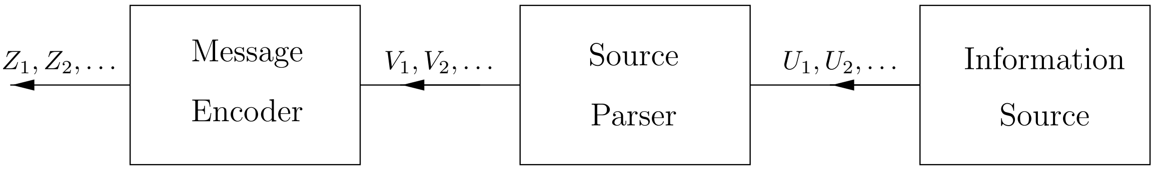


Figure 2.4: General Scheme for Coding an Information Source

The function of the *source parser* in Figure 2.4 is to divide the source output sequence into messages, which are the objects that are then assigned *codewords* by the *message encoder.* We write $V_1, V_2, \ldots$ to denote the semi-infinite sequence of messages from the source parser. The simplest, and very useful, source parser is the *L-block parser* for which each message consists of the next L source output digits, i.e.,

$$\begin{aligned} V_1 &= [U_1, U_2, \ldots, U_L] \\ V_2 &= [U_{L+1}, U_{L+2}, \ldots, U_{2L}] \\ \text{etc.} & \end{aligned}$$

We can show the separation of the source output sequence into messages performed by the L-block parser as follows:

$$U_1, U_2, \ldots, U_L \,\Big|\, U_{L+1}, U_{L+2}, \ldots, U_{2L} \,\Big|\, U_{2L+1}, \ldots$$

where the vertical bar indicates the end of a message. This way of showing the function of the L-block parser perhaps explains why the term "parsing" is an appropriate description. We shall later see that "variable-length" source parsers are also of interest.

2.3.3 Block-to-Variable-Length Coding of a DMS

We will now consider coding of a DMS. In keeping with our earlier convention not to provide codewords for values of a random variable having probability zero, we make the corresponding convention, for the K-ary DMS's that we will encode, that *all K values of an output digit have non-zero probability.*

We can now consider a message V formed by a source parser to be a random variable that we can encode in the manner that we considered at length in Section 2.2. By *block-to-variable-length D-ary coding of a K-ary DMS*, we mean that the message parser for the K-ary DMS is the *L-block* message parser (for some specified L) and that the encoder does a *variable-length* prefix-free D-ary encoding of each message formed by that parser, e.g., a Huffman coding or a Shannon-Fano encoding.

When the output of a K-ary DMS is parsed by the L-block message parser, then the i.i.d. nature of the source output sequence $U_1, U_2, \ldots$ guarantees that the parser output sequence $V_1, V_2, \ldots$ is also i.i.d., i.e., is a sequence of independent and identically-distributed K^L-ary random variables. We can therefore with no loss of essential generality consider the coding of the first message in this sequence which we denote simply by V, i.e., the message

$$V = [U_1, U_2, \ldots, U_L].$$

We have now reached the point where we can show that Shannon's measure of information gives precisely the right answer to the question: How many D-ary digits per source letter are required on the average to represent the output sequence of a particular K-ary DMS?

> *The Block-to-Variable-Length Coding Theorem for a DMS:* There exists a D-ary prefix-free coding of an L-block message from a DMS such that the average number of D-ary code digits per source letter satisfies
>
> $$\frac{\mathrm{E}[W]}{L} < \frac{H(U)}{\log D} + \frac{1}{L}, \tag{2.20}$$
>
> where $H(U)$ is the uncertainty of a single source letter. Conversely, for every D-ary prefix-free coding of an L-block message,
>
> $$\frac{\mathrm{E}[W]}{L} \geq \frac{H(U)}{\log D}. \tag{2.21}$$

Proof: Because the message $V = [U_1, U_2, \ldots, U_L]$ has L i.i.d. components, it follows from (1.31) and Theorem 1.2 that

$$\begin{aligned} H(V) &= H(U_1) + H(U_2) + \cdots + H(U_L) \\ &= LH(U), \end{aligned} \tag{2.22}$$

where the first equality follows from the independence, and the second equality follows from the identical distribution, of $U_1, U_2, \ldots, U_L$. Thus, we can immediately apply the above theorem to conclude that, for any D-ary prefix-free code for V,

$$\mathrm{E}[W] \geq \frac{H(V)}{\log D} = \frac{LH(U)}{\log D}$$

from which (2.21) follows. Conversely, we can apply (2.19) to conclude that there exists a prefix-free code for V such that

$$\mathrm{E}[W] < \frac{H(V)}{\log D} + 1 = \frac{LH(U)}{\log D} + 1.$$

This, upon dividing by L, establishes (2.20). □

It should be clear that an *optimum* block-to-variable-length coding scheme [in the sense of minimizing $\mathrm{E}[W]$, the average number of D-ary letters per source letter, over all D-ary prefix-free codes for V] can be obtained by applying the Huffman algorithm to the random variable V considered in the preceding proof.

2.3.4 Variable-Length to Block Coding of a DMS

The variable length codewords that we have considered to this point are sometimes inconvenient in practice. For instance, if the codewords are stored in a computer memory, one would usually prefer to use codewords whose length coincides with the computer wordlength. Or if the digits in the codeword are to be transmitted synchronously (say, at the rate of 2400 bits/sec), one would usually not wish to buffer variable length codewords to assure a steady supply of digits to be transmitted. But it was precisely the variability of the codeword lengths that permitted us to encode efficiently a single random variable, such as the message V from an L-block parser for a DMS. How can we get similar coding efficiency for a DMS when all codewords are forced to have the same length? The answer is that we must assign codewords not to blocks of source letters but rather to variable-length sequences of source letters, i.e., we must do *variable-length parsing.*

We now consider the coding of a variable number of source letters into an N-block codeword as indicated in Figure 2.5.

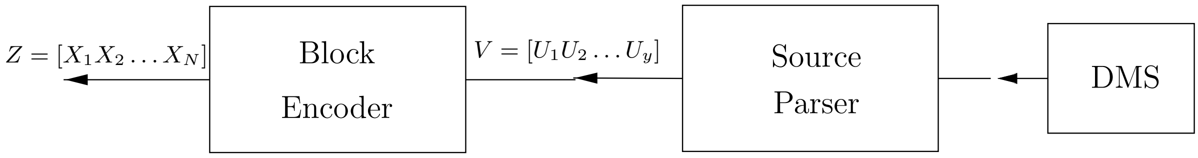
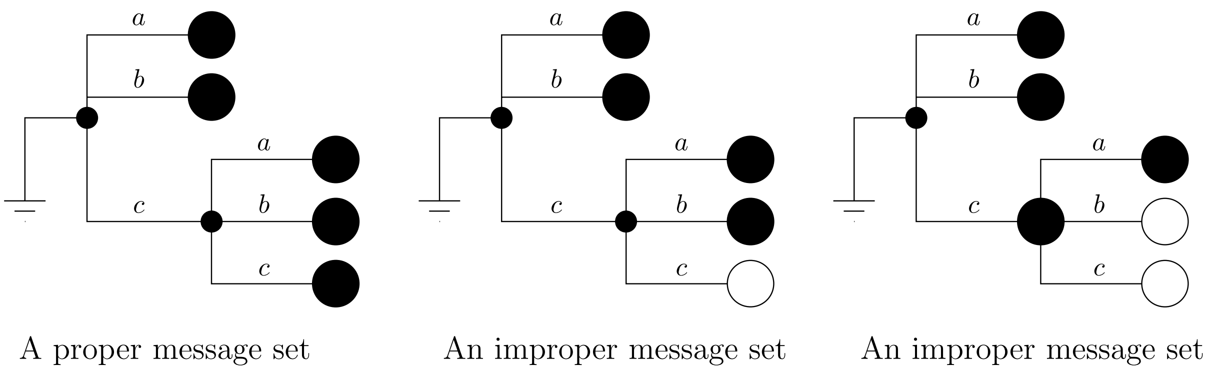


Figure 2.5: Variable-Length-to-Block Coding of a Discrete Memoryless Source

Here, the D-ary codewords all have length N, but the length, Y, of the message V to which the codewords are assigned is a random variable. The source parser simply stores letters from the DMS until it sees that these letters form a valid message. The criterion of goodness is $\mathrm{E}[Y]$, the average message length. Notice that $N/\mathrm{E}[Y]$ is the average number of D-ary code digits per source letter – thus, we should like to make $\mathrm{E}[Y]$ as large as possible.

In order that we can always reconstruct the source output sequence from the codewords, it is necessary and sufficient that every sufficiently long source sequence have some message as a prefix. In order that the message former be able to recognize a message as soon as it is complete, it is necessary and sufficient that no message be the prefix of a longer message. This motivates the definition of a *proper message set* for a K-ary source as a set of messages that form a complete set of leaves for a rooted K-ary tree.

Example 2.13: $K = 3$, source alphabet $= \{a, b, c\}$



The second of these message sets cannot be used in a source parser since there would be no valid parsing of $U_1, U_2, \ldots = c, c, \ldots$ as a sequence of messages. The third message set, however, could be used, perhaps with the rule: always choose the longest message if more than one parsing is possible.

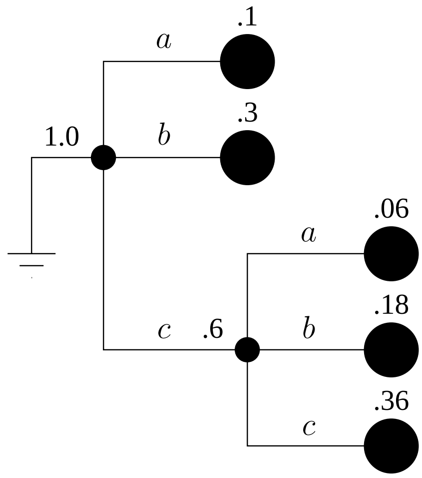
It should be noted that the sequence $V_1, V_2, \ldots$ of messages from the message-former is itself an i.i.d. sequence when the message set is proper. This follows from the fact that the message-former recognizes the last letter in each message without looking further into the source output sequence; this, together with the fact that the source is a DMS, implies that the source letters that form V_i are statistically independent from those that form $V_1, V_2, \ldots, V_{i-1}$ and have the same distribution as those for V_1. Thus, when a proper message set is used, it suffices to consider coding of only the first message, which we have already called V.

We can assign probabilities to the K-ary rooted tree corresponding to a proper message set by assigning probability 1 to the root and, to each subsequent vertex, assigning a probability equal to the probability of the node from which it stems multiplied by the probability that the DMS emits the letter on the branch connecting that node to this vertex. In this way, the probabilities of the leaves will be just the probabilities that the DMS emits these messages, and hence the leaf entropy will be $H(V)$, the uncertainty of the message, i.e.,

$$H_{leaf} = H(V). \tag{2.23}$$

Example 2.14: Suppose the DMS is ternary with $P_U(a) = .1$, $P_U(b) = .3$ and $P_U(c) = .6$.

Then this DMS creates the following ternary rooted tree with probabilities for the proper message set given in Example 2.13:



.

Because all nodes in the message tree have branching entropy equal to $H(U)$, i.e.,

$$H_i = H(U), \qquad \text{all } i, \tag{2.24}$$

Theorem 2.1 can now be applied to give

$$H(V) = H(U) \sum_{i=1}^{N} P_i \tag{2.25}$$

where P_i is the probability of the i-th node. Applying the Path Length Lemma to (2.25) gives the following fundamental result.

Theorem 2.2: The uncertainty, $H(V)$, of a proper message set for a K-ary DMS with output uncertainty $H(U)$ satisfies

$$H(V) = \mathrm{E}[Y]H(U), \tag{2.26}$$

where $\mathrm{E}[Y]$ is the average message length.

Example 2.15: For the DMS of Example 2.14, the output uncertainty is $H(U) = -.1\log(.1) - .3\log(.3) - .6\log(.6) = 1.295$ bits.

By the Path Length Lemma applied to the tree in Example 2.14, we find $\mathrm{E}[Y] = 1.6$ letters. It follows from Theorem 2.2 that the uncertainty of this proper message set is

$$H(V) = (1.6)(1.295) = 2.073 \text{ bits.}$$

The reader is invited to check this result directly from the probabilities ($.1, .3, .06, .18$ and $.36$) of the five messages in this message set.

We now wish to establish a "converse to the coding theorem" for variable-length-to-block encoding of a DMS. But, since a block code is a very special type of prefix-free code, we might just as well prove the following stronger result:

General Converse to the Coding Theorem for a DMS: For any D-ary prefix-free encoding of any proper message set for a DMS, the ratio of the average codeword length, $\mathrm{E}[W]$, to the average message length, $\mathrm{E}[Y]$, satisfies

$$\frac{\mathrm{E}[W]}{\mathrm{E}[Y]} \geq \frac{H(U)}{\log D} \tag{2.27}$$

where $H(U)$ is the uncertainty of a single source letter.

Proof: Letting V be the message to which the codeword is assigned, we have from Theorem 2.2 that

$$H(V) = \mathrm{E}[Y]H(U) \tag{2.28}$$

and from Theorem 2.1 that

$$\mathrm{E}[W] \geq \frac{H(V)}{\log D}. \tag{2.29}$$

Combining (2.28) and (2.29), we obtain (2.27). □

Although the above proof is quite trivial, there is a more subtle aspect to this converse theorem; namely, why is $\mathrm{E}[W]/\mathrm{E}[Y]$ the appropriate measure of the "average number of code digits used per source letter" rather than $\mathrm{E}[W/Y]$? The answer comes from consideration of the "law of large numbers". Suppose we let $Y_1, Y_2, Y_3, \ldots$ be the sequence of message lengths for the sequence $V_1, V_2, V_3, \ldots$ of messages, and we let $W_1, W_2, W_3, \ldots$ be the sequence of codeword lengths for the corresponding sequence $Z_1, Z_2, Z_3, \ldots$ of codewords. Because $V_1, V_2, V_3, \ldots$ is an i.i.d. sequence, the sequences $Y_1, Y_2, Y_3, \ldots$ and $W_1, W_2, W_3, \ldots$ are each i.i.d. Thus, the weak law of large numbers implies that

$$\operatorname*{plim}_{n\to\infty} \frac{Y_1 + Y_2 + \cdots + Y_n}{n} = \mathrm{E}[Y] \tag{2.30}$$

and

$$\operatorname*{plim}_{n\to\infty} \frac{W_1 + W_2 + \cdots + W_n}{n} = \mathrm{E}[W]. \tag{2.31}$$

But (2.30), (2.31) and the fact that $\mathrm{E}[Y] \neq 0$ imply that

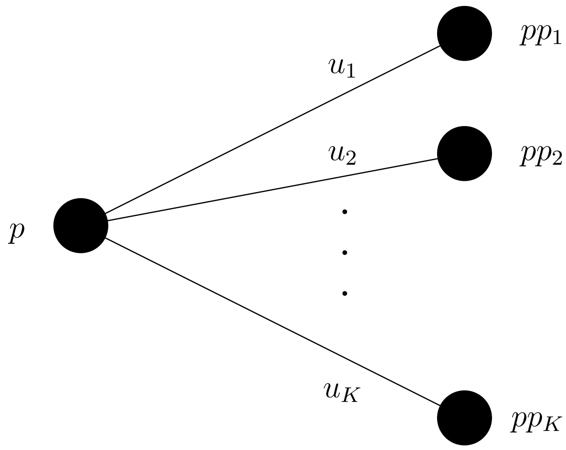
$$\operatorname*{plim}_{n\to\infty} \frac{W_1 + W_2 + \cdots + W_n}{Y_1 + Y_2 + \cdots + Y_n} = \frac{\mathrm{E}[W]}{\mathrm{E}[Y]}. \tag{2.32}$$

Equation (2.32) tells us that, after we have encoded a large number of messages, we can be virtually certain that the ratio of the total number of code digits we have used to code the total number of source letters will be virtually equal to $\mathrm{E}[W]/\mathrm{E}[Y]$. Thus, the ratio $\mathrm{E}[W]/\mathrm{E}[Y]$ is the quantity of *physical interest* that measures "code digits per source letter" and thus the quantity that we wish to minimize when we design a coding system for a DMS.

It is now time to develop the procedure to perform optimum variable-length-to-block encoding of a K-ary DMS. If the block length is chosen as N, then $\mathrm{E}[W] = N$ so that minimization of $\mathrm{E}[W]/\mathrm{E}[Y]$ is equivalent to maximization of the average message length, $\mathrm{E}[Y]$. We now consider how one forms a proper message set with maximum $\mathrm{E}[Y]$. From the node-counting lemma and the definition of a proper message set, it follows that the number, M, of messages in a proper message set for a K-ary DMS must be of the form

$$M = K + q(K-1) \tag{2.33}$$

for some nonnegative integer q. We shall form message sets for a DMS by "extending" nodes in a K-ary rooted tree with probabilities. By "extending a node", we mean the process of converting a leaf with probability p into a node by appending K branches to it as indicated in the following diagram:



where $u_1, u_2, \ldots, u_K$ are the possible values of the source letter U and where

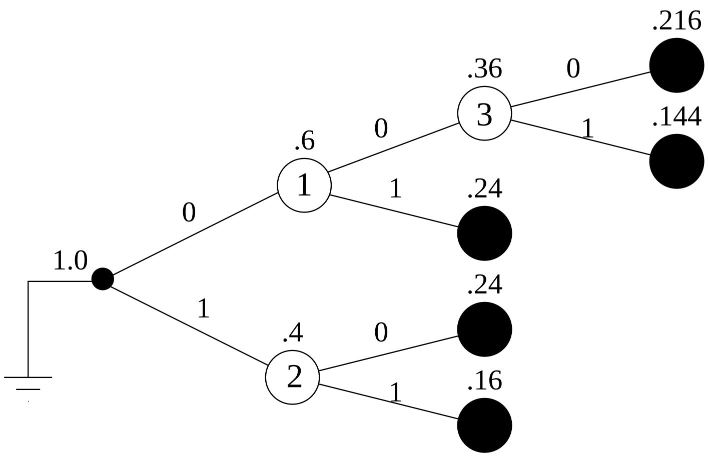
$$p_i = P_U(u_i) \tag{2.34}$$

is the probability that the next letter from the DMS will be u_i, given that the source has already emitted the digits on the branches from the root to the node that we have extended.

> *Definition:* A message set with $M = K + q(K-1)$ messages is a *Tunstall message set* for a K-ary DMS if the K-ary rooted tree can be formed, beginning with the extended root, by q applications of the rule: extend the most likely leaf.

Example: Consider the binary memoryless source (BMS) having $P_U(0) = p_1 = .6$ and $P_U(1) = p_2 = .4$. The unique Tunstall message with $M = 5$ messages corresponds to the tree:

where the nodes have been numbered to show the order in which they were extended by the rule of extending the most likely leaf. (There are two different Tunstall message sets with $M = 6$ messages for this source because there is a tie for the most likely leaf that would be extended next in the above tree.) By the path length lemma, we see that the average message length for this Tunstall message set is



$$\mathrm{E}[Y] = 1 + .60 + .40 + .36 = 2.36 \text{ letters.}$$

Notice that, for the Tunstall message set of the previous example, the probability (.24) of the most likely leaf does not exceed the probability (.36) of the least likely node. This is no accident!

> *The Tunstall Lemma:* A proper message set for a K-ary DMS is a Tunstall message set if and only if, in its K-ary rooted tree, every node is at least as probable as every leaf.

Proof: Consider growing a Tunstall message set by beginning from the extended root and repeatedly extending the most likely leaf. The extended root trivially has the property that no leaf is more likely than its only node (the root itself). Suppose this property continues to hold for the first i extensions. On the next extension, none of the K new leaves can be more likely than the node just extended and thus none is more likely than any of the old nodes since these were all at least as likely as the old leaf just extended. But none of the remaining old leaves is more likely than any old node nor more likely than the new node since this latter node was previously the most likely of the old leaves. Thus, the desired property holds also after $i + 1$ extensions. By induction, this property holds for every Tunstall message set.

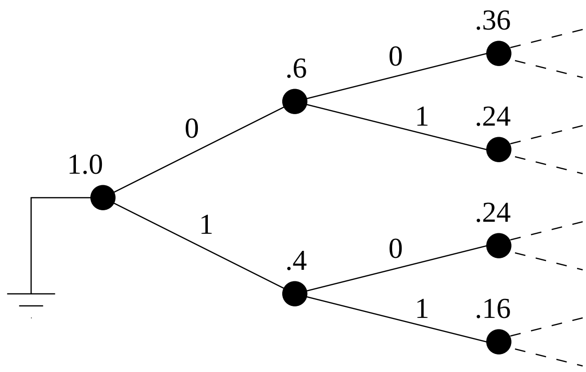
Conversely, consider any proper message set with the property that, in its K-ary tree, no leaf is more likely than any intermediate node. This property still holds if we "prune" the tree by cutting off the K branches stemming from the least likely node. After enough such prunings, we will be left only with the extended root. But if we then re-grow the same tree by extending leaves in the reverse order to our pruning of nodes, we will at each step be extending the most likely leaf. Hence this proper message set was indeed a Tunstall message set, and the lemma is proved. □

It is now a simple matter to prove:

> *Theorem 2.3* A proper message set with M messages for a DMS maximizes the average message length, $\mathrm{E}[Y]$, over all such proper message sets if and only if it is a Tunstall message set.

Proof: Consider the semi-infinite K-ary rooted tree for the DMS having each node labelled with the probability that the source emits the sequence of letters on the branches from the root to that node. For instance, for the BMS with $P_U(0) = .6$, this tree is

The key observations to be made are



(1) that all the vertices in the semi-infinite subtree extending from any given node are less probable than that node, and

(2) that all the vertices (both nodes and leaves) in the K-ary tree of a proper messages set for this DMS are also nodes with the same probabilities in this semi-infinite tree.

It then follows from the Tunstall lemma that a proper message set with $M = K + q(K-1)$ messages is a Tunstall message set if and only if its $q+1$ nodes are the $q+1$ most likely nodes in this semi-infinite tree. Thus, the sum of the probabilities of the nodes is the same for all Tunstall message sets with M messages and strictly exceeds the sum of the probabilities of the nodes in any non-Tunstall proper messages set with M messages. The theorem now follows from the Path Length Lemma. $\square$

The reader can now surely anticipate how Tunstall messages sets will be used to perform optimal variable-length-to-block coding of a DMS. The only question is how large the message set should be. Suppose that the block length N is specified, as well as the size D of the coding alphabet. There are then only D^N possible codewords, so the number M of messages must be no greater than D^N. We should choose M as large as possible since we want to maximize $\mathrm{E}[Y]$, the average message length. From (2.33), we see that we can increase M only in steps of size $K-1$. Thus, the largest value of M that we may use corresponds to the integer q such that

$$0 \leq D^N - M = D^N - K - q(K-1) < K - 1$$

or, equivalently, such that

$$D^N - K = q(K-1) + r \qquad \text{where } 0 \leq r < K-1. \tag{2.35}$$

But, thanks to Euclid, we now see from (2.35) that this value of q is nothing more than the *quotient* when $D^N - K$ is divided by $K-1$. [Note that $D^N \geq K$ is required because the smallest proper message set (namely, the extended root) has K messages]. Thus, with the aid of Theorem 2.3, we have now justified:

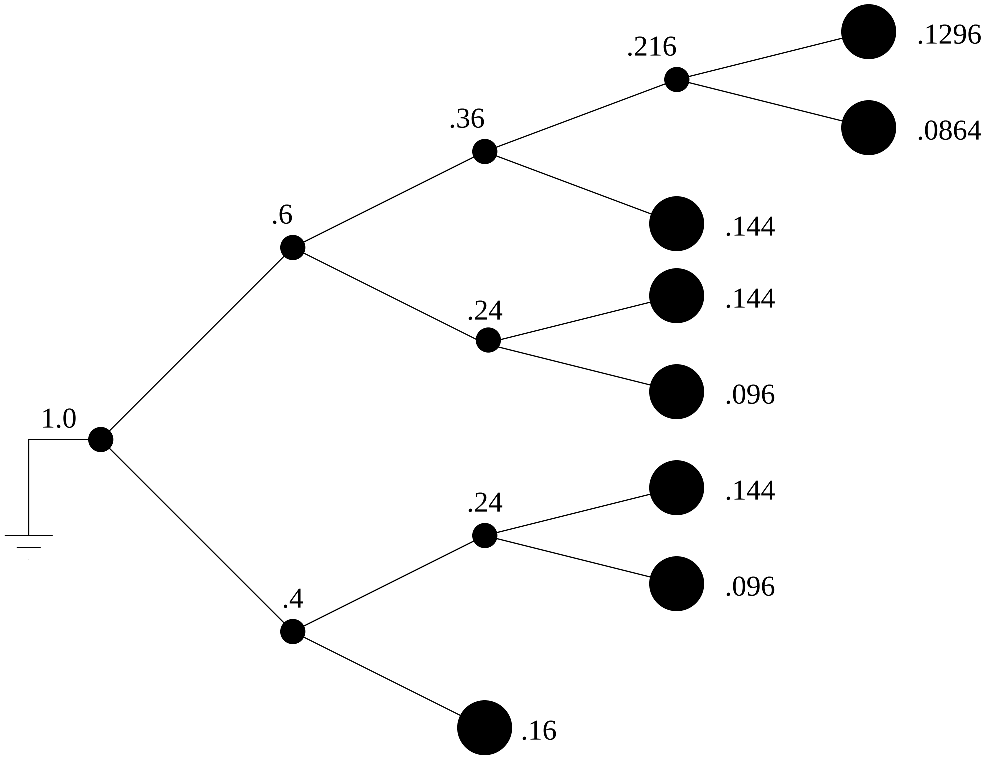
Tunstall's Algorithm for Optimum D-ary Block Encoding with Block-Length N of a Proper Message Set for a K-ary DMS with Output Variable U.

Step0: Check to see that $D^N \geq K$. If not, abort because no such coding is possible. Otherwise, calculate the quotient q when $D^N - K$ is divided by $K-1$.

Step1: Construct the Tunstall message set of size $M = K + q(K-1)$ for the DMS by beginning from the extended root and making q extensions of the most likely leaf at each step.

Step2: Assign a distinct D-ary codeword of length N to each message in the Tunstall message set.

Example 2.16: For the BMS ($K = 2$) with $P_U(0) = .6$, suppose we wish to do an optimum binary ($D = 2$) block coding with block length $N = 3$ of a proper message set. From Step 0 in the above algorithm, we find $q = 6$. Step 1 then yields the following Tunstall message set:



In accordance with Step 2, we now assign codewords.

message	codeword
0 0 0 0	0 0 0
0 0 0 1	1 1 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1	1 1 1

For convenience of implementation, we have chosen the codeword to be the same as the message for those messages of length $N = 3$, and as similar to the message as we could manage otherwise. By the path length lemma, we see that the average message length is

$$\mathrm{E}[Y] = 1 + .6 + .4 + .36 + .24 + .24 + .216 = 3.056$$

so that

$$\frac{N}{\mathrm{E}[Y]} = .982 \text{ code digits/source digit.}$$

The converse to the coding theorem for prefix-free coding of a proper message set tells us that the above ratio could not have been smaller than

$$\frac{H(U)}{\log D} = h(.4) = .971 \text{ code digits/source digit.}$$

Notice that if we had used "no coding at all" (which is possible here since the source alphabet and coding alphabet coincide), then we would have achieved trivially 1 code digit / source digit. It would be an economic question whether the 2% reduction in code digits offered by the Tunstall scheme in the above example was worth the cost of its implementation. In fact, we see that no coding scheme could "compress" this binary source by more than 3% so that Tunstall has not done too badly.

The converse to the coding theorem for prefix-free coding of a proper message set implies the upper bound of the following theorem.

The Variable-Length-to-Block Coding Theorem for a DMS: The ratio, $E[Y]/N$, of average message length to block length for an optimum D-ary block length N encoding of a proper message set for a K-ary DMS satisfies

$$\frac{\log D}{H(U)} - \frac{\log(2/p_{min})}{NH(U)} < \frac{E[Y]}{N} \leq \frac{\log D}{H(U)} \tag{2.36}$$

where $H(U)$ is the uncertainty of a single source letter and where $p_{min} = \min_u P_U(u)$ is the probability of the least likely source letter.

Our convention for coding a DMS implies that $p_{min} > 0$. We see then from (2.36) that we can make $E[Y]/N$ as close as we like to the upper bound $\log D/H(U)$. Thus, we have another affirmation of the fact that $H(U)$ is the true "information rate" of a DMS.

Proof of lower bound in (2.36): The least likely message in the message set has probability Pp_{min} where P is the probability of the node from which it stems; but since there are M messages, this least likely message has probability at most $1/M$ so we must have

$$Pp_{min} \leq \frac{1}{M}. \tag{2.37}$$

By the Tunstall lemma, no message (i.e., no leaf) has probability more than P so that (2.37) implies

$$P_V(v) \leq \frac{1}{(Mp_{min})}, \qquad \text{all } v,$$

which in turn implies

$$-\log P_V(v) \geq \log M - \log\left(\frac{1}{p_{min}}\right). \tag{2.38}$$

Next, we recall that the number $M = K + q(K-1)$ of messages was chosen with q as large as possible for D^N codewords so that surely

$$M + (K-1) > D^N. \tag{2.39}$$

But $M \geq K$ so that (2.39) further implies

$$2M \geq M + K > D^N,$$

which, upon substitution in (2.38), gives

$$H(V) > N \log D - \log\left(\frac{2}{p_{min}}\right). \tag{2.40}$$

Making use of Theorem 2.2, we see that (2.40) is equivalent to

$$\mathrm{E}[Y]H(U) > N \log D - \log\left(\frac{2}{p_{min}}\right). \tag{2.41}$$

Dividing now by $NH(U)$ gives the left inequality in (2.36). □

In spite of its fundamental importance, Tunstall's work was never published in the open literature. Tunstall's doctoral thesis (A. Tunstall, "Synthesis of Noiseless Compression Codes", Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, 1968), which contains this work, lay unnoticed for many years before it became familiar to information theorists.

Tunstall's work shows how to do optimal variable-length-to-block coding of a DMS, but only when one has decided to use a proper message set. A "quasi-proper" message set (such as the third message set in Example 2.13) in which there is at least one message on each path from the root to a leaf in its K-ary tree, but sometimes more than one message, can also be used. Quite surprisingly (as the exercises will show), one can sometimes do better variable-length-to-block coding of a DMS with a quasi-proper message set than one can do with a proper message set having the same number of messages, although one cannot use less than $(\log D)/H(U)$ source letters per D-ary code digit on the average. Perhaps even more surprisingly, nobody today knows how to do optimal variable-length-to-block coding of a DMS with a given number of messages when quasi-proper message sets are allowed.

2.4 Coding of Sources with Memory

2.4.1 Discrete Stationary Sources

Most real information sources exhibit substantial memory so that the DMS is not an appropriate model for them. We now introduce a model for sources with memory that is sufficiently general to model well most real information sources, but is still simple enough to be analytically tractable.

A K-ary *discrete stationary source* (DSS) is a device that emits a sequence $U_1, U_2, U_3, \ldots$ of K-ary random variables such that for every $n \geq 1$ and every $L \geq 1$, the random vectors $[U_1, U_2, \ldots, U_L]$ and $[U_{n+1}, U_{n+2}, \ldots, U_{n+L}]$ have the same probability distribution. In other words, for every window length L along the DSS output sequence, one will see the same statistical behavior regardless of where that window is placed along the DSS output sequence. Another way to say this is to say that the *statistics* of the DSS output sequence are *time-invariant.*

To specify a DSS, we must in principle specify the joint probability distribution $P_{U_1 U_2 \ldots U_L}$ for every $L \geq 1$. This is equivalent to specifying P_{U_1} and $P_{U_L|U_1 \ldots U_{L-1}}$ for all $L \geq 2$.

A DSS is said to have finite memory μ if, for all $n > \mu$,

$$P_{U_n|U_1 \ldots U_{n-1}}(u_n|u_1, \ldots, u_{n-1}) = P_{U_n|U_{n-\mu} \ldots U_{n-1}}(u_n|u_{n-\mu}, \ldots, u_{n-1}) \tag{2.42}$$

holds for all choices of $u_1, \ldots, u_n$ and if μ is the smallest nonnegative integer such that this holds. [By way of convention, one interprets $P_{U_n|U_{n-\mu} \ldots U_{n-1}}$ to mean just P_{U_n} when $\mu = 0$.]

The discrete memoryless source (DMS) is the special case of the discrete stationary source (DSS) with memory $\mu = 0$. It follows from (2.42) that, for a DSS with memory μ,

$$H(U_n \big| U_1 \dots U_{n-1}) = H(U_n \big| U_{n-\mu} \dots U_{n-1}) \tag{2.43}$$

holds for all $n > \mu$. [One interprets $H(U_n \big| U_1 \dots U_{n-1})$ to mean just $H(U_1)$ when $n = 1$.] In fact, (2.43) can alternatively be considered as the definition of finite memory μ for a DSS, because (2.42) is equivalent to (2.43), as follows from Corollary 2 to Theorem 1.2.

By now the reader will have come to expect that the "information rate" of a source is the "uncertainty per letter" of its output sequence. The obvious way to define the uncertainty per letter of the output sequence is first to define the uncertainty per letter of L letters as

$$H_L(U) = \frac{1}{L} H(U_1 U_2 \dots U_L) \tag{2.44}$$

and then to let L go to infinity. But there is another reasonable way to define the uncertainty per letter, namely as the uncertainty of one output letter given $L-1$ preceeding letters, i.e., as $H(U_L \big| U_1 \dots U_{L-1})$, and then to let L go to infinity. Which is the right notion? The following result shows that in fact both of these ways to define the "uncertainty per letter" of a DSS give the same value as L goes to infinity, i.e., both notions are correct!

Information-Theoretic Characterization of a DSS: If $U_1, U_2, U_3, \dots$ is the output sequence of a DSS, then

(i) $H(U_L \big| U_1 \dots U_{L-1}) \le H_L(U)$ for all $L \ge 1$;

(ii) $H(U_L \big| U_1 \dots U_{L-1})$ is non-increasing as L increases;

(iii) $H_L(U)$ is non-increasing as L increases; and

(iv) $\lim_{L\to\infty} H(U_L \big| U_1 \dots U_{L-1}) = \lim_{L\to\infty} H_L(U)$, i.e., both of these limits always exist and both have the same value, say $H_\infty(U)$.

Proof: We begin by proving (ii), which is the easiest place to start.

$$H(U_{L+1} \big| U_1 \dots U_L) \le H(U_{L+1} \big| U_2 \dots U_L) = H(U_L \big| U_1 \dots U_{L-1})$$

where the inequality follows because removing conditioning can only increase uncertainty and where the equality follows by the time invariance of the statistics of a DSS. This proves (ii), and we now use (ii) to prove (i).

$$\begin{aligned} H(U_1 U_2 \dots U_L) &= H(U_1) + H(U_2 \big| U_1) + \dots + H(U_L \big| U_1 \dots U_{L-1}) \\ &\ge L H(U_L \big| U_1 \dots U_{L-1}) \end{aligned}$$

where the inequality follows because (ii) implies that each term in the preceeding sum is at least as great as the last term; dividing by L in the inequality gives (i). We can now use (i) and (ii) to prove (iii) as follows:

$$
\begin{aligned}
H(U_1 \dots U_L U_{L+1}) &= H(U_1 \dots U_L) + H(U_{L+1} \big| U_1 \dots U_L) \\
&\le H(U_1 \dots U_L) + H(U_L \big| U_1 \dots U_{L-1}) \\
&\le H(U_1 \dots U_L) + H_L(U) \\
&= (L+1) H_L(U)
\end{aligned}
$$

where the first inequality follows from (ii) and the second inequality follows from (i); dividing by $L+1$ gives

$$
H_{L+1}(U) \le H_L(U)
$$

which is the claim of (iii).

Of course, $H_L(U) \ge 0$ and $H(U_L \big| U_1, \dots, U_{L-1}) \ge 0$ because uncertainties cannot be negative. But a non-increasing sequence bounded below always has a limit. Thus (ii) and (iii) ensure that both $\lim_{L\to\infty} H(U_L \big| U_1, \dots, U_{L-1})$ and $\lim_{L\to\infty} H_L(U)$ always exist. Moreover, (i) implies that

$$
\lim_{L\to\infty} H(U_L \big| U_1, \dots, U_{L-1}) \le \lim_{L\to\infty} H_L(U) \stackrel{\triangle}{=} H_\infty(U). \tag{2.45}
$$

The only task remaining is to show that in fact equality holds in this inequality. Here we need to use a bit of trickery. We begin with

$$
\begin{aligned}
H(U_1 \dots U_L \dots U_{L+n}) &= H(U_1 \dots U_L) + H(U_{L+1} \big| U_1 \dots U_L) + \dots + H(U_{L+n} \big| U_1 \dots U_{L+n-1}) \\
&\le H(U_1 \dots U_L) + n H(U_{L+1} \big| U_1 \dots U_L),
\end{aligned}
$$

where the inequality follows from the fact that (ii) implies $H(U_{L+1} \big| U_1, \dots, U_L)$ is the greatest of the n such conditional uncertainties in the preceeding sum; dividing by $n+L$ gives

$$
H_{L+n}(U) \le \frac{H(U_1 \dots U_L)}{n+L} + \frac{n}{n+L} H(U_{L+1} \big| U_1 \dots U_L).
$$

The trick now is to let n, rather than L, go to infinity, which gives

$$
H_\infty(U) \le H(U_{L+1} \big| U_1 \dots U_L), \tag{2.46}
$$

where we have used the fact that $H(U_1 \dots U_L) \le L \log K$ so that this term divided by $n+L$ vanishes as n goes to infinity. But it now follows from (2.46) that

$$
\lim_{L\to\infty} H(U_L \big| U_1 \dots U_{L-1}) \ge H_\infty(U). \tag{2.47}
$$

Together (2.45) and (2.47) show that equality holds in these two inequalties, which completes the proof. □

We have now seen that $H_\infty(U)$ is the natural way to define the "uncertainty per letter" of a DSS. Our experience up to now suggests that it should be possible to encode a DSS with D-ary letters, say with some sort of prefix-free code, in such a way that we use as close to $H_\infty(U)/\log D$ code digits per source letter on the average as we wish, but that we can use no less. In the remainder of this chapter, we show that this is indeed the case. Note that, for a DSS with finite memory μ, (2.43) shows that $H_\infty(U) = H(U_{\mu+1} \big| U_1 \dots U_\mu)$, which is again very natural.

2.4.2 Converse to the Coding Theorem for a DSS

For simplicity, we will consider only block parsing of the output sequence of a DSS, but variable-length parsers are also often used in practice. The situation that we consider is shown in figure 2.6.

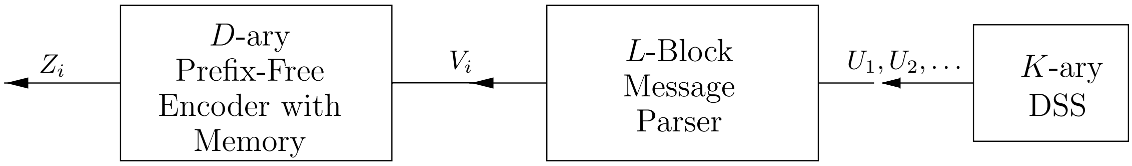


Figure 2.6: The situation for block-to-variable-length encoding of a DSS.

Because of the L-block message parser, the i-th message to be encoded is simply

$$V_i = [U_{iL-L+1}, \ldots, U_{iL-1}, U_{iL}]$$

for $i = 1, 2, \ldots$. Because the source has memory, however, it makes sense to allow the encoder to take previous messages into account when encoding the message V_i. In principle, the encoder could use a different prefix-free code for each possible sequence of values for the previous messages $V_1, V_2, \ldots, V_{i-1}$; the decoder would already have decoded these messages and thus would also know which code was used to give the codeword Z_i, which it then could decode as V_i.

Given that $[V_1, \ldots, V_{i-1}] = [v_1, \ldots, v_{i-1}]$, it follows from the converse to the coding theorem for a single random variable that the length W_i of the codeword Z_i for the message V_i must satisfy

$$\mathrm{E}\left[W_i \big| [V_1, \ldots, V_{i-1}] = [v_1, \ldots v_{i-1}]\right] \geq \frac{H(V_i \big| [V_1, \ldots, V_{i-1}] = [v_1, \ldots, v_{i-1}])}{\log D}.$$

Multiplying by $P_{V_1 \ldots V_{i-1}}(v_1, \ldots, v_{i-1})$ and summing over all values $v_1, \ldots, v_{i-1}$ gives

$$\mathrm{E}[W_i] \geq \frac{H(V_i \big| V_1 \ldots V_{i-1})}{\log D}, \tag{2.48}$$

where on the left side of the inequality we have invoked the theorem of "total expectation". But

$$\begin{aligned} H(V_i \big| V_1 \ldots V_{i-1}) &= H(U_{iL-L+1} \ldots U_{iL} \big| U_1 \ldots U_{iL-L}) \\ &= H(U_{iL-L+1} \big| U_1 \ldots U_{iL-L}) + \cdots + H(U_{iL} \big| U_1 \ldots U_{iL-1}) \\ &\geq LH(U_{iL} \big| U_1 \ldots U_{iL-1}) \\ &\geq LH_\infty(U), \end{aligned} \tag{2.49}$$

where the first inequality follows because, by part (ii) of the Information-Theoretic Characterization of a DSS, the last term is the smallest in the preceeding sum, and where the second inequality follows from parts (ii) and (iv). Substituting (2.49) into (2.48) gives the following fundamental result.

Converse to the Coding Theorem for Block-to-Variable-Length Encoding of a Discrete Stationary Source: If block length L messages from a DSS are encoded by a D-ary prefix-free code, where the code used for each message may depend on previous messages, then the length W_i of the codeword for the i-th message satisfies

$$\frac{\mathrm{E}[W_i]}{L} \geq \frac{H_\infty(U)}{\log D} \tag{2.50}$$

where $H_\infty(U)$ is the uncertainty per letter of the DSS.

2.4.3 Coding the Positive Integers

In this section we describe a very clever binary prefix-free code for the positive integers that we will later use within an efficient coding scheme for any DSS.

Everyone today is familiar with the "standard" binary representation of the positive integers as illustrated in the following short table:

n	$B(n)$
1	1
2	1 0
3	1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1
8	1 0 0 0

The first obvious feature of this "radix-two" code for the positive integers is that it is not prefix-free; in fact every codeword $B(n)$ is the prefix of infinitely many other codewords! Two other properties are of special interest: (1) every codeword has 1 as its first digit, and (2) the length $L(n)$ of the codeword $B(n)$ is given by

$$L(n) = \lfloor \log_2 n \rfloor + 1. \tag{2.51}$$

There is something intuitively pleasing about the fact that $L(n)$ is roughly equal to $\log_2 n$, but at this point it is hard to say why this is so natural. Nonetheless we would certainly by pleased, and perhaps surprised, to find that there was a binary *prefix-free* code for the positive integers such that the length of the codeword for n was also roughly equal to $\log_2 n$, and even more surprised, if this coding could be easily implemented. But precisely such a code has been given by Elias.

Elias began by first introducing a binary code $B_1(.)$ for the positive integers that is obviously prefix-free, but not at all obviously useful. The trick is simply to add a prefix of $L(n) - 1$ zeroes to the standard codeword $B(n)$ to form the codeword $B_1(n)$, as illustrated in the following short table.

n	$B_1(n)$
1	1
2	0 1 0
3	0 1 1
4	0 0 1 0 0
5	0 0 1 0 1
6	0 0 1 1 0
7	0 0 1 1 1
8	0 0 0 1 0 0 0

The number of leading zeroes in a codeword tells us exactly how many digits will follow the inevitable leading 1. Thus one knows where the codeword ends when it is followed by other codewords, which means the code is prefix-free. For instance the sequence of codewords 0100011000111... is unambiguously recognizable as the sequence of codewords for the positive integer sequence $2, 6, 7, \ldots$. The only problem is that the codewords are about twice as long as we would like. More precisely, the length $L_1(n)$ of the codeword $B_1(n)$ is given by

$$L_1(n) = 2\lfloor \log_2 n \rfloor + 1. \tag{2.52}$$

The second trick that Elias pulled out of his magician's hat was to use his first code to signal the number of following digits in the codeword, then to append the natural code $B(n)$, whose leading 1 can now be discarded to gain efficiency. More precisely, the codeword $B_2(n)$ in this *second Elias code* for the positive integers begins with $B_1(L(n))$, followed by $B(n)$ with the leading 1 removed. Again we give a short table.

n	$B_2(n)$
1	1
2	0 1 0 0
3	0 1 0 1
4	0 1 1 0 0
5	0 1 1 0 1
6	0 1 1 1 0
7	0 1 1 1 1
8	0 0 1 0 0 0 0 0

Because the code $B_1(.)$ is prefix-free, we can recognize the first part $B_1(L(n))$ of the codeword $B_2(n)$ as soon as we see the last digit of $B_1(L(n))$, and hence we know the number $L(n)$ of digits in the remainder of the codeword. Thus the coding $B_2(.)$ is indeed prefix-free. It is also easy to obtain the natural code $B(n)$ from $B_2(n)$, we just place a 1 in front of the second part of the codeword.

How efficient is this second Elias prefix-free code $B_2(n)$ for the integers? The length $L_2(n)$ of $B_2(n)$ is given by

$$\begin{aligned} L_2(n) &= L_1(L(n)) + L(n) - 1 \\ &= L_1(\lfloor \log_2 n \rfloor + 1) + \lfloor \log_2 n \rfloor \\ &= \lfloor \log_2 n \rfloor + 2\lfloor \log_2(\lfloor \log_2 n \rfloor + 1) \rfloor + 1, \end{aligned} \tag{2.53}$$

where we have made use of (2.51) and (2.52). Because $\log_2(\log_2 n)$ becomes negligible compared to $\log_2 n$ for large positive integers n, we see from (2.53) that Elias has done the seemingly impossible, namely given an easily implementable, binary, prefix-free code for the positive integers in which the length of the codeword for n is roughly $\log_2 n$. Of course, this code works best when n is large, but it isn't at all bad for rather small n. The reader is invited to make a table of $B(n), B_1(n)$ and $B_2(n)$ for $1 \leq n \leq 64$, which will confirm that

$$L_2(n) > L_1(n) \text{ only for } 2 \le n \le 15,$$
$$L_2(n) = L_1(n) \text{ for } n = 1 \text{ and } 16 \le n \le 31,$$
$$L_2(n) < L_1(n) \text{ for all } n \ge 32.$$

We will soon see how Elias' second coding of the integers can be used for efficient binary prefix-free coding of any DSS.

2.4.4 Elias-Willems Source Coding Scheme

Figure 2.7 shows the essentials of the Elias-Willems source coding scheme for a K-ary DSS.

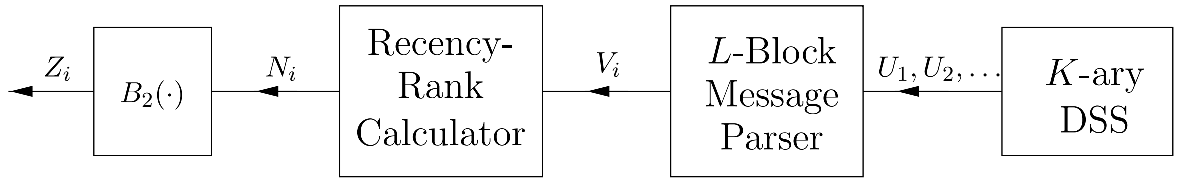


Figure 2.7: The Elias-Willems Source Coding Scheme for a DSS.

The only part of this source coding scheme that we have not seen before is the device that we have called the "recency-rank calculator". To explain this device, we suppose first that the scheme has been operating for a long time, i.e., $i \gg 1$ and that every one of the K^L possible values of an L-block message has occurred as the value of some past message $V_j, 1 \le j < i$. Then the message value most recently seen has recency rank 1, the next different message value most recently seen has recency rank 2, etc. As a simple example, suppose that $L = 2$ so that the four possible message values are [00], [01], [10], and [11]. Suppose that the sequence of past messages is

$$\ldots \Big| V_{i-2} \Big| V_{i-1} = \ldots |11|00|11|10|01|11|01|01$$

Then the message [01] has recency rank 1, the message [11] has recency rank 2, [10] has recency rank 3, and [00] has recency rank 4. The *recency-rank calculator* of the Elias-Willems scheme assigns to the message V_i the recency rank N_i of the value of this message in the sequence of past messages. Continuing our example, we see that if $V_i = [10]$ then $N_i = 3$ would be the output of the recency-rank calculator. At the next step, [10] would have recency rank 1, etc. Of course, when coding first begins and for some time afterward, not all message values will have a genuine recency rank. This is not a problem of real importance and can be simply solved by giving all message values an artificial, arbitrary recency rank when coding begins – when a particular message value appears it will acquire an honest recency rank and continue to have a genuine recency rank at all future times.

It remains to determine how efficiently the Elias-Willems scheme encodes a DSS. To this end, we suppose that encoding has been in progress for a time sufficiently long for all message values to have appeared at least once in the past so that all recency ranks are genuine. We now define the *time* Δ_i *to the most recent occurrence* of V_i in the manner that if $V_i = v$, then $\Delta_i = \delta$ where $V_{i-\delta}$ is the most recent past message with this same value v. In our previous example with $V_i = [10]$, we have $\Delta_i = 5$ since V_{i-5} is the most recent past message with value [10]. Recall, however, that $N_i = 3$ for this example. We next note that

$$N_i \le \Delta_i \tag{2.54}$$

always holds, since only the values of $V_{i-1}, \dots, V_{i-\Delta_i+1}$ (which might not all be different) could have smaller recency ranks than the value v of V_i.

We now make the assumption that the DSS being coded is *ergodic*, i.e., that its "time statistics" are the same as its "probabilistic statistics". This means that if one looks at a very long output sequence $U_1, U_2, \dots, U_{N+L-1}$, then the fraction of the N windows $[U_{i+1}, U_{i+2}, \dots, U_{i+L}], 0 \le i < N$, in which a particular pattern $[a_1, a_2, \dots, a_L]$ appears is virtually equal to $P_{U_1 \dots U_L}(a_1, \dots, a_L)$ for all $L \ge 1$ and all choices of $a_1, \dots, a_L$. Ergodicity is a technical requirement that is not of much importance – most interesting real information sources are ergodic. Our only reason for assuming that the DSS is ergodic is so that we can make the simple conclusion that

$$\mathrm{E}[\Delta_i \big| V_i = v] = \frac{1}{P_V(v)} \tag{2.55}$$

where $V = [U_1, \dots, U_L]$. This follows from the fact that, because the fraction of messages equal to v must be $P_V(v)$ by ergodicity, the average spacing between these messages must be $1/P_V(v)$ messages. [We have also tacitly used the obvious fact that the ergodicity of the DSS output sequence $U_1, U_2, \dots$ guarantees the ergodicity of the message sequence $V_1, V_2, \dots$.]

Because the binary codeword Z_i for the message V_i is just the second Elias prefix-free codeword $B_2(N_i)$ for the recency rank N_i, it follows that its length W_i is

$$W_i = L_2(N_i) \le L_2(\Delta_i) \tag{2.56}$$

because of (2.54) and the fact that $L_2(n)$ is non-decreasing as n increases. Weakening (2.53) slightly gives

$$L_2(n) \le \log_2 n + 2\log_2(\log_2 n + 1) + 1$$

which we can now use in (2.56) to obtain

$$W_i \le \log_2(\Delta_i) + 2\log_2(\log_2(\Delta_i) + 1) + 1. \tag{2.57}$$

Taking the conditional expectation, given that $V_i = v$, results in

$$\mathrm{E}[W_i | V_i = v] \le \log_2(\mathrm{E}[\Delta_i | V_i = v]) + 2\log_2\big(\log_2(\mathrm{E}[\Delta_i | V_i = v]) + 1\big) + 1 \tag{2.58}$$

where our taking the expectation inside the logarithms is justified by Jensen's inequality and the fact that $\log_2(.)$ [and thus also $\log_2(\log_2(.))$] is convex-$\cap$ [see Supplementary Notes at the end of Chapter 4]. We now use (2.55) in (2.58) to obtain

$$\mathrm{E}[W_i | V = v] \le -\log_2(P_V(v)) + 2\log_2(-\log_2(P_V(v)) + 1) + 1.$$

Multiplying by $P_V(v)$, summing over v, and invoking the theorem of "total expectation" gives

$$\mathrm{E}[W_i] \le H(V) + 2\log_2(H(V) + 1) + 1 \tag{2.59}$$

where we have made one more use of Jensen's inequality and the convexity-$\cap$ of $\log_2(.)$. The final step is to note that

$$\begin{aligned} H(V) &= H(U_1 \dots U_L) \\ &= L H_L(U). \end{aligned} \tag{2.60}$$

Substituting (2.60) into (2.59) and dividing by L gives the following result.

Elias-Willems Block-to-Variable-Length Coding Theorem for a DSS: For the Elias-Willems binary coding scheme for block length L messages from a K-ary ergodic discrete stationary source, the codeword length W_i for all i sufficiently large satisfies

$$\frac{\mathrm{E}[W_i]}{L} \leq H_L(U) + \frac{2}{L}\log_2(LH_L(U) + 1) + \frac{1}{L}. \tag{2.61}$$

By choosing L sufficiently large, $\mathrm{E}[W_i]/L$ can be made arbitrarily close to H_∞.

In fact, (2.61) holds even if the DSS is not ergodic, but the proof is then slightly more complicated because we cannot make use of the convenient (2.55).

The Elias-Willems source coding scheme, unlike all our previous source coding schemes, is *universal*, i.e., it works very well for all sources in some large class and not only for a particular source to which it has been tailored. In a very real sense, the Elias-Willems scheme learns about the particular DSS that it is encoding from the very process of encoding. Such universality is, of course, a very important practical feature of a source coding scheme.

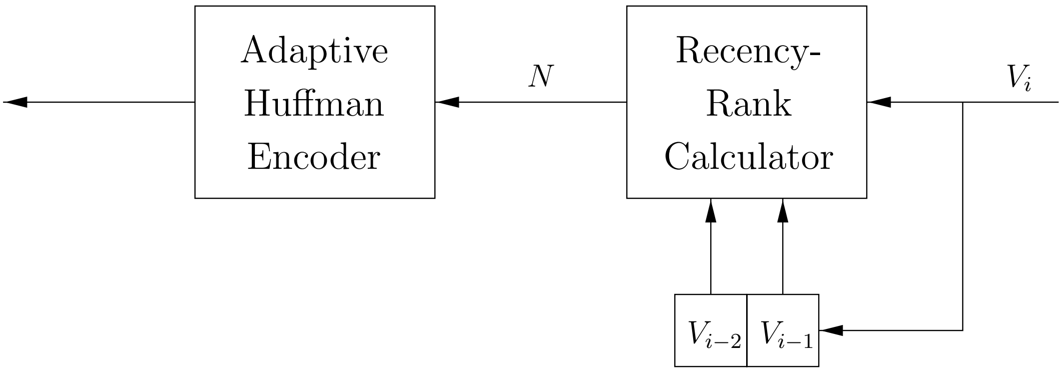


Figure 2.8: A practical source coding scheme based on the Elias-Willems scheme.

Not only is the Elias-Willems scheme asymptotically optimal as L goes to infinity, it can also be modified to be a practical and efficient way of encoding real information sources. In practical versions (cf. P. Portmann & I. Spieler, *"Generalized Recency-Rank Source Coding"*, Sem. Arb. am ISI, WS 1994/95), it turns out to be advisable to keep a different recency-rank list for each possible value of M previous messages as indicated in Figure 2.8 for $M = 2$. It also turns out to be wise to replace the Elias $B_2(.)$ encoder of Figure 2.7 with an "adaptive Huffman encoder", i.e., with a Huffman code designed for the observed frequency of occurence of past inputs. For $L = 8$ and $M = 2$, such a scheme has been verified to compress a wide variety of real information sources as well as any other universal coding scheme in use today.

Chapter 3

THE METHODOLOGY OF TYPICAL SEQUENCES

3.1 Lossy vs. Lossless Source Encoding

The reader will perhaps have noted that in the previous chapter we considered three different ways to encode a discrete memoryless source (DMS), namely:

(1) block-to-variable-length encoding,

(2) variable-length-to-block encoding, and

(3) variable-length-to-variable-length encoding.

This raises the obvious question: Why did we not also consider block-to-block encoding? This question has both an obvious answer and a very subtle one.

The obvious answer is that block-to-block coding of a DMS is not interesting. Figure 3.1 shows the coding situation. A message is a block of L letters from the K-ary DMS; a codeword is a block of N D-ary letters. Assuming that $P_U(u) > 0$ for all K possible values of the DMS output U, we see that the *smallest* N that suffices to provide a different codeword for each of the K^L messages (all having non-zero probability) is determined by the inequalities

$$D^{N-1} < K^L \leq D^N$$

or, equivalently,

$$N = \left\lceil L \frac{\log K}{\log D} \right\rceil . \tag{3.1}$$

The striking conclusion is that this smallest blocklength N depends only on the alphabet sizes K and D and on the message length L. The uncertainty $H(U)$ of the output U of the DMS is irrelevant. Shannon's theory of information seems to say nothing interesting about the conceptually simple problem of block-to-block encoding of a DMS, at least nothing obvious.

The subtle answer to the question of why we have ignored block-to-block encoding is that a satisfactory treatment of this type of source coding requires us to make a distinction that the other three types of source encoding do not require, namely the distinction between "lossless" and "lossy" encoding.

A *lossless* source encoding scheme is one in which it is always possible to make an exact reconstruction of the source output sequence from the encoded sequence. All of the source coding schemes considered

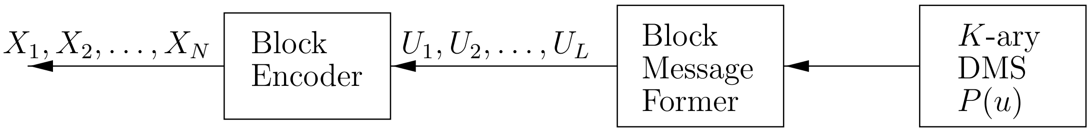


Figure 3.1: Block-to-block encoding of a discrete memoryless source.

in the previous chapter were lossless. A *lossy* source coding scheme is one in which one cannot always make such a perfect reconstruction. But if the probability of an incorrect reconstruction is sufficiently small, a lossy source encoding scheme is generally quite acceptable in practice.

To treat lossy source coding, we shall use one of the most powerful tools introduced by Shannon in his 1948 paper, namely, the notion of a "typical" sequence.

3.2 An Example of a Typical Sequence

The idea of a "typical" sequence can perhaps best be illustrated by an example. Consider a sequence of $L = 20$ binary digits emitted by a binary memoryless source with

$$P_U(0) = \frac{3}{4} \qquad \text{and} \qquad P_U(1) = \frac{1}{4}.$$

We will now give a list of three sequences, one of which was actually the output sequence of such a source, but the other two of which were artificially chosen. The reader is asked to guess which was the "real" output sequence. The three sequences are:

(1) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.

(2) 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1.

(3) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.

Unless he or she thinks too much about this problem, the reader will surely have correctly guessed that (2) was the real output sequence. But more thought might have led the reader to reject this obvious (and correct!) answer. By the definition of a DMS, it follows that

$$\begin{aligned} P_{U_1,\ldots,U_{20}}(u_1,\ldots,u_{20}) &= \left(\frac{3}{4}\right)^z \left(\frac{1}{4}\right)^{20-z} \\ &= \left(\frac{1}{4}\right)^{20} (3)^z \end{aligned}$$

where z is the number of zeroes in the sequence $u_1, u_2, \ldots, u_{20}$. Because $z = 0$ for sequence (1) whereas $z = 14$ for sequence (2), it follows that the DMS is $3^{14} = 4,782,969$ times as likely to emit sequence (2) as to emit sequence (1). But $z = 20$ for sequence (3) and we are thus forced to conclude that the DMS is $3^{20}/3^{14} = 3^6 = 729$ times as likely to emit sequence (3) as to emit sequence (2). Why then does our intuition correctly tell us that sequence (2) was the real output sequence from our DMS rather than the overwhelmingly more probable sequence (3)?

The "law of large numbers" says roughly that if some event has probability p then the fraction of times that this event will occur in many independent trials of the random experiment will, with high probability, be close to p. The event that our DMS emits a 1 has probability 1/4, while the event that it emits a 0 has probability 3/4. The fraction of times that these events occur within the 20 trials of the experiment that yielded sequence (2) is $6/20 = 3/10$ and $14/20 = 7/10$, respectively. In sequence (3), these fractions are

0 and 1, respectively. Thus sequence (2) conforms much more closely to the law of large numbers than does sequence (3). Our intuition correctly tells us that the DMS is virtually certain to emit a sequence whose composition of 0's and 1's will conform to the law of large numbers, even if there are other output sequences overwhelmingly more likely than any particular one of these "typical sequences". But if this is so, then in a block-to-block coding scheme we need only provide codewords for the typical source output sequences. This is indeed the right strategy and it makes block-to-block source coding very interesting indeed.

3.3 The Definition of a Typical Sequence

In his 1948 paper, Shannon spoke of "typical long sequences" but gave no precise definitions of such sequences. The wisdom of Shannon's refusal to be precise on this point has been confirmed by subsequent events. At least half a dozen precise definitions of typical sequences have been given since 1948, but none is clearly "better" than the others. For some purposes, one definition may be more convenient, but for other purposes less convenient. It is hard to escape the conclusion that Shannon was well aware in 1948 of a certain arbitrariness in the definition of typicality, and did not wish to prejudice the case by adopting any particular definition. The choice of definition for typical sequences is more a matter of taste than one of necessity. The definition that appeals to our taste and that we shall use hereafter differs slightly from those in the literature. It should hardly be necessary to advise the reader that whenever he or she encounters arguments based on typical sequences, he or she should first ascertain what definition of typicality has been chosen.

Consider a DMS with output probability distribution $P_U(u)$ and with a *finite* output alphabet $\{a_1, a_2, \ldots, a_K\}$, i.e., a K-ary DMS. (As we shall see later, our definition of typical sequences cannot be used for DMS's with a countably infinite output alphabet, but this is not an important loss of generality since any DMS of the latter type can be approximated as closely as desired by a DMS with a finite output alphabet.) Let $\underline{U} = [U_1, U_2, \ldots, U_L]$ denote an output sequence of *length* L emitted by this DMS and let $\underline{u} = [u_1, u_2, \ldots, u_L]$ denote a possible value of $\underline{U}$, i.e., $u_j \in \{a_1, a_2, \ldots, a_K\}$ for $1 \leq j \leq L$. Let $n_{a_i}(\underline{u})$ denote the number of occurrences of the letter a_i in the sequence $\underline{u}$. Notice that if $\underline{U} = \underline{u}$, then $n_{a_i}(\underline{u})/L$ is the fraction of times that the DMS emitted the output letter a_i in the process of emitting the L digits of the sequence $\underline{u}$. The law of large numbers says that this fraction should be close to the probability $P_U(a_i)$ of the event that the DMS will emit the output letter a_i on any one trial. We are finally ready for our definition. First we require that some *positive number* ε be given. Then we say that $\underline{u}$ is an *ε-typical output sequence of length L for this K-ary DMS* if

$$(1-\varepsilon)P_U(a_i) \leq \frac{n_{a_i}(\underline{u})}{L} \leq (1+\varepsilon)P_U(a_i), \qquad 1 \leq i \leq K. \tag{3.2}$$

We shall usually be thinking of ε as a very small positive number, but this is not required in the definition. When ε is very small, however, say $\varepsilon \approx 10^{-3}$, then we see that $\underline{u}$ is an ε-typical sequence just when its composition of a_1's, a_2's, ..., and a_K's conforms almost exactly to that demanded by the law of large numbers.

Example 3.1: Consider the binary DMS with $P_U(0) = 3/4$ and $P_U(1) = 1/4$. Choose $\varepsilon = 1/3$. Then a sequence $\underline{u}$ of length $L = 20$ is ε-typical if and only if both

$$\frac{2}{3} \cdot \frac{3}{4} \leq \frac{n_0(\underline{u})}{20} \leq \frac{4}{3} \cdot \frac{3}{4}$$

and

$$\frac{2}{3} \cdot \frac{1}{4} \leq \frac{n_1(\underline{u})}{20} \leq \frac{4}{3} \cdot \frac{1}{4}.$$

Equivalently, $\underline{u}$ is ε-typical if and only if both

$$10 \leq n_0(\underline{u}) \leq 20$$

and

$$4 \leq n_1(\underline{u}) \leq 6.$$

Notice that only sequence (2), which has $n_0(\underline{u}) = 14$ and $n_1(\underline{u}) = 6$, is ε-typical among the three sequences (1), (2) and (3) considered in Section 3.2. Notice also that if the second condition is satisfied, i.e., if $4 \leq n_1(\underline{u}) \leq 6$, then it must be the case that $14 \leq n_0(\underline{u}) \leq 16$ and thus the first condition will automatically be satisfied. This is an accident due to the fact that $K = 2$ rather than a general property. The reader can readily check that if the most probable value of U, say a_1, has probability $P_U(a_1) \geq 1/2$ (as always happens when $K = 2$), then the satisfaction of (3.2) for $i = 2, 3, \ldots, K$ implies that (3.2) is also satisfied for $i = 1$. In general, however, the K inequalities specified by (3.2) are independent.

In using typical sequence arguments, we shall usually require ε to be a very small positive number. Notice that, when $\varepsilon < 1$, $(1 - \varepsilon)P_U(a_i)$ is positive whenever $P_U(a_i)$ is positive. But it then follows from (3.2) that, if $\underline{u}$ is ε-typical, $n_{a_i}(\underline{u})$ must be positive for every i such that $P_U(a_i)$ is positive. But, because

$$\sum_{i=1}^{K} n_{a_i}(\underline{u}) = L,$$

we see that this is impossible unless L is at least as large as the number of different i for which $P_U(a_i)$ is positive. This is why our definition of a typical sequence cannot be used for a DMS with a countably infinite output alphabet for which infinitely many output letters have positive probability. When $\varepsilon < 1$, there would be no ε-typical output sequences (by our definition) of length L from such a source because L would always be smaller than the number of different i for which $P_U(a_i)$ is positive.

One further aspect of our definition of a typical sequence deserves mention. When $P_U(a_i) = 0$, we see that (3.2) is satisfied if and only if $n_{a_i}(\underline{u}) = 0$. Thus, if $\underline{u}$ is an ε-typical output sequence for a DMS, then *$\underline{u}$ cannot contain any letters a_i such that $P_U(a_i) = 0$*. This seems a very natural requirement on a "typical sequence", but some definitions of typical sequences do not demand this intuitively-pleasing prohibition of zero probability letters within "typical sequences".

3.4 Tchebycheff's Inequality and the Weak Law of Large Numbers

In Section 3.2, we invoked the "law of large numbers" in a qualitative way to motivate our definition of a typical sequence. We now provide a precise formulation of the "law of large numbers" that we shall use to make a quantitative study of typical sequences.

Our starting point is a very useful inequality due to Tchebycheff. Recall that if X is any real-valued random variable with finite mean $m_X = \mathrm{E}[X]$, then the variance of X, denoted $Var(X)$, is defined as

$$Var(X) = \mathrm{E}[(X - m_X)^2]. \tag{3.3}$$

Let A denote the event that $|X - m_X| \geq \varepsilon$ and A^c the complementary event that $|X - m_X| < \varepsilon$, where ε is any positive number. Then, the right side of (3.3) can be rewritten to give

$$Var(X) = \mathrm{E}\left[(X - m_X)^2 \middle| A\right] P(A) + \mathrm{E}\left[(X - m_X)^2 \middle| A^c\right] P(A^c).$$

But both expectations are obviously non-negative so that

$$Var(X) \geq \mathrm{E}\left[(X - m_X)^2 \middle| A\right] P(A).$$

Whenever A occurs, $(X - m_X)^2 \geq \varepsilon^2$ so that

$$\mathrm{E}\left[(X - m_X)^2 \middle| A\right] \geq \varepsilon^2$$

and hence

$$Var(X) \geq \varepsilon^2 P(A).$$

Dividing by ε^2 and writing $P(A)$ more transparently as $P(|X - m_X| \geq \varepsilon)$, we obtain *Tchebycheff's inequality*

$$P(|X - m_X| \geq \varepsilon) \leq \frac{Var(X)}{\varepsilon^2}, \tag{3.4}$$

which holds for every $\varepsilon > 0$.

We shall need three additional properties of variance. To derive these, let X and Y be real-valued random variables with finite means and variances. The first property is the trivial one that

$$Var(X) = \mathrm{E}[X^2] - m_X^2 \tag{3.5}$$

which follows from the definition (3.3) and the fact that

$$\begin{aligned}
\mathrm{E}[(X - m_X)^2] &= \mathrm{E}[X^2 - 2m_X X + m_X^2] \\
&= \mathrm{E}[X^2] - 2m_X \mathrm{E}[X] + m_X^2 \\
&= \mathrm{E}[X^2] - 2m_X^2 + m_X^2
\end{aligned}$$

The second property is the only slightly less trivial one that

$$Y = cX \qquad \Rightarrow Var(Y) = c^2 Var(X), \tag{3.6}$$

which follows from 3.5 and the fact that

$$\begin{aligned}
\mathrm{E}[Y^2] - m_Y^2 &= \mathrm{E}[Y^2] - (\mathrm{E}[Y])^2 \\
&= \mathrm{E}[c^2X^2] - (\mathrm{E}[cX])^2 \\
&= c^2\mathrm{E}[X^2] - c^2(\mathrm{E}[X])^2.
\end{aligned}$$

The last and least trivial of our desired three properties is the property that

$$X \text{ and } Y \text{ statistically independent} \Rightarrow Var(X + Y) = Var(X) + Var(Y). \tag{3.7}$$

This property follows from (3.5) and the facts that

$$\begin{aligned}
(\mathrm{E}[X + Y])^2 &= (\mathrm{E}[X] + \mathrm{E}[Y])^2 \\
&= (\mathrm{E}[X])^2 + 2\mathrm{E}[X]\mathrm{E}[Y] + (\mathrm{E}[Y])^2
\end{aligned}$$

and that

$$\begin{aligned}
\mathrm{E}\left[(X + Y)^2\right] &= \mathrm{E}[X^2 + 2XY + Y^2] \\
&= \mathrm{E}[X^2] + 2\mathrm{E}[XY] + \mathrm{E}[Y^2] \\
&= \mathrm{E}[X^2] + 2\mathrm{E}[X]\mathrm{E}[Y] + \mathrm{E}[Y^2],
\end{aligned}$$

where the last step follows from the statistical independence of X and Y because then

$$\begin{aligned}
\mathrm{E}[XY] &= \sum_x \sum_y xyP_{XY}(x, y) \\
&= \sum_x \sum_y xyP_X(x)P_Y(y) \\
&= \sum_x xP_X(x) \sum_y yP_Y(y) \\
&= \mathrm{E}[X]\mathrm{E}[Y].
\end{aligned} \tag{3.8}$$

By repeated application of (3.7), it follows that

$$X_1, X_2, \dots, X_N \text{ statistically independent } \Rightarrow Var\left(\sum_{i=1}^{N} X_i\right) = \sum_{i=1}^{N} Var(X_i). \tag{3.9}$$

Although we shall make no use of the fact, we would be remiss in our duties not to point out that the implication in (3.9) holds under the weaker hypothesis that $X_1, X_2, \dots, X_N$ are uncorrelated. By definition, $X_1, X_2, \dots, X_N$ are *uncorrelated* if $\mathrm{E}[X_i X_j] = \mathrm{E}[X_i]\mathrm{E}[X_j]$ whenever $1 \leq i < j \leq N$. From (3.8) we see that if $X_1, X_2, \dots, X_N$ are statistically independent, then they are also uncorrelated – but the converse is not true in general. We see also that we needed only the relation (3.8) in order to prove that the implication in (3.7) was valid. Thus, the implications in (3.7) and (3.9) remain valid when "statistically independent" is replaced by "uncorrelated".

We are at last ready to come to grips with the "law of large numbers". Suppose we make N independent trials of the same random experiment and that this random experiment defines some real-valued random variable X with mean m_X and finite variance $Var(X)$. Let X_i be the random variable whose value is the value of X on the i-th trial of our random experiment. We interpret the intuitive concept of "independent trials of the same random experiment" to mean that the random variables $X_1, X_2, \dots, X_N$ are i.i.d. and that each has the same probability distribution as X. Thus, in particular, $\mathrm{E}[X_i] = m_X$ and $Var(X_i) = Var(X)$ for $i = 1, 2, \dots, N$. We now define the *sample mean* S_N of our N independent observations of X to be the random variable

$$S_N = \frac{1}{N}(X_1 + X_2 + \cdots + X_N). \tag{3.10}$$

It follows immediately that

$$\mathrm{E}\left[S_N\right] = \frac{1}{N}\sum_{i=1}^{N} \mathrm{E}\left[X_i\right] = m_X. \tag{3.11}$$

Moreover, using (3.6) in (3.10) gives

$$Var\left[S_N\right] = \frac{1}{N^2} Var(X_1 + X_2 + \cdots + X_N).$$

Now using (3.9) yields

$$Var\left[S_N\right] = \frac{1}{N^2}\sum_{i=1}^{N} Var(X_i) = \frac{1}{N} Var(X). \tag{3.12}$$

We can now use Tchebycheff's inequality (3.4) to conclude that, for any given $\varepsilon > 0$,

$$P\left(|S_N - m_X| \geq \varepsilon\right) \leq \frac{Var(X)}{N\varepsilon^2}.$$

Equivalently, for any given $\varepsilon > 0$,

$$P\left(|S_N - m_X| < \varepsilon\right) \geq 1 - \frac{Var(X)}{N\varepsilon^2}. \tag{3.13}$$

By assumption, $Var(X)$ is finite; thus (3.13) implies that, for any given $\varepsilon > 0$,

$$\lim_{N \to \infty} P\left(|S_N - m_X| < \varepsilon\right) = 1. \tag{3.14}$$

Equation (3.14) is sometimes also written (see Chapter 0) as

$$\operatorname*{plim}_{N \to \infty} S_N = m_X.$$

Equation (3.14) is the qualitative formulation of the so-called *Weak Law of Large Numbers*, which asserts that *when N is very large, then the sample mean of N independent samples of a real-valued random variable X is virtually certain to be very close to the mean of X.* Inequality (3.13) is a quantitative formulation of this Weak Law of Large Numbers.

The reader will surely remember that in Section 3.2 we invoked the law of large numbers (in an intuitive way) to claim that *the fraction of times that some event occurs in many independent trials of a random experiment is virtually certain to be very close to the probability of that event.* To see how this conclusion follows from the Weak Law of Large Numbers, one uses the trick of "indicator random variables" for events. If A is an event, then the real-valued random variable X such that $X = 1$ when A occurs and $X = 0$ when A does not occur is called the *indicator random variable* for the event A. Thus $P_X(1) = P(A)$ and $P_X(0) = 1 - P(A)$, from which it follows that

$$\mathrm{E}[X] = P(A). \tag{3.15}$$

The fact that the mean of an indicator random variable equals the probability of the event that it "indicates" is what makes this random variable so useful. But $X^2 = X$ so that (3.15) implies

$$\mathrm{E}\left[X^2\right] = P(A),$$

which, together with (3.5), now gives

$$Var(X) = P(A)[1 - P(A)]. \tag{3.16}$$

Now suppose as before that we make N independent trials of this same random experiment and that X_i is the value of X on the i-th trial. Then $X_1 + X_2 + \cdots + X_N$ is just the number of times that the event A occured, which we denote now as N_A, and hence $S_N = N_A/N$ is just the fraction of times that event A occured. From (3.13), (3.15) and (3.16), we now conclude that, for any $\varepsilon > 0$,

$$P\left(\left|\frac{N_A}{N} - P(A)\right| < \varepsilon\right) \geq 1 - \frac{P(A)[1 - P(A)]}{N\varepsilon^2}. \tag{3.17}$$

Inequality (3.17) is a quantitative formulation of the *Weak Law of Large Numbers for Events*; the qualitative formulation is that, for any $\varepsilon > 0$,

$$\lim_{N\to\infty} P\left(\left|\frac{N_A}{N} - P(A)\right| < \varepsilon\right) = 1. \tag{3.18}$$

Inequality (3.17) is of course equivalent to the complementary inequality that states that, for any $\varepsilon > 0$,

$$P\left(\left|\frac{N_A}{N} - P(A)\right| \geq \varepsilon\right) \leq \frac{P(A)[1 - P(A)]}{N\varepsilon^2}. \tag{3.19}$$

It is this complementary inequality that we shall apply in our study of typical sequences.

Our discussion of the "weak law" of large numbers has surely made the thoughtful reader curious as to what the "strong law" might be, so we digress here to discuss this latter law. Suppose we make infinitely many independent trials of the random experiment that defines the real-valued random variable X and that, as before, we let X_i denote the random variable whose value is the value of X on the i-th trial. The sample means $S_1, S_2, S_3 \ldots$ are now highly dependent since, for example,

$$\begin{aligned} S_{N+1} &= \frac{1}{N+1}(X_1 + X_2 + \cdots + X_N + X_{N+1}) \\ &= \frac{N}{N+1}S_N + \frac{1}{N+1}X_{N+1}. \end{aligned}$$

Thus we should expect that when S_N is close to $\mathrm{E}[X]$ then S_{N+1} will simultaneously be close to $\mathrm{E}[X]$. The *Strong Law of Large Numbers* asserts that when N is very large, then, in an infinite sequence of independent samples of a real-valued random variable X, it is virtually certain that the sample means of the first N samples, the first $N+1$ samples, the first $N+2$ samples, etc., will all simultaneously be very close to the mean of X. The equivalent qualitative statement is that, for any $\varepsilon > 0$,

$$\lim_{N\to\infty} P\left(\sup_{i\geq N}|S_i - m_X| < \varepsilon\right) = 1 \tag{3.20}$$

where "sup" denotes the "supremum" of the indicated quantities, i.e., the maximum of these quantities when there is a maximum and the least upper bound on these quantities otherwise.

We applied Tchebycheff's inequality to S_N above to conclude that, for any $\varepsilon > 0$,

$$P(|S_N - m_X| \geq \varepsilon) \leq \frac{Var(X)}{N\varepsilon^2}. \tag{3.21}$$

The bound (3.21) was tight enough to imply (3.14), the Weak Law of Large Numbers, but it is not tight enough to imply (3.20), the Strong Law. The reason is as follows. Tchebycheff's inequality (3.4) is the tightest bound on $P(|X - m_X| \geq \varepsilon)$ that can be proved when only m_X and $Var(X)$ are known. However, we know much more about the random variable S_N than just its mean and variance. We know also that S_N is the sum of N i.i.d. random variables. This further information allows one to prove a much stronger bound than Tchebycheff's inequality on $P(|S_N - m_X| \geq \varepsilon)$. This much stronger bound, *Chernoff's bound*, shows that, for any $\varepsilon > 0$,

$$P(|S_N - m_X| \geq \varepsilon) \leq 2(\beta_\varepsilon)^N \tag{3.22}$$

where β_ε depends on ε, but always satisfies

$$0 < \beta_\varepsilon < 1.$$

Note that the upper bound β_ε^N of (3.22) decreases *exponentially fast* as N increases, whereas the bound of (3.21) decreases only as $1/N$. Chernoff's bound (3.22) gives a much truer picture of the way that $P(|S_N - m_X| \geq \varepsilon)$ decreases with N than does our earlier bound (3.21). However, we shall continue to use the simple bound, in its form (3.19) for events, in our study of typical sequences because it is sufficiently tight to allow us to establish the *qualitative* results of most interest. When we seek *quantitative* results, as in Chapter 5, we shall not use typical sequence arguments at all. The typical sequence approach is excellent for gaining insight into the results of information theory, but it is not well-suited for quantitative analysis.

To see how the Chernoff bound (3.22) can be used to derive the Strong Law of Large Numbers, (3.20), we note that if the supremum of $|S_i - m_X|$ for $i \geq N$ is at least ε, then $|S_i - m_X| \geq \varepsilon$ must hold for at least one value of i with $i \geq N$. By the *union bound* (i.e., by the fact that the probability of a union of events is upper bounded by the sum of the probabilities of the events), it follows then that, for any $\varepsilon > 0$,

$$P\left(\sup_{i\geq N}|S_i - m_X| \geq \varepsilon\right) \leq \sum_{i=N}^{\infty} P(|S_i - m_X| \geq \varepsilon). \tag{3.23}$$

If the simple bound (3.21) is used on the right in (3.23), the summation diverges and no useful conclusion is possible. Substituting the Chernoff bound (3.22) in (3.23), however, gives

$$P\left(\sup_{i\geq N}|S_i - m_X| \geq \varepsilon\right) \leq \frac{2}{1-\beta_\varepsilon}(\beta_\varepsilon)^N. \tag{3.24}$$

The right side of (3.24) approaches 0 (exponentially fast!) as N increases, and this establishes the Strong Law of Large Numbers, (3.20).

This concludes our rather long discussion of Tchebycheff's inequality and the Weak Law of Large Numbers, the only result of which that we shall use in the sequel is the simple bound (3.19). It would be misleading, however, to use (3.19) without pointing out the extreme looseness of this simple bound; and we would find it most unsatisfactory to discuss a "weak law" without telling the reader about the "strong law" as well. But the reader is surely as impatient as we are to return to the investigation of typical sequences.

3.5 Properties of Typical Sequences

As in Section 3.3, $\underline{U} = [U_1, U_2, \ldots, U_L]$ will denote a length L output sequence from a K-ary DMS with output alphabet $\{a_1, a_2, \ldots, a_K\}$, $\underline{u} = [u_1, u_2, \ldots, u_L]$ will denote a possible value of $\underline{U}$, and $n_{a_i}(\underline{u})$ will denote the number of occurrences of a_i in the sequence $\underline{u}$. From the definition of a DMS, it follows that

$$\begin{aligned} P_{\underline{U}}(\underline{u}) &= \prod_{j=1}^{L} P_U(u_j) \\ &= \prod_{i=1}^{K} [P_U(a_i)]^{n_{a_i}(\underline{u})} . \end{aligned} \tag{3.25}$$

The definition (3.2) of an ε-typical sequence can equivalently be written as

$$(1-\varepsilon)LP_U(a_i) \leq n_{a_i}(\underline{u}) \leq (1+\varepsilon)LP_U(a_i). \tag{3.26}$$

Using the right inequality of (3.26) in (3.25) gives

$$P_{\underline{U}}(\underline{u}) \geq \prod_{i=1}^{K} [P_U(a_i)]^{(1+\varepsilon)LP_U(a_i)}$$

or, equivalently,

$$P_{\underline{U}}(\underline{u}) \geq \prod_{i=1}^{K} 2^{(1+\varepsilon)LP_U(a_i)\log_2 P_U(a_i)}$$

or, again equivalently,

$$P_{\underline{U}}(\underline{u}) \geq 2^{(1+\varepsilon)L\sum_{i=1}^{K} P_U(a_i)\log_2 P_U(a_i)}.$$

But this last inequality is just

$$P_{\underline{U}}(\underline{u}) \geq 2^{-(1+\varepsilon)LH(U)}$$

where the entropy $H(U)$ is in bits. A similar argument using the left inequality of (3.26) in (3.25) gives

$$P_{\underline{U}}(\underline{u}) \leq 2^{-(1-\varepsilon)LH(U)}$$

and completes the proof of the following property.

Property 1 of Typical Sequences: If $\underline{u}$ is an ε-typical output sequence of length L from a K-ary DMS with entropy $H(U)$ in bits, then

$$2^{-(1+\varepsilon)LH(U)} \leq P_{\underline{U}}(\underline{u}) \leq 2^{-(1-\varepsilon)LH(U)}. \tag{3.27}$$

We now wish to show that, when L is very large, the output sequence $\underline{U}$ of the DMS is virtually certain to be ε-typical. To this end, we let B_i denote the event that $\underline{U}$ takes on a value $\underline{u}$ such that (3.2) is *not* satisfied. Then, if $P_U(a_i) > 0$, we have

$$P(B_i) = P\left(\left|\frac{n_{a_i}(\underline{u})}{L} - P_U(a_i)\right| > \varepsilon P_U(a_i)\right)$$
$$\leq \frac{P_U(a_i)\,[1 - P_U(a_i)]}{L\,[\varepsilon P_U(a_i)]^2}$$

where we have made use of (3.19). Simplifying, we have

$$P(B_i) \leq \frac{1 - P_U(a_i)}{L\varepsilon^2 P_U(a_i)} \tag{3.28}$$

whenever $P_U(a_i) > 0$. Now defining $P_{\min}$ as the smallest non-zero value of $P_U(u)$, we weaken (3.28) to the simpler

$$P(B_i) < \frac{1}{L\varepsilon^2 P_{\min}}. \tag{3.29}$$

Because $P(B_i) = 0$ when $P_U(a_i) = 0$, inequality (3.29) holds for all $i, 1 \leq i \leq K$. Now let F be the "failure" event that $\underline{U}$ is not ε-typical. Because when F occurs at least one of the events $B_i, 1 \leq i \leq K$, must occur, it follows by the union bound that

$$P(F) \leq \sum_{i=1}^{K} P(B_i).$$

Using (3.29), we now obtain our desired bound

$$P(F) < \frac{K}{L\varepsilon^2 P_{\min}}, \tag{3.30}$$

which assures us that $P(F)$ indeed approaches 0 as L increases.

Property 2 of Typical Sequences: The probability, $1 - P(F)$, that the length L output sequence $\underline{U}$ from a K-ary DMS is ε-typical satisfies

$$1 - P(F) > 1 - \frac{K}{L\varepsilon^2 P_{\min}} \tag{3.31}$$

where $P_{\min}$ is the smallest positive value of $P_U(u)$.

It remains only to obtain bounds on the number M of ε-typical sequences $\underline{u}$. The left inequality in (3.27) gives

$$1 = \sum_{\text{all } \underline{u}} P_{\underline{U}}(\underline{u}) \geq M \cdot 2^{-(1+\varepsilon)LH(U)},$$

which gives our desired upper bound

$$M \leq 2^{(1+\varepsilon)LH(U)}. \tag{3.32}$$

On the other hand, the total probability of the ε-typical sequences $\underline{u}$ is $1 - P(F)$, so we can use the right inequality in (3.27) to obtain

$$1 - P(F) \leq M \cdot 2^{-(1-\varepsilon)LH(U)}.$$

Using (3.31), we now have our desired lower bound

$$M > \left(1 - \frac{K}{L\varepsilon^2 P_{\min}}\right) \cdot 2^{(1-\varepsilon)LH(U)}. \tag{3.33}$$

Property 3 of Typical Sequences: The number M of ε-typical sequences $\underline{u}$ from a K-ary DMS with entropy $H(U)$ in bits satisfies

$$\left(1 - \frac{K}{L\varepsilon^2 P_{\min}}\right) \cdot 2^{(1-\varepsilon)LH(U)} < M \le 2^{(1+\varepsilon)LH(U)}, \tag{3.34}$$

where $P_{\min}$ is the smallest positive value of $P_U(u)$.

Property 3 says that, when L is large and ε is small, there are roughly $2^{LH(U)}$ ε-typical sequences $\underline{u}$. Property 1 says then that each of these ε-typical sequences has probability roughly equal to $2^{-LH(U)}$. Finally, Property 2 says that the total probability of these ε-typical sequences is very nearly 1. The statement of these three properties is sometimes called the *asymptotic equipartition property* (AEP) of the output sequence of a DMS and is what should be remembered about typical sequences; it is rough truths rather than fine details that are most useful in forming one's intuition in any subject.

3.6 Block-to-Block Coding of a DMS

We now return to the block-to-block coding problem for a DMS that originally prompted our investigation of typical sequences. We again consider the situation shown in Figure 3.1, but now we allow our encoder to be "lossy". Suppose we assign a unique D-ary codeword of length N to each of the M ε-typical source output sequences $\underline{u}$, but use a single additional codeword to code all the non-typical source output sequences. The smallest N that suffices satisfies

$$D^{N-1} < M + 1 \le D^N,$$

and hence we are sure that we can have

$$M \ge D^{N-1}$$

or equivalently, that

$$(N-1)\log D \le \log M.$$

Now using (3.32) we obtain

$$(N-1)\log_2 D \le (1+\varepsilon)LH(U)$$

or, equivalently,

$$\frac{N}{L} \le \frac{H(U)}{\log_2 D} + \frac{\varepsilon H(U)}{\log_2 D} + \frac{1}{L}. \tag{3.35}$$

The probability of loss, i.e., the probability that the codeword will not uniquely specify the source output sequence, is just the probability $P(F)$ that $\underline{U}$ is not ε-typical. Thus, $P(F)$ is upper bounded according to inequality (3.30). Because we can take ε to be an arbitrarily small positive number and we can choose L to be arbitrarily large, inequalities (3.30) and (3.35) establish the following theorem.

The Block-to-Block Coding Theorem for a DMS: Given a K-ary DMS with output entropy $H(U)$ and given any positive numbers ε_1 and ε_2, there exists, for all sufficiently large L, a D-ary block code of blocklength N for block message sets of blocklength L such that

$$\frac{N}{L} < \frac{H(U)}{\log D} + \varepsilon_1$$

and such that the probability, $P(F)$, that the codeword will not uniquely specify the message satisfies

$$P(F) < \varepsilon_2$$

It should be apparent that one cannot make $P(F)$ arbitrarily small and at the same time have N/L substantially less than $H(U)/\log D$. In fact, we could prove this "converse" to the above theorem also by typical sequence arguments. However, such a converse would not answer the question of whether it is possible to recover virtually all of the digits in $\underline{U}$ correctly when N/L was substantially less than $H(U)/\log D$. This is the more interesting question, and we shall treat it in an exercise in the next chapter.

This chapter should have given the reader some idea of the power of typical sequence arguments.

Chapter 4

CODING FOR A NOISY DIGITAL CHANNEL

4.1 Introduction

The basic goal of the communications engineer is to transmit information efficiently and reliably from its source through a "channel" to its destination. In Chapter 2, we learned some fundamental things about information sources. Now we wish to learn some equally fundamental things about channels. The first question to be answered is: what is a "channel"?

Kelly has given a provocative description of a channel as that part of the communication system that one is either "unwilling or unable to change." If we must use radio signals in a certain portion of the frequency spectrum, then that necessity becomes part of the channel. If we have a certain radio transmitter on hand and are unwilling to design or purchase another, then that constraint also becomes part of the channel. But when the channel has finally been specified, the remaining freedom is precisely what can be used to transmit information. If we could send only one signal (say a sinusoid with a given amplitude, frequency and phase) then there is no way that we could transmit information to the destination. We need the freedom to choose freely one of at least two signals if we are going to transmit information. Once we have chosen our signal, the channel governs what happens to that signal on its way to the destination, but the choice of the signal was ours. In mathematical terms, this means that the channel specifies the *conditional* probabilities of the various signals that can be received, but the *a priori* probabilities of the input signals are chosen by the sender who uses the channel.

We shall consider only time-discrete channels so that the channel input and output can both be described as sequences of random variables. The input sequence $X_1, X_2, X_3, \ldots$ is chosen by the sender; but the channel then determines the resulting conditional probabilities of the output sequence $Y_1, Y_2, Y_3, \ldots$. The *discrete memoryless channel* (DMC) is mathematically the simplest such channel and the one to which we shall give the most attention. A DMC consists of the specification of three quantities:

(1) the *input alphabet*, A, which is a finite or at most countably infinite set $\{a_1, a_2, a_3, \ldots\}$, each member of which represents one of the signals that the sender can choose at each time instant when he uses the channel;

(2) the *output alphabet*, B, which is also a finite or at most countably infinite set $\{b_1, b_2, b_3, \ldots\}$, each member of which represents one of the output signals that might result each time instant that the channel is used; and

(3) the *conditional probability distributions* $P_{Y|X}(.|x)$ over B for each $x \in A$ which govern the channel

behavior in the manner that

$$P(y_n|x_1,\ldots,x_{n-1},x_n,y_1,\ldots,y_{n-1}) = P_{Y|X}(y_n|x_n), \tag{4.1}$$

for $n = 1, 2, 3, \ldots$

Equation (4.1) is the mathematical statement that corresponds to the "memoryless" nature of the DMC. What happens to the signal sent on the n-th use of the channel is independent of what happens on the previous $n-1$ uses.

It should also be noted that (4.1) implies that the DMC is *time-invariant* in the sense that the probability distribution $P_{Y_n|X_n}$ does not depend on n. This is the reason that we could not write simply $P(y_n|x_n)$ on the right side of (4.1), as this is our shorthand notation for $P_{Y_n|X_n}(y_n|x_n)$ and would not imply that this probability distribution does not depend on n.

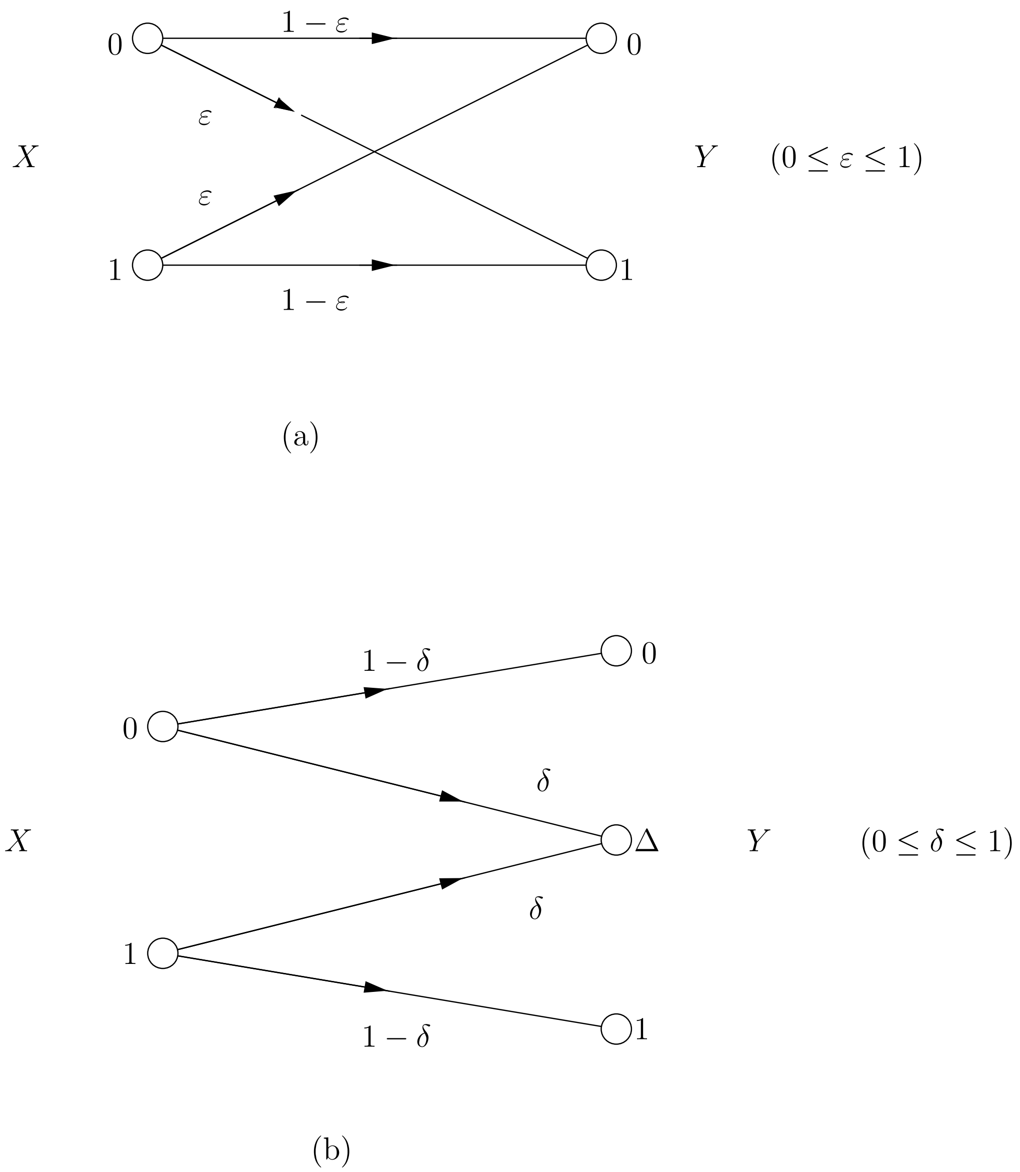


Figure 4.1: (a) The binary symmetric channel (BSC), and (b) the binary erasure channel (BEC).

DMC's are commonly specified by diagrams such as those in Figure 4.1 where:

(1) the nodes at the left indicate the input alphabet A;

(2) the nodes at the right indicate the output alphabet B; and

(3) the directed branch from a_i to b_j is labelled with the conditional probability $P_{Y|X}(b_j|a_i)$ (unless this probability is 0 in which case the branch is simply omitted.)

The channel of Figure 4.1(a) is called the *binary symetric channel* (BSC), while that of Figure 4.1(b) is called the *binary erasure channel* (BEC). We shall give special attention hereafter to these two deceptively-simple appearing DMC's.

The following description of how a BSC might come about in practice may help to give a "feel" for this channel. Suppose that every millisecond one can transmit a radio pulse of duration one millisecond at one of two frequencies, say f_0 or f_1, which we can think of as representing 0 or 1, respectively. Suppose that these signals are transmitted through some broadband medium, e.g., a coaxial cable, and processed by a receiver which examines each one-millisecond interval and then makes a "hard decision" as to which waveform was more likely present. (Such hard-decision receivers are unfortunately commonly used; a "soft-decision" receiver that also gives some information about the reliability of the decision is generally preferable, as we shall see later.) Because of the thermal noise in the transmission medium and in the receiver "front-end", there will be some probability ε that the decision in each millisecond interval is erroneous. If the system is sufficiently wideband, errors in each interval will be independent so that the *overall system* [consisting of the "binary-frequency-shift-keyed (BFSK) transmitter", the waveform transmission medium, and the hard-decision BFSK demodulator] becomes just a BSC that is used once each millisecond. One could give endlessly many other practical situations for which the BSC is the appropriate information-theoretic model. The beauty of information theory is that it allows us to ignore the many technical details of such practical systems and to focus instead on the only feature that really counts as far as the ability to transmit information is concerned, namely the conditional probability distributions that are created.

We now point out one rather trivial but possible confusing mathematical detail about DMC's. Suppose we are using the BSC of Figure 4.1(a) and decide that we shall always send only 0's. In other words, we choose $P_{X_n}(0) = 1$ and $P_{X_n}(1) = 0$ for all n. Note that the channel still specifies $P_{Y_n|X_n}(1|1) = 1 - \varepsilon$, even though the conditioning event has zero probability by the formula $P_{X_nY_n}(1,1)/P_{X_n}(1)$. Here, we are merely exploiting the fact that, mathematically, a conditional probability distribution can be arbitrarily chosen when the conditioning event has probability zero (see the discussion following Equation (16)), but we always are allowing the channel to make this arbitrary choice for us to avoid having to treat as special cases those uninteresting cases when the sender decides never to use one or more inputs.

Finally, we wish to discuss the situation when we use a DMC *without feedback*, i.e., when we select the inputs so that

$$P(x_n|x_1,\ldots,x_{n-1},y_1,\ldots,y_{n-1}) = P(x_n|x_1,\ldots,x_{n-1}) \tag{4.2}$$

for $n = 1, 2, 3, \ldots$. Notice that (4.2) does not imply that we choose each input digit independently of previous input digits, only that we are not using the past output digits in any way (as we could if a feedback channel was available from the output to the input of the DMC) when we choose successive input digits. We now prove a fundamental result, namely:

Theorem 4.1: When a DMC is used without feedback,

$$P(y_1,\ldots y_n|x_1,\ldots,x_n) = \prod_{i=1}^{n} P_{Y|X}(y_i|x_i) \tag{4.3}$$

for $n = 1, 2, 3, \ldots$.

Proof: From the multiplication rule for conditional probabilities, we have

$$P(x_1,\ldots,x_n,y_1,\ldots y_n)=\prod_{i=1}^{n}P(x_i|x_1,\ldots,x_{i-1},y_1,\ldots,y_{i-1})P(y_i|x_1,\ldots,x_i,y_1,\ldots,y_{i-1}) \tag{4.4}$$

Making use of (4.1) and (4.2) on the right of (4.4), we obtain

$$\begin{aligned}
P(x_1,\ldots,x_n,y_1,\ldots y_n) &= \prod_{i=1}^{n}P(x_i|x_1,\ldots,x_{i-1})P_{Y|X}(y_i|x_i)\\
&= \left[\prod_{j=1}^{n}P(x_j|x_1,\ldots,x_{j-1})\right]\left[\prod_{i=1}^{n}P_{Y|X}(y_i|x_i)\right]\\
&= P(x_1,\ldots,x_n)\prod_{i=1}^{n}P_{Y|X}(y_i|x_i).
\end{aligned}$$

Dividing now by $P(x_1,\ldots,x_n)$, we obtain (4.3). □

The relationship (4.3) is so fundamental that it is often erroneously given as the definition of the DMC. The reader is warned to remember this when delving into the information theory literature. When (4.3) is used, one is tacitly assuming that the DMC is being used without feedback.

Example 4.1: Notice that, for the BSC,

$$P_{Y|X}(y|x)=\begin{cases}1-\varepsilon & \text{if } y=x\\ \varepsilon & \text{if } y\neq x.\end{cases}$$

Thus, when a BSC is used without feedback to transmit a block of N binary digits, then

$$\begin{aligned}
P(\underline{y}|\underline{x}) &= \varepsilon^{d(\underline{x},\underline{y})}(1-\varepsilon)^{N-d(\underline{x},\underline{y})}\\
&= (1-\varepsilon)^N\left(\frac{\varepsilon}{1-\varepsilon}\right)^{d(\underline{x},\underline{y})}
\end{aligned} \tag{4.5}$$

where $\underline{x}=[x_1,x_2,\ldots,x_N]$ is the transmitted block, $\underline{y}=[y_1,y_2,\ldots,y_N]$ is the received block, and $d(\underline{x},\underline{y})$ is the Hamming distance between $\underline{x}$ and $\underline{y}$, i.e., the number of positions in which the vectors $\underline{x}$ and $\underline{y}$ differ.

4.2 Channel Capacity

Recall that a DMC specifies the conditional probability distribution $P(y|x)$, but that the sender is free to choose the input probability distribution $P(x)$. Thus, it is natural to define the *capacity*, C, of a DMC as the maximum average mutual information $I(X;Y)$ that can be obtained by choice of $P(x)$, i.e.,

$$\boxed{C=\max_{P_X} I(X;Y).} \tag{4.6}$$

Equivalently,

$$C = \max_{P_X} [H(Y) - H(Y|X)]. \tag{4.7}$$

But now we have the more challenging task to show that this definition is useful, i.e., to show that C gives the answer to some important practical question about information transmission. This we shall shortly do, but first we shall amuse ourselves by calculating the capacity of some especially simple channels which, thanks to the benevolence of nature, include most channels of practical interest.

For convenience, suppose that the DMC has a finite number, K, of input letters, and a finite number, J, of output letters. We will say that such a DMC is *uniformly dispersive* if the probabilities of the J transitions leaving an input letter have, when put in decreasing order, the same values (say, $p_1 \geq p_2 \geq \cdots \geq p_J$) for each of the K input letters. Both channels in Figure 4.1 are uniformly dispersive as is also that of Figure 4.2(a) (for which $p_1 = .5$ and $p_2 = .5$), but the channel of Figure 4.2(b) is not uniformly dispersive. When a DMC is uniformly dispersive, no matter which input letter we choose, we get the same "spread of probability" over the output letters. Thus, it is not surprising to discover:

Lemma 4.1: Independently of the choice of a distribution $P(x)$, for a uniformly dispersive DMC,

$$H(Y|X) = -\sum_{j=1}^{J} p_j \log p_j \tag{4.8}$$

where $p_1, p_2, \ldots, p_J$ are the transition probabilities leaving each input letter.

Proof: Immediately from the definition of a uniformly dispersive channel, it follows that

$$H(Y|X = a_k) = -\sum_{j=1}^{J} p_j \log p_j \tag{4.9}$$

for $k = 1, 2, \ldots, K$. Equation (4.8) now follows from (4.9) and (1.18). □

Example 4.2: For the BSC and BEC of Figure 4.1, (4.8) gives

$$\begin{aligned} H(Y|X) &= h(\varepsilon) && \text{(BSC)} \\ H(Y|X) &= h(\delta) && \text{(BEC).} \end{aligned}$$

For the channel of Figure 4.2(a), (4.8) gives

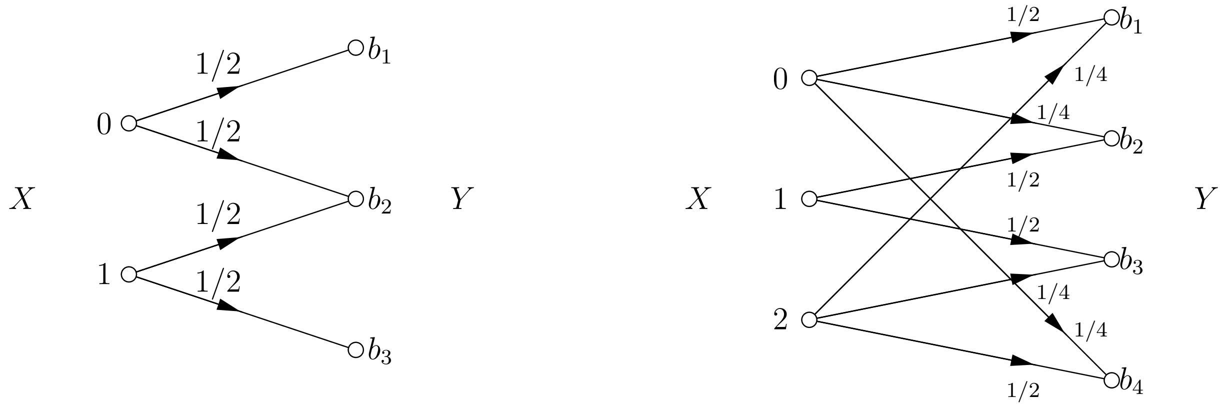
$$H(Y|X) = h(.5) = 1 \text{ bit}$$

From (4.7), we can see that, for a uniformly dispersive DMC, finding the input probability distribution that achieves capacity reduces to the simpler problem of finding the input distribution that maximizes the uncertainty of the output. In other words, *for a uniformly dispersive DMC,*

$$C = \max_{P_X} [H(Y)] + \sum_{j=1}^{J} p_j \log p_j \tag{4.10}$$

where $p_1, p_2, \ldots, p_J$ are the transition probabilities leaving an input letter.

A uniformly dispersive channel is one in which the probabilities depart from each input letter in the same manner. We now consider channels with the property that the probabilities converge upon each output letter in the same manner. We shall say that a DMC is *uniformly focusing* if the probabilities of the K transitions to an output letter have, when put in decreasing order, the same values for each of the



(a) A uniformly dispersive DMC that is not uniformly focusing.

(b) A uniformly focusing DMC that is not uniformly dispersive.

Figure 4.2: Two DMC's which are not strongly symmetric.

J output letters. The BSC [See Figure 4.1(a)] is uniformly focusing, but the BEC [see Figure 4.1(b)] is not. The channel of Figure 4.2(b) is uniformly focusing, but that of Figure 4.2(a) is not. The following property of uniformly focusing channels should hardly be surprising:

Lemma 4.2: If a K-input, J-output DMC is uniformly focusing, then the uniform input probability distribution [i.e., $P(x) = \frac{1}{K}$, for all x] results in the uniform output probability distribution [i.e., $P(y) = \frac{1}{J}$ for all y.]

Proof:

$$P(y) = \sum_x P(y|x)P(x) = \frac{1}{K}\sum_x P(y|x).$$

But, since the DMC is uniformly focusing, the sum on the right in this last equation is the same for all J values of y. Thus $P(y)$ is the uniform probability distribution. □

It follows immediately from Lemma 4.2 and Theorem 1.1 that, *for a uniformly focusing DMC* with K inputs and J outputs,

$$\max_{P_X}\,[H(Y)] = \log J \tag{4.11}$$

and is achieved (not necessarily uniquely) by the uniform input distribution

$$P(x) = \frac{1}{K}, \qquad \text{for all } x. \tag{4.12}$$

The BSC is both uniformly dispersive and uniformly focusing, but the BEC is not. Yet the BEC certainly appears to be "symmetric", so we should give a stronger name for the kind of "symmetry" that the BSC possesses. Thus, we shall say that a DMC is *strongly symmetric* when it is both uniformly dispersive and uniformly focusing. The following theorem is now an immediate consequence of (4.10) and (4.11) and (4.12) above.

Theorem 4.2: For a strongly symmetric channel with K input letters and J output letters,

$$C = \log J + \sum_{j=1}^{J} p_j \log p_j \tag{4.13}$$

where $p_1, p_2, \ldots, p_J$ are the transition probabilities leaving an input letter. Moreover, the uniform input probability distribution

$$P(x) = \frac{1}{K}, \qquad \text{all } x \tag{4.14}$$

achieves capacity (but there may also be other capacity-achieving distributions).

Example 4.3: The BSC [see Figure 4.1(a)] is strongly symmetric. Thus,

$$C = 1 + \varepsilon \log_2 \varepsilon + (1-\varepsilon)\log_2(1-\varepsilon)$$

or, equivalently,

$$C = 1 - h(\varepsilon)\text{bits/use.} \qquad \text{(BSC)} \tag{4.15}$$

Capacity is achieved by the uniform input distribution $P_X(0) = P_X(1) = \frac{1}{2}$, which is the unique capacity-achieving distribution unless $\varepsilon = \frac{1}{2}$ in which case all input distributions trivially obtain the useless capacity $C = 0$.

From (4.11), we see that the necessary and sufficient condition for an input probability distribution to achieve capacity on a strongly symmetric DMC is that it gives rise to the uniform output probability distribution.

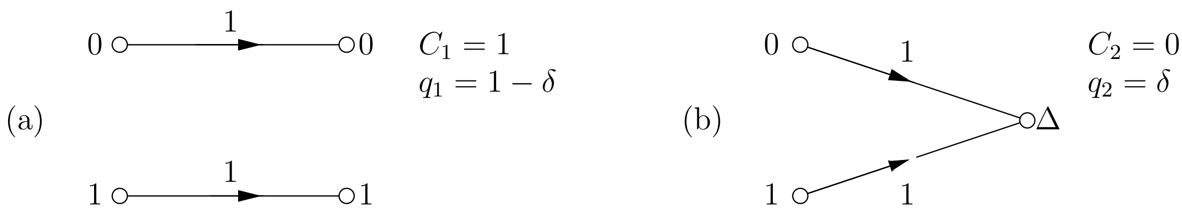


Figure 4.3: The two strongly symmetric channels that compose the BEC.

We now wish to give a general definition of "symmetry" for channels that will include the BEC. The BEC [see Figure 4.1(b)] is not strongly symmetric, but a closer examination reveals that it is composed of the two strongly symmetric channels shown in Figure 4.3, in the sense that we can consider that, after the sender chooses an input letter, the BEC chooses to send that letter over the channel of Figure 4.3(a) or the channel of Figure 4.3(b) with probabilities $q_1 = 1 - \delta$ and $q_2 = \delta$, respectively. In Figure 4.4, we have shown this decomposition of the BEC more precisely.

We now define a DMC to be *symmetric* if, for some L, it can be decomposed into L strongly symmetric channels with selection probabilities $q_1, q_2, \ldots, q_L$. It is fairly easy to test a DMC for symmetry. One merely needs to examine each subset of output letters that have the same focusing (i.e., have the same K probabilities in some order on the K transitions into that output letter), then check to see that each input letter has the same dispersion into this subset of output letters (i.e., has the same J_i probabilities in some order on the transitions from that input letter to the J_i output letters in the subset being considered). If so, one has found a component strongly symmetric channel whose selection probability q_i is the sum of the J_i probabilities on the transitions from an input letter to these J_i output letters. If and only if this can be done for each of these subsets of output letters, then the channel is symmetric. (Notice that, if the channel is symmetric, the entire channel must be uniformly dispersive. Thus, if a

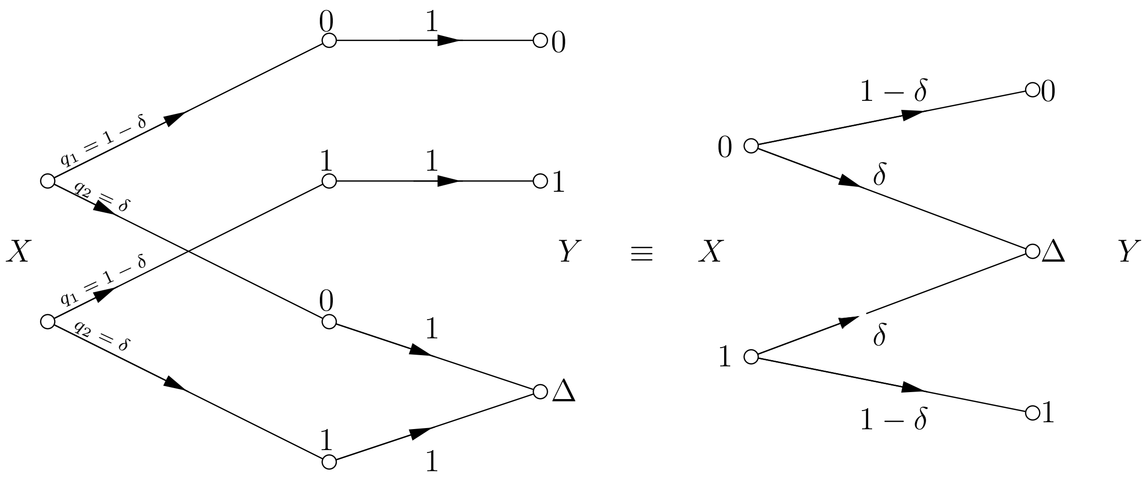


Figure 4.4: The decomposition of the BEC into two strongly symmetric channels with selection probabilities q_1 and q_2.

DMC is not uniformly dispersive, one need not resort to this procedure to conclude that it cannot be symmetric.) The reader can test his mastery of this procedure on the two DMC's of Figure 4.2, neither of which is strongly symmetric, but one of which is symmetric.

The importance of symmetric channels arises from the fact that, because of natural "symmetries", the DMC's encountered in practice are usually symmetric. Thus, it is fortunate that it is very easy to find the capacity of symmetric channels.

Theorem 4.3: The capacity C of a symmetric DMC is given by

$$C = \sum_{i=1}^{L} q_i C_i \tag{4.16}$$

where $C_1, C_2, \ldots, C_L$ are the capacities of the L strongly symmetric channels into which the DMC can be decomposed with selection probabilities $q_1, q_2, \ldots, q_L$, respectively.

Example 4.4: Referring to Figures 4.3 and 4.4, we see that the BEC can be decomposed into $L = 2$ strongly symmetric channels with capacities $C_1 = 1, C_2 = 0$ and selection probabilities $q_1 = 1 - \delta, q_2 = \delta$, respectively. Thus, Theorem 4.3 gives

$$C = 1 - \delta \text{ bits/use.} \qquad \text{(BEC)} \tag{4.17}$$

Proof of Theorem 4.3: Consider the DMC as decomposed into L strongly symmetric channels with selection probabilities $q_1, q_2, \ldots, q_L$ (as shown in Figure 4.4 for the BEC), and define the random variable Z to have value i (where $1 \leq i \leq L$) in case the output letter that occurs is an output letter of the i-th component channel. Thus,

$$H(Z|Y) = 0. \tag{4.18}$$

Notice that we can also consider Z as indicating which of the L component channels was selected for transmission so that

$$P_Z(i) = q_i \tag{4.19}$$

and, moreover, Z is statistically independent of the input X. It follows from (4.18) that

$$H(YZ) = H(Y) + H(Z|Y) = H(Y)$$

and thus that

$$
\begin{aligned}
H(Y) &= H(YZ) \\
&= H(Z) + H(Y|Z) \\
&= H(Z) + \sum_{i=1}^{L} H(Y|Z=i)P_Z(i) \\
&= H(Z) + \sum_{i=1}^{L} H(Y|Z=i)q_i. \qquad (4.20)
\end{aligned}
$$

But (4.18) also implies

$$H(Z|XY) = 0$$

and thus

$$H(YZ|X) = H(Y|X),$$

which further implies

$$
\begin{aligned}
H(Y|X) &= H(YZ|X) \\
&= H(Z|X) + H(Y|XZ) \\
&= H(Z|X) + \sum_{i=1}^{L} H(Y|X, Z=i)q_i \\
&= H(Z) + \sum_{I=1}^{L} H(Y|X, Z=i)q_i \qquad (4.21)
\end{aligned}
$$

where at the last step we have made use of the fact that X and Z are independent. Subtracting (4.21) from (4.20), we obtain

$$I(X;Y) = \sum_{i=1}^{L} [H(Y|Z=i) - H(Y|X, Z=i)]q_i. \qquad (4.22)$$

Let $I_i(X;Y)$ denote the mutual information between the input and output of the i-th component strongly symmetric channel when this channel *alone* is used with the *same* input probability distribution $P(x)$ as we are using for the symmetric DMC itself. We first note that, by Theorem 4.2,

$$I_i(X;Y) \leq C_i \qquad (4.23)$$

with equality when $P(x)$ is the uniform distribution. Moreover, we note further that

$$I_i(X;Y) = H(Y|Z=i) - H(Y|X, Z=i)$$

because $P_{X|Z}(x|i) = P_X(x)$ by the assumed independence of X and Z.

Hence, (4.22) can be rewritten as

$$I(X;Y) = \sum_{i=1}^{L} I_i(X;Y)q_i \qquad (4.24)$$

But the uniform distribution $P(x)$ *simultaneously* maximizes each $I_i(X;Y)$ and hence achieves

$$C = \max_{P_X} I(X;Y) = \sum_{i=1}^{L} C_i q_i \qquad (4.25)$$

as was to be shown. □

It is important to note that to go from (4.24) to (4.25), we made strong use of the fact that one input probability distribution simultaneously maximized the mutual information for all the component channels. Thus, the relation (4.16) will *not* hold in general when we decompose a channel into *non-symmetric* channels selected according to certain probabilities. (In general, $C = \sum_{i=1}^{L} q_i C_i$ will then be only an upper bound on capacity.)

Note: The reader is cautioned that the terminology we have used in discussing channel symmetry is not standard. Fano uses the terms "uniform from the input" and "uniform from the output" for what we have called "uniformly dispersive" and "uniformly focusing", respectively, and he uses "doubly uniform" for what we have called "strongly symmetric". Our definition of "symmetric" coincides with that given by Gallager who, however, did not give the interpretation of a symmetric DMC as a collection of strongly symmetric channels selected according to certain probabilities.

4.3 The Data Processing Lemma and Fano's Lemma

In this section, we introduce two general information-theoretic results that we shall subsequently apply in our study of channels, but which have many other applications as well.

The first of these results speaks about the situation shown in Figure 4.5. The two "processors" shown in this figure are completely arbitrary devices. They may be deterministic or only probabilistic – for instance each could contain one or more monkeys flipping coins, the results of which affect the output. The "black boxes" may have anything at all inside. The only thing that Figure 4.5 asserts is that there is no "hidden path" by which X can affect Z, i.e., X can affect Z only indirectly through its effect on Y. In mathematical terms, this constraint can be expressed as

$$P(z|xy) = P(z|y) \tag{4.26}$$

for all y such that $P(y) \neq 0$, which means simply that, when y is given, then z is not further influenced by also giving x.

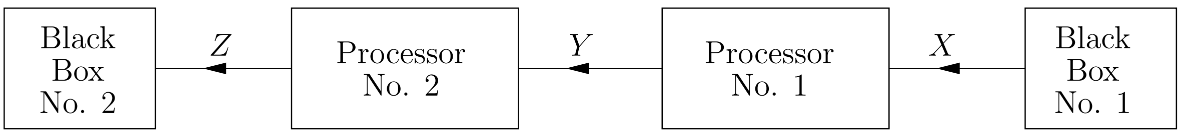


Figure 4.5: The conceptual situation for the Data Processing Lemma.

(A more elegant way to state (4.26) is to say that the sequence X, Y, Z is a *Markov chain.*)

The Data Processing Lemma states essentially that information cannot be increased by any sort of processing (although it can perhaps be put into a more convenient form).

Data Processing Lemma: When (4.26) holds, i.e., when X, Y, Z is a Markov chain, then

$$I(X;Z) \leq I(X;Y) \tag{4.27}$$

and

$$I(X;Z) \leq I(Y;Z). \tag{4.28}$$

Proof: We first observe that, because of Corollary 2 to Theorem 1.2, (4.26) implies that

$$H(Z|XY) = H(Z|Y) \tag{4.29}$$

and hence

$$\begin{aligned} I(Y;Z) &= H(Z) - H(Z|Y) \\ &= H(Z) - H(Z|XY) \\ &\geq H(Z) - H(Z|X) = I(X;Z) \end{aligned}$$

which proves (4.28). To prove (4.27), we need only show that Z, Y, X (the reverse of the sequence X, Y, Z) is also a Markov chain. We prove this by noting that

$$\begin{aligned} H(XYZ) &= H(Y) + H(X|Y) + H(Z|XY) \\ &= H(Y) + H(X|Y) + H(Z|Y) \end{aligned}$$

where we have used (4.29), and that

$$H(XYZ) = H(Y) + H(Z|Y) + H(X|YZ).$$

Equating these two expressions for $H(XYZ)$ shows that

$$H(X|YZ) = H(X|Y)$$

which by Corollary 2 to Theorem 1.2 is equivalent to

$$P(x|yz) = P(x|y) \tag{4.30}$$

for all y such that $P(y) \neq 0$. But (4.30) is just the statement that Z, Y, X is a Markov chain. □

The alert reader will have noticed that (4.29) can be used instead of (4.26), to which it is equivalent, as the definition of a Markov chain of length 3. In fact Markov chains of all lengths can also be formulated in terms of uncertainties, which formulation greatly simplifies the proofs of their properties, e.g., the proof that the reverse of a Markov chain (of any length) is again a Markov chain.

Now, for the first time in our study of information theory, we introduce the notion of "errors". Suppose we think of the random variable $\hat{U}$ as being an *estimate* of the random variable U. For this to make sense, $\hat{U}$ needs to take on values from the same alphabet as U. Then an *error* is just the event that $\hat{U} \neq U$ and the probability of error, P_e, is thus

$$P_e = P(\hat{U} \neq U). \tag{4.31}$$

We now are ready for one of the most interesting and important results in information theory, one that relates P_e to the conditional uncertainty $H(U|\hat{U})$.

Fano's Lemma: Suppose that U and $\hat{U}$ are L-ary random variables with the same alphabet and that the error probability P_e is defined by (4.31), then

$$h(P_e) + P_e \log_2(L-1) \geq H(U|\hat{U}) \tag{4.32}$$

where the uncertainty $H(U|\hat{U})$ is in bits.

Proof: We begin by defining the random variable Z as the indicator random variable for an error, i.e.,

$$Z = \begin{cases} 0 & \text{when } \hat{U} = U \\ 1 & \text{when } \hat{U} \neq U, \end{cases}$$

which of course implies by virtue of (4.31) that

$$H(Z) = h(P_e). \tag{4.33}$$

Next, we note that

$$\begin{aligned} H(UZ|\hat{U}) &= H(U|\hat{U}) + H(Z|U\hat{U}) \\ &= H(U|\hat{U}), \end{aligned}$$

as follows from (1.33) and the fact that U and $\hat{U}$ uniquely determine Z. Thus

$$\begin{aligned} H(U|\hat{U}) &= H(UZ|\hat{U}) \\ &= H(Z|\hat{U}) + H(U|\hat{U}Z) \\ &\leq H(Z) + H(U|\hat{U}Z) \end{aligned} \tag{4.34}$$

where we have made use of (1.21). But

$$H(U|\hat{U}, Z = 0) = 0 \tag{4.35}$$

since U is uniquely determined in this case, and

$$H(U|\hat{U}, Z = 1) \leq \log_2(L-1) \tag{4.36}$$

since, for each value $\hat{u}$ of $\hat{U}$, $\#(\text{supp } P_{U|\hat{U}Z}(\cdot|\hat{u}, 1)) \leq L - 1$, i.e., there are at most $L-1$ values of U with non-zero probability when $Z = 1$. Equations (4.35) and (4.36) imply

$$\begin{aligned} H(U|\hat{U}Z) &\leq P(Z = 1)\log_2(L-1) = \\ &= P_e \log_2(L-1). \end{aligned} \tag{4.37}$$

Substituting (4.33) and (4.37) into (4.34), we obtain (4.32) as was to be shown. □

We now turn to the interpretation of Fano's Lemma. First, we observe that the function on the left of (4.32) is the sum of a convex-∩ function of P_e. In Figure 4.6, we have sketched this sum function (which is also convex-∩ in P_e). The important observation is that this sum function is positive for all P_e such that $0 < P_e \leq 1$.

Thus, when a positive value of $H(U|\hat{U})$ is given, (4.32) implicitly specifies a positive lower bound on P_e.

Example 4.5: Suppose that U and $\hat{U}$ are binary-valued (i.e., $L = 2$) and that $H(U|\hat{U}) = \frac{1}{2}$ 4.32) gives

$$h(P_e) \geq \frac{1}{2}$$

or, equivalently,

$$.110 \leq P_e \leq .890.$$

The fact that here (4.32) also gives a non-trivial upper bound on P_e is a consequence of the fact that the given $H(U|\hat{U})$ exceeds the value taken on by the left side of (4.32) when $P_e = 1$, namely $\log_2(L-1)$. For instance, with $L = 3$ and $H(U|\hat{U}) = \frac{1}{2}$ bits we would obtain from (4.32)

$$h(P_e) + P_e \geq \frac{1}{2}$$

which is equivalent to

$$P_e \geq .084.$$

The trivial upper bound $P_e \leq 1$ is now the only upper bound that can be asserted.

A review of the proof of Fano's Lemma will reveal that *equality holds in (4.32) if and only if the probability of an error given that $\hat{U} = u$ is the same for all u and when there is an error (i.e., when $U \neq \hat{U}$), the $L-1$ erroneous values of U are always equally likely.* It follows that (4.32) gives the strongest possible lower bound on P_e, given only $H(U|\hat{U})$ and the size, L, of the alphabet for U.

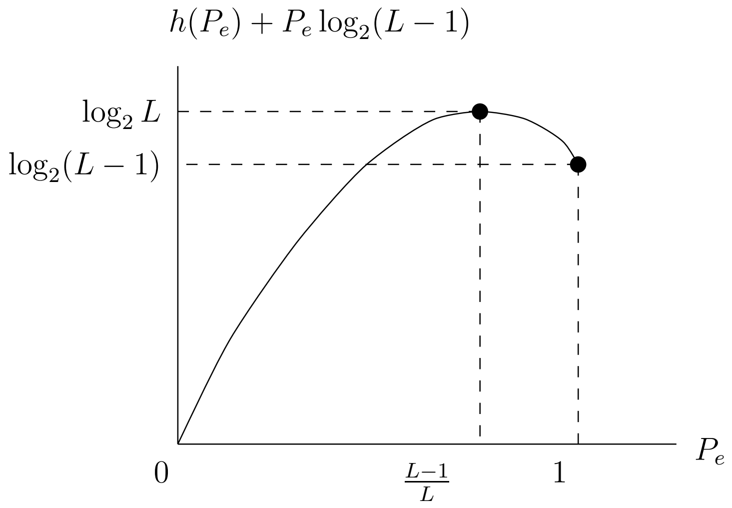


Figure 4.6: The conceptual situation for the Data Processing Lemma.

4.4 The Converse to the Noisy Coding Theorem for a DMC Used Without Feedback

We will now show that it is impossible to transmit "information" "reliably" through a DMC at a "rate" above its capacity. Without loss of essential generality, we suppose that the "information" to be transmitted is the output from a *binary symmetric source* (BSS), which is the DMS with $P_U(0) = P_U(1) = \frac{1}{2}$ whose "rate" is $H(U) = 1$ bit per source letter. It is common practice to call the letters in the binary output sequence $U_1, U_2, U_3, \ldots$ of the BSS *information bits* because each carries one bit of information, although this is a slight abuse of terminology. The full situation that we wish to consider is shown in Figure 4.7. Notice that we have assumed that the DMC is being used without feedback. Thus, by Theorem 4.1,

$$P(y_1, \ldots, y_n | x_1, \ldots, x_N) = \prod_{i=1}^{N} P(y_i | x_i)$$

which in turn implies

$$H(Y_1 \ldots Y_N | X_1 \ldots X_N) = \sum_{i=1}^{N} H(Y_i | X_i). \tag{4.38}$$

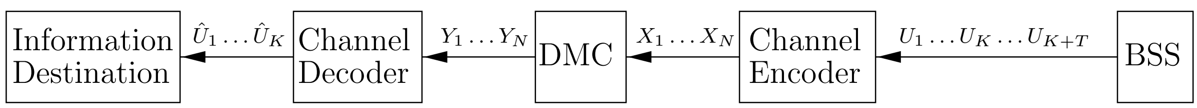


Figure 4.7: Conceptual Situation for the Noisy Coding Theorem

Next, we note that Figure 4.7 implies that the decisions about the first K information bits are made by the "channel decoder" using only the first N digits received over the channel. The "channel encoder" uses the first $K + T$ information bits to determine the first N transmitted digits for the channel – normally one would expect $T = 0$, but we do not wish to force this choice on the encoder. To describe the situation shown in Figure 4.7, we will say that K information bits are sent to their destination via N uses of the DMC without feedback. Notice that the rate of transmission, R, is

$$R = \frac{K}{N} \text{bits/use} \tag{4.39}$$

where "use" means "use of the DMC".

If we consider the "Channel Encoder" and the "DMC with Channel Decoder" in Figure 4.7 as "Processor No. 1" and "Processor No. 2", respectively, in Figure 4.5, we can use the Data Processing Lemma to conclude that

$$I(U_1 \dots U_K \dots U_{K+T}; \hat{U}_1 \dots \hat{U}_K) \leq I(X_1 \dots X_N; \hat{U}_1 \dots \hat{U}_K)$$

which further implies

$$I(U_1 \dots U_K; \hat{U}_1 \dots \hat{U}_K) \leq I(X_1 \dots X_N; \hat{U}_1 \dots \hat{U}_K) \tag{4.40}$$

Next, we consider the "DMC" and the "Channel Decoder" in Figure 4.7 as "Processor No. 1" and "Processor No. 2", respectively, in Figure 4.5 so that we can use the Data Processing Lemma again to conclude that

$$I(X_1 \dots X_N; \hat{U}_1 \dots \hat{U}_K) \leq I(X_1 \dots X_N; Y_1 \dots Y_N). \tag{4.41}$$

Combining (4.40) and (4.41), we obtain the fundamental inequality

$$\boxed{I(U_1 \dots U_K; \hat{U}_1 \dots \hat{U}_K) \leq I(X_1 \dots X_N; Y_1 \dots Y_N).} \tag{4.42}$$

To proceed further, we next note that

$$\begin{aligned} I(X_1 \dots X_N; Y_1 \dots Y_N) &= H(Y_1 \dots Y_N) - H(Y_1 \dots Y_N | X_1 \dots X_N) \\ &= H(Y_1 \dots Y_N) - \sum_{i=1}^{N} H(Y_i | X_i) \end{aligned} \tag{4.43}$$

where we have made use of (4.38). But, from (1.33) and Theorem 1.2 it follows that

$$H(Y_1 \dots Y_N) \leq \sum_{i=1}^{N} H(Y_i)$$

which, upon substitution into (4.43) gives

$$\begin{aligned} I(X_1 \dots X_N; Y_1 \dots Y_N) &\leq \sum_{i=1}^{N} [H(Y_i) - H(Y_i | X_i)] \\ &= \sum_{i=1}^{N} I(X_i; Y_i) \\ &\leq NC \end{aligned} \tag{4.44}$$

where we have made use of the definition of channel capacity.

Combining (4.42) and (4.44) gives

$$\boxed{I(U_1 \dots U_K; \hat{U}_1 \dots \hat{U}_K) \leq NC} \tag{4.45}$$

which is both pleasingly simple and intuitively satisfying.

Finally, we need to consider the appropriate meaning of "reliability". If K were very large (say $6 \cdot 10^{23}$), we would not be distressed by a few errors in the decoded sequence $\hat{U}_1, \ldots, \hat{U}_K$. Our real interest is in the fraction of these digits that are in error, i.e., in the *bit error probability*

$$P_\mathrm{b} = \frac{1}{K}\sum_{i=1}^{K} P_{\mathrm{e}i}, \tag{4.46}$$

where we have used the notation

$$P_{\mathrm{e}i} = P(\hat{U}_i \neq U_i). \tag{4.47}$$

Our task is to show that, when $R > C$, there is a positive lower bound on P_b that no manner of coding can overcome. As a first step in this direction, we note that

$$\begin{aligned} H(U_1 \ldots U_K|\hat{U}_1 \ldots \hat{U}_K) &= H(U_1 \ldots U_K) - I(U_1 \ldots U_K; \hat{U}_1 \ldots \hat{U}_K) \\ &= K - I(U_1 \ldots U_K; \hat{U}_1 \ldots \hat{U}_K) \\ &\geq K - NC \\ &= N(R - C) \end{aligned} \tag{4.48}$$

where we have made use of (4.45) and (4.39). To proceed further, we observe that

$$\begin{aligned} H(U_1 \ldots U_K|\hat{U}_1 \ldots \hat{U}_K) &= \sum_{i=1}^{K} H(U_i|\hat{U}_1 \ldots \hat{U}_K U_1 \ldots U_{i-1}) \\ &\leq \sum_{i=1}^{K} H(U_i|\hat{U}_i) \end{aligned} \tag{4.49}$$

since further conditioning can only reduce uncertainty. Substituting (4.49) into (4.48) gives

$$\sum_{i=1}^{K} H(U_i|\hat{U}_i) \geq N(R - C). \tag{4.50}$$

We can now use Fano's Lemma to assert that

$$\sum_{i=1}^{K} H(U_i|\hat{U}_i) \leq \sum_{i=1}^{K} h(P_{\mathrm{e}i}) \tag{4.51}$$

where we have used the definition (4.47). Combining (4.51) and (4.50), we have

$$\frac{1}{K}\sum_{i=1}^{K} h(P_{\mathrm{e}i}) \geq \frac{N}{K}(R - C) = 1 - \frac{C}{R}, \tag{4.52}$$

where we have again made use of the definition (4.39). The final step is to note (see Problem 4.4) that, because $h(p)$ is convex-$\cap$ in p for $0 \leq p \leq 1$,

$$\frac{1}{K}\sum_{i=1}^{K} h(P_{\mathrm{e}i}) \leq h\left(\frac{1}{K}\sum_{i=1}^{K} P_{\mathrm{e}i}\right) = h(P_\mathrm{b}), \tag{4.53}$$

where we have made use of the definition (4.46) of P_b. Substituting (4.53) into (4.52) now gives the eloquently simple result

$$h(P_\mathrm{b}) \geq 1 - \frac{C}{R}. \tag{4.54}$$

Thus, we have at long last arrived at:

The Converse Part of the Noisy Coding Theorem: If information bits from a BSS are sent to their destination at a Rate R (in bits per channel use) via a DMC of capacity C (in bits per use) without feedback, then the bit error probability at the destination satisfies

$$P_{\mathrm{b}} \geq h^{-1}\left(1 - \frac{C}{R}\right), \text{ if } R > C. \tag{4.55}$$

(Here, we have written h^{-1} to denote the inverse binary entropy function defined by $h^{-1}(x) = \min\{p : h(p) = x\}$, where the minimum is selected in order to make the inverse unique.)

Example 4.6: Suppose the DMC has capacity $C = \frac{1}{4}$ bit/use and that the BSS is transmitted at a rate $R = \frac{1}{2}$ bit/use. Then (4.55) requires

$$P_{\mathrm{b}} \geq h^{-1}\left(1 - \frac{1}{2}\right) = .110.$$

Thus, at least 11% of the information bits must be decoded incorrectly. (Note that we could also have used (4.54) to assert that $P_{\mathrm{b}} \leq .890$. This necessity can be seen from the fact that, if we could make P_{b} exceed .890, we could complement the decoder output to achieve P_{b} less than .110.)

We see that, whenever $R > C$, (4.55) will specify a positive lower bound on P_{b} that no amount of encoder and decoder complexity can overcome.

The reader may well wonder whether (4.55) still holds when feedback is permitted from the output of the DMC to the channel encoder. The answer is a somewhat surprising yes. To arrive at (4.44), we made strong use of the fact that the DMC was used without feedback – in fact (4.44) may not hold when feedback is present. However, (4.45) can still be shown to hold (cf. Problem 4.7) so that the converse (4.55) still holds true when feedback is present. This fact is usually stated as saying that *feedback does not increase the capacity of a DMC*. This does *not* mean, however, that feedback is of no value when transmitting information over a DMC. When $R < C$, feedback can often be used to simplify the encoder and decoder that would be required to achieve some specified P_{b}.

4.5 The Noisy Coding Theorem for a DMC

Now that we have seen what is impossible to do, we turn to see what is possible when one wishes to transmit a BSS over a DMC. We consider here *block coding* in which each "block" of N channel digits is determined by a corresponding "block" of K information bits. The decoder makes its decisions on a block of K information bits by an examination of only the corresponding received block of N digits. The rate R, and the bit error probability, P_{b}, are already defined by (4.39) and (4.46), respectively.

We now introduce the *block error probability*, P_{B}, which is defined as

$$P_{\mathrm{B}} = P(\hat{U}_1 \ldots \hat{U}_K \neq U_1 \ldots U_K). \tag{4.56}$$

Notice that, when a block error occurs, there must be at least one bit error but no more than K bit errors among the K decoded information bits. Thus, it follows that

$$\frac{1}{K} P_{\mathrm{B}} \leq P_{\mathrm{b}} \leq P_{\mathrm{B}} \tag{4.57}$$

or, equivalently,

$$P_{\mathrm{b}} \leq P_{\mathrm{B}} \leq K P_{\mathrm{b}}. \tag{4.58}$$

We have earlier argued that P_b is the quantity of real interest, and in fact have proved that P_b cannot be very small when $R > C$. We note from (4.58) that this also implies that P_B cannot be very small when $R > C$. We now want assurance that P_b can be made very small when $R < C$, but we will in fact get the even greater assurance that P_B can be made very small (as this will by (4.58) also imply that P_b can be made very small). To get the strongest results, one should always use bit error probability in converses to coding theorems, but always use block error probability in the coding theorem itself – and this is what we shall do.

The Noisy Coding Theorem for a DMC: Consider transmitting information bits from a BSS to their destination at a rate $R = \frac{K}{N}$ using block coding with blocklength N via a DMC of capacity C (in bits per use) used without feedback. Then, given any $\varepsilon > 0$, provided that $R < C$, one can always, by choosing N sufficiently large and designing appropriate encoders and decoders, achieve

$$P_\mathrm{B} < \varepsilon.$$

This theorem was the "bombshell" in Shannon's 1948 paper. Prior to its appearance, it was generally believed that in order to make communications more reliable it was necessary to reduce the rate of transmission (or, equivalently, to increase the "signal-to-noise-ratio", as the well-educated communications engineer of 1947 would have said it.) Shannon dispelled such myths forever – in theory, they have been slow to die in practice! Shannon showed that, provided $R < C$, increased reliability could be purchased entirely by increased complexity in the coding system – with no change in the signal-to-noise-ratio!

We shall not now give a proof of the above theorem despite its paramount importance in information theory and despite the fact that it is not particularly difficult to prove. There are two reasons for postponing this proof. The first is that, without some indication of how large N needs to be for a given ε (i.e., how complex the encoder and decoder need to be to achieve a given reliability), the theorem lacks a vital practical consideration. The second is that the proof of the theorem will be much more meaningful after we have further developed our understanding of channel encoders and decoders, as we shall do in the next chapter.

4.6 Supplementary Notes for Chapter 4: Convexity and Jensen's Inequality

A. Functions of One variable

Intervals on the real line may be of three kinds:

(i) *closed* (contain both endpoints): $[a, b]$ where $a \leq b$.

(ii) *open* (contain no endpoints): (a, b) where $a < b$ or (a, ∞) or $(-\infty, b)$ or $(-\infty, \infty)$.

(iii) *semi-open* (contain one endpoint): $[a, b)$ or $(a, b]$ where $a < b$, or $[a, \infty)$ or $(-\infty, b]$.

The length of the interval is $b - a$, or ∞, as appropriate.

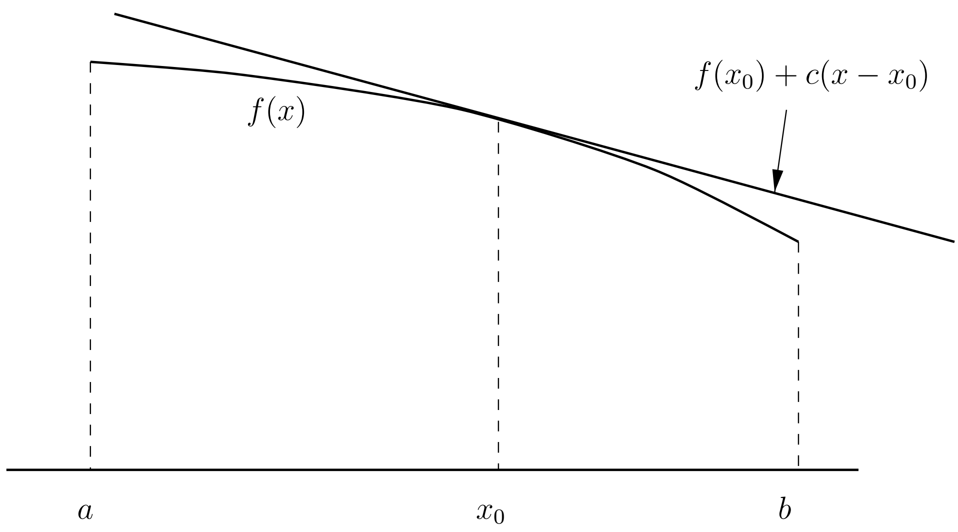
In what follows, $\mathcal{I}$ denotes an interval of non-zero length on the real line and f denotes a real-valued function whose domain inludes $\mathcal{I}$.

Definition 4-1: f is convex-$\cap$ (also called "concave") on $\mathcal{I}$ if, for each interior point x_0 of $\mathcal{I}$, there exists a real number c (which may depend on x_0) such that

$$f(x) \leq f(x_0) + c(x - x_0), \text{ all } x \text{ in } \mathcal{I}. \tag{4.59}$$

The convexity-$\cap$ (or "concavity") is said to be *strict* when the inequality (4.59) is strict whenever $x \neq x_0$.

Pictorially:



For any choice of x_0, the function lies on or below some line touching the function at $x = x_0$.

Remark 4-1: The convexity will be strict unless and only unless f is linear (i.e., of the form $mx + d$) on some subinterval of $\mathcal{I}$ of positive length.

The following test covers most functions of interest in engineering applications.

A Test for Convexity: If f is continous in $\mathcal{I}$ and f" exists and is continous in the interior of $\mathcal{I}$, then f is convex-$\cap$ on $\mathcal{I}$ if and only if $f''(x) \leq 0$ at every interior point x of $\mathcal{I}$. Moreover, the convexity is strict unless and only unless $f''(x) = 0$ for all x in some subinterval of $\mathcal{I}$ of positive length.

Proof: For such a function f as hypothesized, Taylor's theorem implies that for any interior point x_0 of $\mathcal{I}$ and any x in $\mathcal{I}$,

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_1) \tag{4.60}$$

where x_1 (which may depend on x) lies strictly between x_0 and x, and hence x_1 is an interior point of $\mathcal{I}$. Thus, when f'' is at most 0 for all interior points of $\mathcal{I}$, (4-2) implies (4-1) and thus f is convex-$\cap$. By Remark 4-1, the convexity will be strict unless and only unless f'' vanishes in a subinterval of $\mathcal{I}$ of positive length.

Conversely, suppose f'' is positive at some interior point of $\mathcal{I}$. The continuity of f'' implies that one can find a positive length subinterval $\mathcal{J}$ of $\mathcal{I}$ such that f'' is positive everywhere in $\mathcal{J}$. Hence $-f$ is strictly convex-$\cap$ on $\mathcal{J}$. Thus, when x_0 is chosen as an interior point of $\mathcal{J}$ inequality (4-1) cannot hold for all x in $\mathcal{J}$, much less for all x in $\mathcal{I}$, so f cannot be convex-$\cap$. □

We will say that a random variable X is *essentially constant* if there exists a value x of X such that $P(X = x) = 1$.

Jensen's Inequality: If f is convex-$\cap$ on $\mathcal{I}$ and X is a random variable taking values in $\mathcal{I}$, then

$$\mathrm{E}[f(X)] \leq f(\mathrm{E}[X]). \tag{4.61}$$

Moreover, if f is strictly convex-$\cap$ on $\mathcal{I}$, then the inequality (4.61) is strict unless and only unless X is essentially constant.

Proof: If X is essentially constant, then (4.61) holds with equality for any f so we may suppose hereafter that X not essentially constant. This implies that $x_0 = \mathrm{E}[X]$ is an interior point of $\mathcal{I}$. Thus, (4.59) implies that for some c

$$f(x) \leq f(\mathrm{E}[X]) + c(x - \mathrm{E}[X]). \tag{4.62}$$

Taking expectations in (4.62) immediately gives (4.61). Moreover, if f is strictly convex-$\cap$, the fact that implies that the probability that (4.62) holds with strict inequality is greater than 0 so that taking expectations in (4.62) yields $\mathrm{E}[f(X)] < f(\mathrm{E}[X])$. □

Definition 4-2: f is convex-$\cup$ (also called "convex") on $\mathcal{I}$ if $-f$ is convex-$\cap$ on $\mathcal{I}$ (or, equivalently, if Definition 4-1 is satisfied when the direction of inequality (4.59) is reversed.)

Remark 4-2: f is both convex-$\cap$ and convex-$\cup$ on $\mathcal{I}$ if and only if f is linear on $\mathcal{I}$ (i.e., $f(x) = mx + d$ for all x in $\mathcal{I}$.)

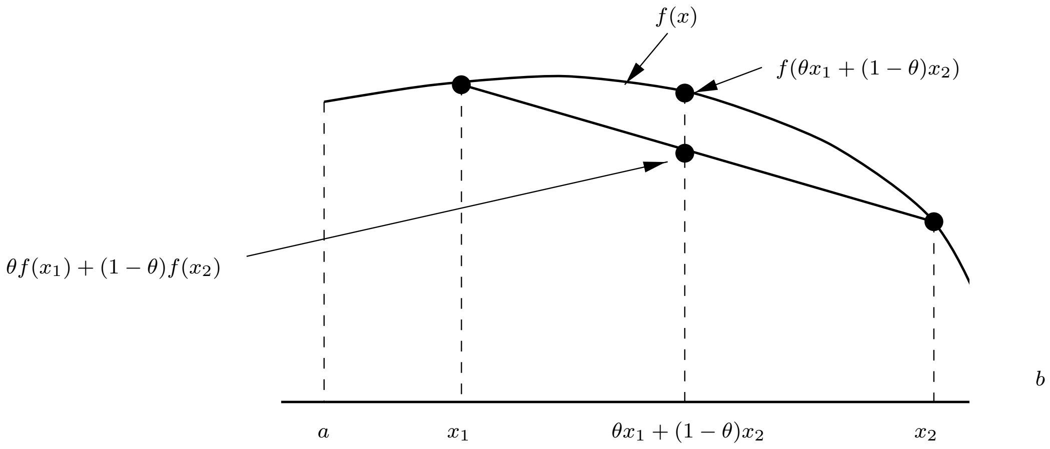
The reader is forewarned that the above definition of convexity, despite its convenience, is not that most often used in information theory. The more usual definition is the characterization of convexity as given in the following theorem.

Theorem 4-1: f is convex-$\cap$ on $\mathcal{I}$ if and only if for every x_1 and x_2 in $\mathcal{I}$,

$$\theta f(x_1) + (1-\theta) f(x_2) \leq f[\theta x_1 + (1-\theta) x_2] \qquad \text{for } 0 < \theta < 1. \tag{4.63}$$

The convexity is strict if and only if inequality (4.63) is strict whenever $x_1 \neq x_2$.

Pictorially:



The function always lies on or above its chords.

Proof: Suppose that f is convex-$\cap$ on $\mathcal{I}$. For x_1, x_2, and θ as specified, define the discrete random variable X such that X takes on values x_1 and x_2 with probabilities θ and $1-\theta$, respectively, where we may assume that $x_1 \neq x_2$. Then Jensen's inequality gives

$$\mathrm{E}[f(X)] = \theta f(x_1) + (1-\theta) f(x_2) \leq f(\mathrm{E}[X]) = f[\theta x_1 + (1-\theta) x_2]$$

so that (4.63) indeed holds. Moreover, Jensen's inequality gives strict inequality here if f is strictly convex-$\cap$.

Conversely, suppose that f is not convex-$\cap$ on $\mathcal{I}$. Let x_0 be a point in $\mathcal{I}$ where (4.59) cannot be satisfied, i.e., where we cannot place a straight line passing through the function at $x = x_0$ that lies everywhere above the function. But then there must be some such line, $f(x_0) + c(x - x_0)$ that lies below the function at some points x_1 and x_2 in $\mathcal{I}$ such that $x_1 < x_0 < x_2$, i.e.,

$$f(x_1) > f(x_0) + c(x_1 - x_0) \tag{4.64}$$

and

$$f(x_2) > f(x_0) + c(x_2 - x_0). \tag{4.65}$$

Now define θ by

$$x_0 = \theta x_1 + (1-\theta) x_2 \tag{4.66}$$

which ensures that $0 < \theta < 1$. Multiplying (4.64) and (4.65) by θ and $1-\theta$, respectively, then adding gives

$$\begin{aligned}
\theta f(x_1) + (1-\theta) f(x_2) &> f(x_0) + c[\theta x_1 + (1-\theta) x_2 - x_0] \\
&= f(x_0) && \text{by (4.66)} \\
&= f[\theta x_1 + (1-\theta) x_2] && \text{again by (4.66).}
\end{aligned}$$

Thus, (4.63) does *not* hold everywhere in $\mathcal{I}$. $\square$

B. Functions of Many Variables

We consider real-valued functions of n real variables, $f(\underline{x})$ where $\underline{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$.

A subset $\mathcal{S}$ of $\mathbb{R}^n$ is said to be a *convex region* if, for every pair of points $\underline{x}_1$ and $\underline{x}_2$ in $\mathcal{S}$, the line between $\underline{x}_1$ and $\underline{x}_2$ lies entirely in $\mathcal{S}$. In other words, $\mathcal{S}$ is a convex region if, for every pair $\underline{x}_1$ and $\underline{x}_2$ in $\mathcal{S}$, $\theta\underline{x}_1 + (1-\theta)\underline{x}_2$ is also in $\mathcal{S}$ for all θ such that $0 < \theta < 1$. For $n = 1$, $\mathcal{S}$ is a convex region if and only if $\mathcal{S}$ is an interval. For $n > 1$, convex regions are the natural generalization of intervals.

Definition 4-3: f is convex-$\cap$ (also called "concave") on the convex region $\mathcal{S}$ if, for each interior point $\underline{x}_0$ of $\mathcal{S}$, there exists a $\underline{c} \in \mathbb{R}^n$ (which may depend on $\underline{x}_0$) such that

$$f(\underline{x}) \le f(\underline{x}_0) + \underline{c}^T(\underline{x} - \underline{x}_0), \qquad \text{all } \underline{x} \text{ in } \mathcal{S}. \tag{4.67}$$

The convexity is said to be *strict* when the inequality (4.67) is strict whenever $\underline{x} \neq \underline{x}_0$.

In words, f is convex-$\cap$ if, for any choice of $\underline{x}_0$, the function lies on or below some hyperplane that touches the function at $\underline{x} = \underline{x}_0$.

Jensen's Inequality: If f is convex-$\cap$ on the convex region $\mathcal{S}$ and is a random vector taking values in $\mathcal{S}$, then

$$\mathrm{E}[f(\underline{X})] \le f(\mathrm{E}[\underline{X}]). \tag{4.68}$$

Moreover, if f is strictly convex on $\mathcal{S}$, then the inequality (4.68) is strict unless and only unless $\underline{X}$ is essentially constant.

Definition 4-4: f is *convex*-$\cup$ (also called "convex") on the convex region $\mathcal{S}$ if $-f$ is convex-$\cap$ on $\mathcal{S}$ (or, equivalently, if Definition 4-3 is satisfied when the direction of inequality (4.67) is reversed.)

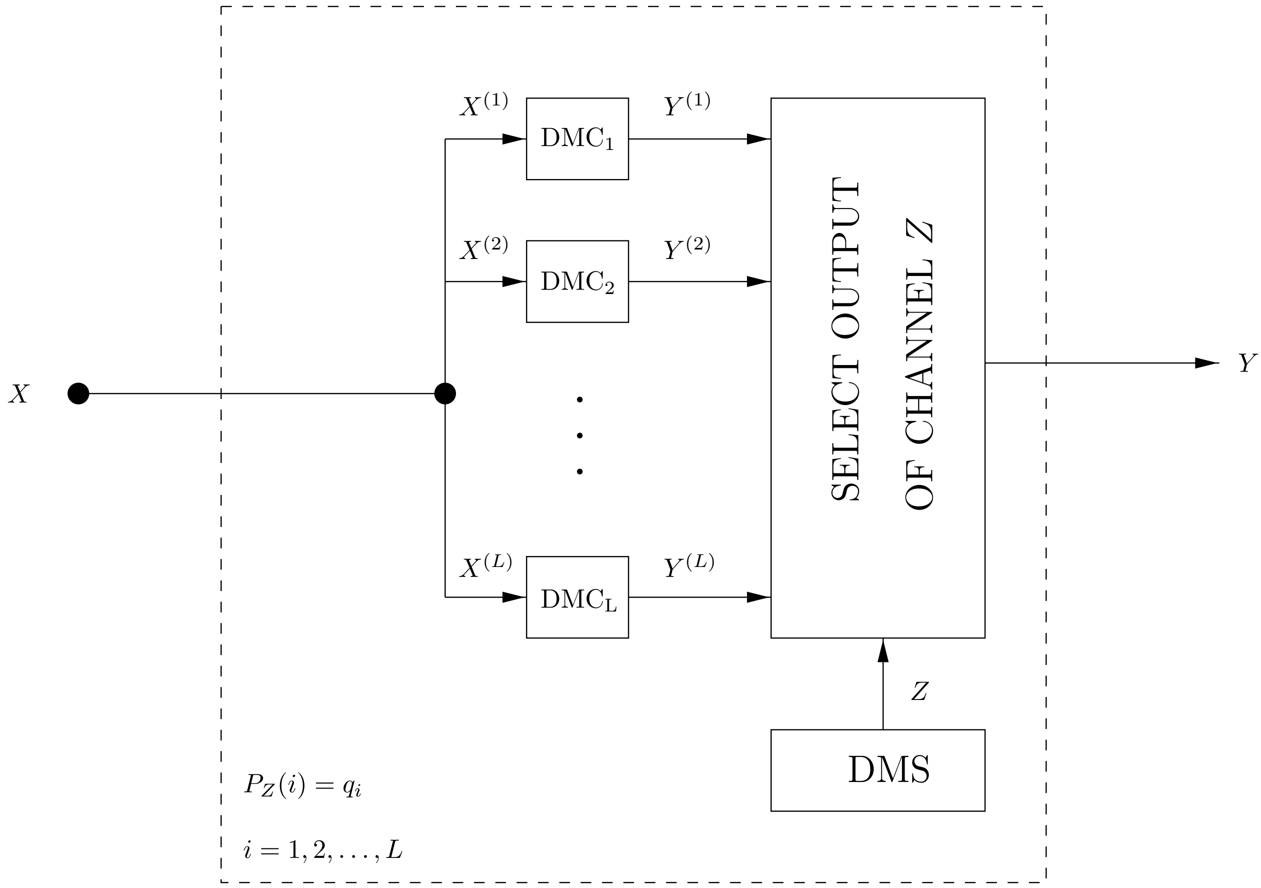
Theorem 4-2: f is convex-$\cap$ on the convex region $\mathcal{S}$ if and only if for every $\underline{x}_1$ and $\underline{x}_2$ in $\mathcal{S}$,

$$\theta f(\underline{x}_1) + (1-\theta) f(\underline{x}_2) \le f[\theta\underline{x}_1 + (1-\theta)\underline{x}_2]. \tag{4.69}$$

The convexity is strict if and only if inequality (4-10) is strict whenever $\underline{x}_1 \neq \underline{x}_2$.

Note again that the meaning of (4-10) is that the function always lies above or on its chords.

C. General DMC Created by L DMC's with Selection Probabilities $q_1, q_2, \ldots, q_L$



Assumptions:

(1) All component DMC's have the same input alphabet:

$$A = A^{(1)} = A^{(2)} = \cdots = A^{(L)}.$$

(2) The component DMC's have *disjoint* output alphabets:

$$B^{(i)} \cap B^{(j)} = \varnothing,\ i \neq j,$$

$$B = B^{(1)} \cup B^{(2)} \cup \cdots \cup B^{(L)}.$$

(3) X and Z are statistically independent.

Consequences:

(4) The composite channel is a DMC with transition probabilities

$$P_{Y|X}(b_j^{(i)}|a_k) = q_i P_{Y^{(i)}|X^{(i)}}(b_j^{(i)}|a_k).$$

(5) Y uniquely determines Z. by (2)

Claim:

$$I(X;Y) = \sum_{i=1}^{L} q_i I(X;Y^{(i)}).$$

Proof of Claim:

$$\begin{aligned} H(YZ) &= H(Y) + H(Z|Y) = H(Y) \\ &= H(Z) + H(Y|Z). && \text{by (5)} \end{aligned}$$

Thus,

$$\begin{aligned} H(Y) &= H(Z) + H(Y|Z) \\ &= H(Z) + \sum_{i=1}^{L} H(Y|Z=i)P_Z(i) \\ &= H(Z) + \sum_{i=1}^{L} H(Y^{(i)})q_i. \end{aligned}$$

Similarly,

$$\begin{aligned} H(YZ|X) &= H(Y|X) + H(Z|YX) = H(Y|X) && \text{by (5),} \\ &= H(Z|X) + H(Y|XZ) = H(Z) + H(Y|XZ) && \text{by (3).} \end{aligned}$$

Thus,

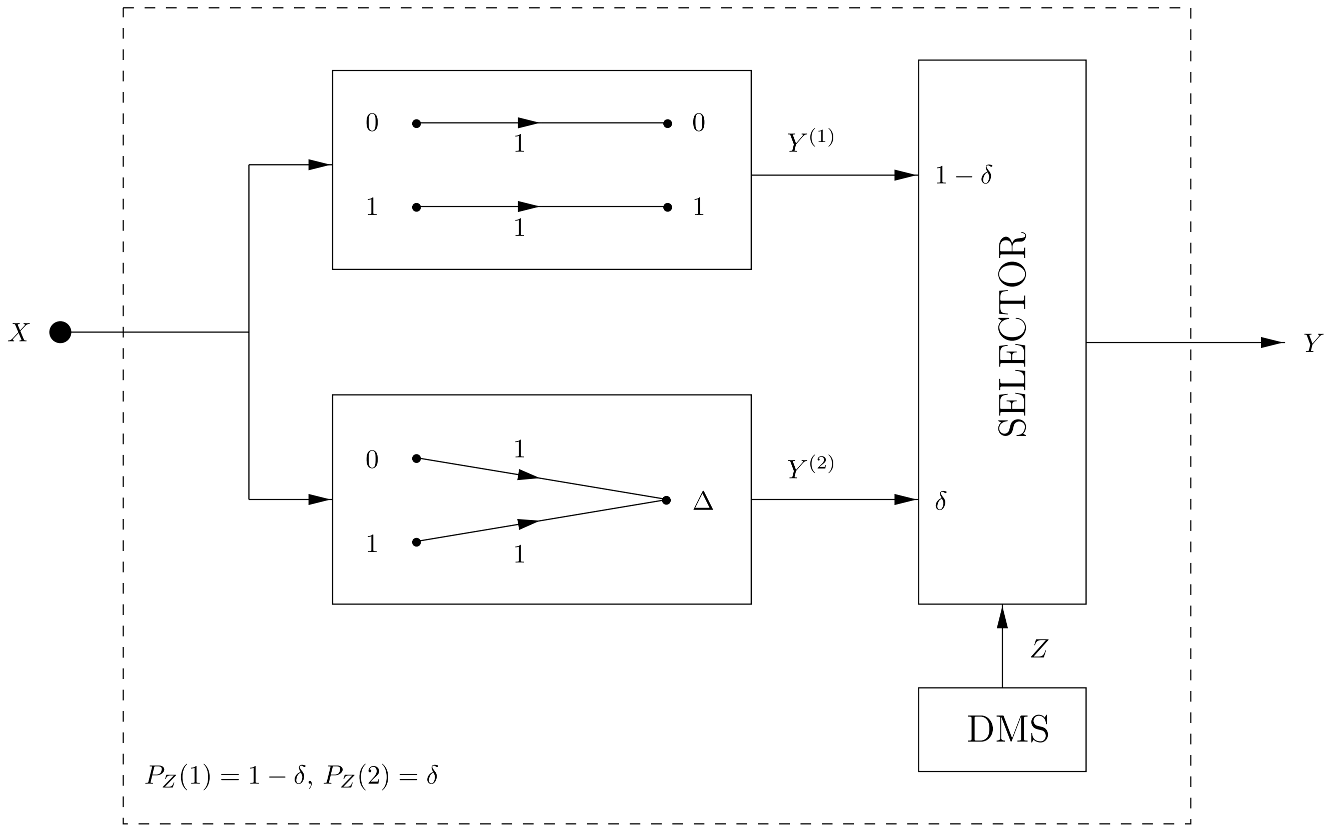
$$\begin{aligned} H(Y|X) &= H(Z) + H(Y|XZ) \\ &= H(Z) + \sum_{i=1}^{L} H(Y|X, Z=i)P_Z(i) \\ &= H(Z) + \sum_{i=1}^{L} H(Y^{(i)}|X)q_i. \end{aligned}$$

Hence, we have

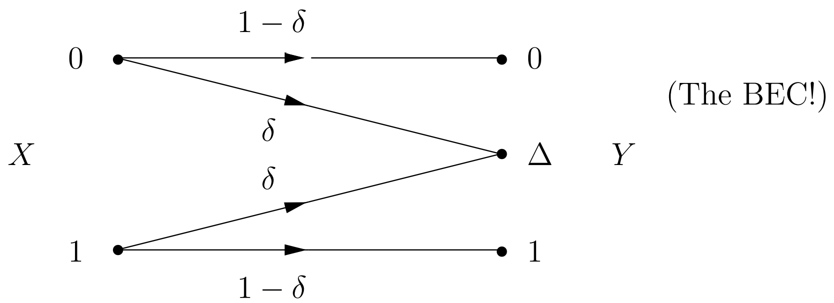
$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= \sum_{i=1}^{L} q_i[H(Y^{(i)}) - H(Y^{(i)}|X)] \\ &= \sum_{i=1}^{L} q_i I(X;Y^{(i)}) \end{aligned}$$

as was to be shown.

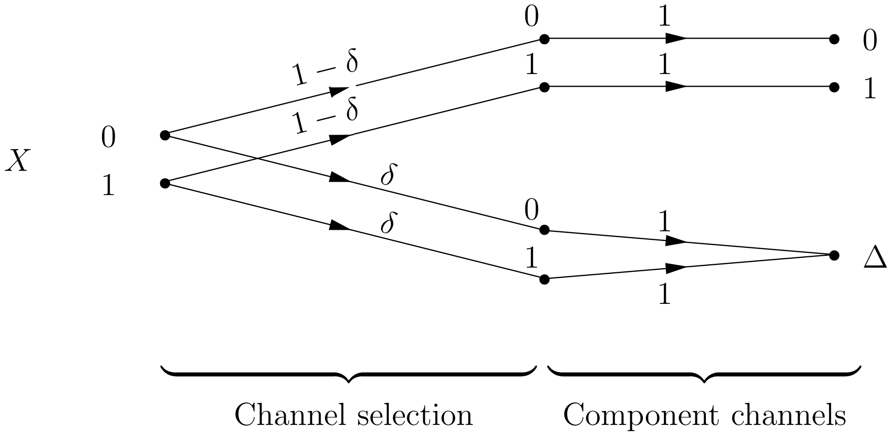
Example:



The composite channel is the following DMC:



The composite channel may be drawn more suggestively as the following cascade of channels:



Definition: A DMC is symmetric if it can be created by L strongly symetric DMC's with selection probabilities $q_1, q_2, \ldots, q_L$.

Theorem 4-3: For a symmetric DMC,

$$C = \sum_{i=1}^{L} q_i C_i$$

(where C_i is the capacity of the i-th strongly symmetric DMC and q_i is its selection probability) and $P_X(a_k) = 1/K$, al k, achieves capacity.

Proof:

$$I(X;Y) = \sum_{i=1}^{L} q_i I(X;Y^{(i)}) \text{ by the Claim.}$$

$$C = \max_{P_X} I(X;Y) \le \sum_{i=1}^{L} q_i \max_{P_X} I(X;Y^{(i)}) = \sum_{i=1}^{L} q_i C_i$$

with equality if and only if there is a P_X that simultaneously maximizes $I(X;Y^{(1)}), I(X;Y^{(2)}), \ldots, I(X;Y^{(L)})$. But since each component DMC is strongly symmetric, $P_X(a_k) = 1/K$, all k, is such a P_X. This proves the theorem. □

Algorithm for determining if a DMC is symmetric:

1. Partition the output letters into subsets $B^{(1)}, B^{(2)}, \ldots, B^{(L)}$ such that two output letters are in the same subset if and only if they have the "same focusing" (same ordered list of transition probabilities from the K input leters). Set $i = 1$.

2. Check to see that all input letters have the "same dispersion" (same ordered list of transition probabilities) into the subset $B^{(i)}$ of output letters. If yes, set q_i equal to the sum of the transition probabilities into $B^{(i)}$ from any input leter.

3. If $i = L$, stop. The channel is symmetric and the selection probabilities $q_1, q_2, \ldots, q_L$ have been found. If $i < L$, increase i by 1 and return to step 2.

Chapter 5

BLOCK CODING PRINCIPLES

5.1 Introduction

The rudiments of block coding for a noisy channel were already introduced on our discussion of the noisy coding theorem in Section 4.5. We now wish to generalize and deepen this discussion. We begin with the general definition of a block code for a channel whose input alphabet is $A = \{a_1, a_2, \ldots, a_L\}$. A *block code* of length N with M codewords for such a channel is a list $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$, each item of which is an N-tuple (or "row vector") of elements from A. The items in the list are of course called *codewords.* It is important to note that the codewords need not all be different, although "good" codes generally will have no codeword with multiple appearances.

Example 5.1: $([000], [011], [101], [110])$ is a block code of length $N = 3$ having $M = 4$ codewords. This code can be used on any binary-input channel (i.e., a channel with input alphabet $A = \{0, 1\}$) such as the BSC or the BEC.

The rate of a block code of length N with M codewords is defined as

$$R = \frac{\log_2 M}{N} \text{ bits/use,} \tag{5.1}$$

where "use" refers to "use of the channel". Notice that this is a true "information rate" only if the M codewords are distinct and equally likely, since only then are there $\log_2 M$ bits of uncertainty about which codeword is sent over the channel.

In Section 4.5 we considered the case where K output digits from a BSS were the "information bits" that selected which codeword was transmitted over the channel. In this case, we have $M = 2^K$ codewords, each having probability 2^{-K} of being transmitted. Now, however, we wish to be more general about both the number of codewords and their probabilities. Toward this end, we introduce the random variable Z, whose possible values are the integers from 1 through M, to specify which codeword is transmitted – in the manner that $Z = i$ specifies that $\underline{x}_i$ is transmitted.

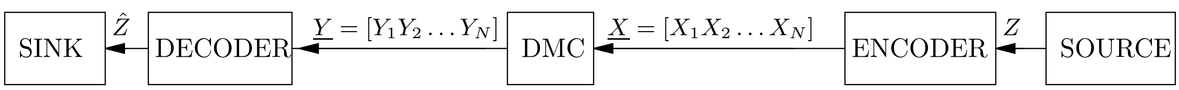


Figure 5.1: A Block Coding System for a DMC used without Feedback.

In the remainder of this chapter, we consider only block coding for a DMC *used without feedback* (so that we shall often make use of Equation 4.3 without further comment.) This implies that the probability distribution P_Z is entirely determined by the output of some information source (such as the BSS which would result in $P_Z(i) = 2^{-K}$ for $i = 1, 2, \ldots, 2^K$), independently of what might be received over the DMC when the codeword is transmitted. Thus, ARQ (automatic repeat/request) strategies and similar methods for making use of feedback information are outside the purview of this chapter.

The essentials of a block coding system for a DMC used without feedback are shown in Figure 5.1. The function of the encoder is to emit the particular codeword specified by Z, i.e., to emit

$$\underline{X} = \underline{x}_Z \tag{5.2}$$

where

$$\underline{X} = [X_1 X_2 \ldots X_N] \tag{5.3}$$

is the transmitted codeword.

Example 5.1 (cont.): The encoder of Figure 5.1 for this example realizes the following function:

$$\begin{array}{c|c} Z & \underline{X} \\ \hline 1 & [0\;0\;0] \\ 2 & [0\;1\;1] \\ 3 & [1\;0\;1] \\ 4 & [1\;1\;0] \end{array} .$$

The function of the decoder is to examine the received N-tuple

$$\underline{Y} = [Y_1, Y_2 \ldots Y_N] \tag{5.4}$$

and then to emit its decision $\hat{Z}$ about the value of the *codeword index* Z. Thus, a decoder is a mapping from N-tuples of channel output symbols into the set $\{1, 2, \ldots, M\}$ of possible values of Z, i.e.,

$$\hat{Z} = F(\underline{Y}) \tag{5.5}$$

for some function F that maps B^N (where B is the channel output alphabet) into $\{1, 2, \ldots, M\}$.

Example 5.1 (cont.): If the previous code were to be used on the BEC of Figure 4.1(b), one possible decoding function is described in Table 5.1.

Notice that the decoder decision $\hat{Z}$ implies the further decision that

$$\underline{\hat{X}} = \underline{x}_{\hat{Z}} \tag{5.6}$$

is the transmitted codeword. When the M codewords are distinct, the decision $\underline{\hat{X}}$ would also imply the decision $\hat{Z}$. In fact, decoders are often realized in practice by a device which emits $\underline{\hat{X}}$, the decision about $\underline{X}$, followed by an "encoder inverse" (which is usually a relatively simple device) to determine $\hat{Z}$ from $\underline{\hat{X}}$, which is possible in general if and only if the M codewords are distinct.

For a given DMC, given blocklength N and given code size M, there are

$$(L^N)^M = L^{MN}$$

different block codes (or, equivalently, different encoders) since there are L^N choices for each codeword in the list of M codewords, where L is the size of the channel input alphabet. There are

$$M^{J^N}$$

different decoders since there are M choices of $\hat{Z}$ for each of the J^N values of $\underline{Y}$, where J is the size of the channel output alphabet. Each such decoder could be used with any such encoder so there are

$$L^{MN} M^{J^N}$$

$\underline{Y}$	$\hat{Z} = F(\underline{Y})$	$\hat{\underline{X}} = \underline{x}_{\hat{Z}}$
[000]	1	[000]
[Δ00]	1	[000]
[0Δ0]	1	[000]
[00Δ]	1	[000]
[ΔΔ0]	1	[000]
[Δ0Δ]	1	[000]
[0ΔΔ]	1	[000]
[ΔΔΔ]	1	[000]
[011]	2	[011]
[Δ11]	2	[011]
[0Δ1]	2	[011]
[01Δ]	2	[011]
[ΔΔ1]	2	[011]
[Δ1Δ]	2	[011]
[101]	3	[101]
[Δ01]	3	[101]
[1Δ1]	3	[101]
[10Δ]	3	[101]
[1ΔΔ]	3	[101]
[110]	4	[110]
[Δ10]	4	[110]
[1Δ0]	4	[110]
[11Δ]	4	[110]
[111]	2	[011]
[100]	1	[000]
[010]	1	[000]
[001]	1	[000]

Table 5.1: A Possible Decoding Function for the Code in Example 5.1.

different coding systems. For instance, with $M = 4, N = 3, L = 2$ and $J = 3$ (as in Example 5.1), there are already enormously many different coding systems, namely

$$2^{12} \cdot 4^{3^3} = 2^{66} \approx 10^{20}.$$

How do we find a good system from this bewildering array of choices? Before we can answer this question, we must first decide what it really is that we want the coding system to do.

5.2 Encoding and Decoding Criteria

We begin by supposing that we have been given a block code of length N and size M for use on a particular channel, and that it is our task to design the decoder. The objective is to make the resulting information transmission system (from encoder input to decoder output) as reliable as possible. In Section 4.4, we argued that the *symbol error probability* P_{b} of (4.42) (which we there called the "bit error probability" because the symbols were binary digits) in the information symbols is the error probability of greatest practical interest, so one might suppose that this is the quantity that the decoder should minimize. However, to do this we would have to know specifically what the information symbols were and what the particular mapping from the information symbols to the codewords was. We can avoid these complicating matters entirely if we choose instead to work with the *block error probability* P_{B} of (4.52). Inequality (4.53b) ensures that if we minimize P_{B}, then we shall also at least roughly minimize

$P_{\rm b}$. Notice for the situation of Figure 5.1 that we can write $P_{\rm B}$ as

$$P_{\rm B} = P(\hat{Z} \neq Z) \tag{5.7}$$

because, whatever mapping actually exists between the information symbols and the codewords, it will always be true that the information symbols uniquely determine Z and vice-versa.

By focusing our attention on the block error probability $P_{\rm B}$ rather than on the symbol error probability $P_{\rm b}$, we are freed from worrying about the details of the mapping from information symbols to codewords. However, it is still not clear what we wish the decoder to do. In particular, there are at least three decoding criteria for each of which one can give a reasonable argument to justify its use, namely the smallness of:

(i) The block error probability $P_{\rm B}$ when the codeword probabilities are determined by the actual information source that we wish to transmit. The decoder that minimizes this $P_{\rm B}$ is called the *maximum a posteriori* (MAP) *decoder.*

(ii) The block error probability $P_{\rm B}$ when the codewords are assumed to be equally likely (whether or not they actually are). The decoder that minimizes this $P_{\rm B}$ is called the *maximum likelihood* (ML) *decoder.*

(iii) The *worst-case block error probability*, $(P_{\rm B})_{\rm wc}$, which for a given decoder is defined as

$$(P_{\rm B})_{\rm wc} = \max_{P_Z} P_{\rm B}. \tag{5.8}$$

The decoder which minimizes $(P_{\rm B})_{\rm wc}$ is called the *minimax decoder* because it minimizes the maximum block error probability for all possible assignments of probabilities to the codewords.

Criterion (i) is perhaps the natural choice, but several factors militate against its use. For one thing, we would need to know the information source statistics before we designed the decoder. For another, we would have to change the decoder if this source were changed. Because the sender may exercise his freedom to use the channel as he wishes, we might find ourselves with a very poor system after such a change even if previously the system was optimum. (In fact, some telephone companies are in economic trouble today because they optimized their systems for voice transmission but now must accommodate digital data transmission.) This discussion suggests that criterion (iii) might be the better one as it will lead to a system that will be the most "foolproof". But criterion (iii) is perhaps too conservative for most engineers' taste. It appeals to believers in "Murphy's Law" ("The worst thing that can happen will happen"). It is also usually a complicated task to find the minimax decoder.

The reader is not wrong if he has by now guessed that we will select criterion (ii). This criterion has many things in its favor. First, of course, it offers the convenience of not having to know the source statistics (which determine the codeword probabilities) before one designs the decoder. Secondly, it minimizes $P_{\rm B}$ for the important case when the source is a BSS, as this source yields equally likely codewords. In fact, the binary ($D = 2$) source coding schemes of Chapter 2 are just ways to convert other information sources to the BSS (or to a good approximation thereto) as we can see from the fact that each binary digit from the source encoder carries one bit of information (or almost this much). Thus by choosing the ML decoder, we choose to minimize $P_{\rm B}$ for the very source that will be created by applying a good "data compression" scheme to any given information source. This argument alone would justify our designation of the ML decoder as "optimum", but there is yet another point in its favor. The case of equally likely codewords is the case when the code is being used to transmit the most information ($\log_2 M$ bits) that is possible with M codewords so that a decoder which works well for this case ought certainly to perform satisfactorily when less information is being transmitted. In fact, we shall later give a rigorous verification of this plausibility argument. In effect, this verification will show that an ML decoder is also "almost minimax".

For all of the above reasons, we now commit ourselves to the following criterion:

Decoding Criterion: The measure of goodness for a decoder for a given block code and given channel is the smallness of the block error probability P_{B} when the codewords are equally likely. Thus, a decoder is optimum if and only if it is a maximum likelihood (ML) decoder.

This choice of decoding criterion forces upon us the following choice of an encoding criterion.

Encoding Criterion: The measure of goodness for a block code with M codewords of length N for a given channel is the smallness of the block error probability P_{B} when the codewords are equally likely and an ML decoder is used.

Perhaps this is the time to say that we have no intention actually to find optimum codes in general or to design true ML decoders in general. Our interest in optimality is only for the yardstick that it gives us to evaluate practical systems. It is generally a hopelessly difficult task to find the optimum code for a given DMC and given choice of M and N, and almost as difficult in practice to implement a true ML decoder. But if we know (or at least have a tight bound on) P_{B} for the optimum system, we can make sensible design choices in arriving at a practical coding system.

5.3 Calculating the Block Error Probability

In this section, we see how to make an exact calculation of P_{B}, from which we shall also see how to find the ML decoder and the MAP decoder for a given code and channel. Actually, it is easier to work with the probability $1 - P_{\mathrm{B}}$ that the block is correctly decoded. From (5.7) and (5.5), we have

$$\begin{aligned} 1 - P_{\mathrm{B}} &= P(Z = \hat{Z}) \\ &= P(Z = F(\underline{Y})). \end{aligned} \tag{5.9}$$

Invoking the Theorem on Total Probability, we can rewrite this as

$$\begin{aligned} 1 - P_{\mathrm{B}} &= \sum_{\underline{y}} P(Z = F(\underline{Y})|\underline{Y} = \underline{y}) P_{\underline{Y}}(\underline{y}) \\ &= \sum_{\underline{y}} P_{Z|\underline{Y}}(F(\underline{y})|\underline{y}) P_{\underline{Y}}(\underline{y}) = \sum_{\underline{y}} P_{Z\underline{Y}}(F(\underline{y}), \underline{y}), \end{aligned} \tag{5.10}$$

where the summation is over all N-tuples $\underline{y} = [y_1 y_2 \ldots y_N]$ of channel output symbols. We can now rewrite the right side of (5.10) as

$$1 - P_{\mathrm{B}} = \sum_{\underline{y}} P_{\underline{Y}|Z}(\underline{y}|F(\underline{y})) P_Z(F(\underline{y})). \tag{5.11}$$

But when $Z = F(\underline{y})$, the transmitted codeword is $\underline{X} = \underline{x}_{F(\underline{y})}$, so we can rewrite (5.11) as

$$1 - P_{\mathrm{B}} = \sum_{\underline{y}} P_{\underline{Y}|\underline{X}}\left(\underline{y}|\underline{x}_{F(\underline{y})}\right) P_Z(F(\underline{y})). \tag{5.12}$$

Equation (5.12) is our desired general expression for P_{B} and applies to any channel (not only to DMC's) and to any probability distribution P_Z for the codewords (not only to the case of equally likely codewords).

We can immediately draw two important inferences from (5.12). Because P_B will be minimized when we maximize the right side of (5.12) and because the choice of a decoder is precisely the choice of the function F in (5.12), we have as our first inference from (5.12):

> *The MAP decoding rule:* Choose the decoding function so that, for each N-tuple $\underline{y}$ of channel output symbols, $F(\underline{y}) = i$ where i is (any one of) the index(es) that maximizes
>
> $$P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)P_Z(i).$$

Because the ML decoding rule is just the MAP decoding rule for the special case when the codewords are equally likely (i.e., when $P_Z(i) = \frac{1}{M}$ for all i), we have as our second inference:

> *The ML decoding rule:* Choose the decoding function F so that, for each N-tuple $\underline{y}$ of channel output symbols, $F(\underline{y}) = i$ where i is (any one of) the index(es) that maximizes
>
> $$P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i),$$
>
> which in the special case of a DMC used without feedback becomes
>
> $$P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) = \prod_{j=1}^{N} P_{Y|X}(y_j|x_{ij}) \tag{5.13}$$
>
> where we have used the notation
>
> $$\underline{x}_i = [x_{i1}x_{i2}\ldots x_{iN}]. \tag{5.14}$$

The reader should now verify that the decoding function F specified in Example 5.1 above is indeed an ML decoding function for the given code and given DMC (the BEC). Because, for certain values of $\underline{y}$, the index i which maximizes (5.13) is not unique, there are other decoding functions which are also ML for this code and channel. (In fact, the reader may wish to verify that, when $0 < \delta < 1$, there are exactly $4 \cdot 2^6 = 256$ different ML decoders for this example, because there is one $\underline{y}$ (namely $[\Delta\Delta\Delta]$) for which all 4 choices of i maximize (5.13) and there are six values of $\underline{y}$ for which 2 choices of i maximize (5.13). When $\delta = 1$, however, every one of the $4^{27} = 2^{54} \approx 10^{16}$ possible decoding rules is ML. The reader may wish to reflect on the significance of this fact.)

Using (5.12), we can in a straightforward way calculate P_B for any block code, any channel, and any decoding rule. It may, however, take a long time to make the calculation. For instance, if we had used $N = 100$ (which is not unduly large for practical systems) and the channel were a BSC (and thus as simple as one can get), the number of terms in the sum on the right of (5.12) is $2^{100} \approx 10^{30}$ – more than a million times greater than Avogadro's number! Calculating P_B exactly appeals in general only to people whose industry far exceeds their imagination. The imaginative person will seek a far less tedious calculation that gives nearly as much information, i.e., he will seek simple bounds on P_B that are tight enough to satisfy his curiosity about the value of P_B.

5.4 Codes with Two Codewords – the Bhattacharyya Bound

Codes with only one codeword are of course uninteresting, but those with $M = 2$ codewords are worthy of considerable attention. From their properties, one can infer many important facts about codes with many codewords.

So that we may study the dependence of P_{B} on the probability distribution P_Z over the codewords, we introduce the notation

$$P_{\mathrm{B}|i} = P(\hat{Z} \neq Z | Z = i) \tag{5.15}$$

for the conditional block error probability given that the i-th codeword is transmitted. By the Theorem on Total Probability, we have

$$P_{\mathrm{B}} = \sum_{i=1}^{M} P_{\mathrm{B}|i} P_Z(i). \tag{5.16}$$

It follows from (5.16) that (for a given encoder, channel and decoder) the worst-case (over choices of the codeword probability distribution P_Z) block error probability as defined by (5.8) is just

$$(P_{\mathrm{B}})_{\mathrm{wc}} = \max_i P_{\mathrm{B}|i} \tag{5.17}$$

and occurs for that P_Z with $P_Z(i) = 1$ for an i that maximizes $P_{\mathrm{B}|i}$.

It is often convenient to specify the decoding function F by the "decoding regions" into which it partitions the set of received N-tuples for the channel in question. The i-th decoding region is just the set

$$\mathcal{D}_i = \{\underline{y} : \underline{y} \in B^N \text{ and } F(\underline{y}) = i\} \tag{5.18}$$

where B is the channel output alphabet. We can then write

$$P_{\mathrm{B}|i} = \sum_{\underline{y} \notin \mathcal{D}_i} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i). \tag{5.19}$$

Equations (5.15) - (5.19) apply to any block code and any channel. But an important simplification occurs in (5.19) when $M = 2$, namely $\underline{y}$ does not belong to the specified decoding region if and only if it does belong to the other. Thus, for instance, we have

$$P_{\mathrm{B}|2} = \sum_{\underline{y} \in \mathcal{D}_1} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2), \qquad M = 2. \tag{5.20}$$

Despite its apparent simplicity, (5.20) is often very difficult to evaluate for interesting decoders (such as an ML decoder) because of the difficulty in carrying out the sum only over those $\underline{y}$ in $\mathcal{D}_1$ rather than over all $\underline{y}$. To obviate this difficulty in the case of ML decoders, we first observe that

$$\underline{y} \in \mathcal{D}_1 \Rightarrow P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1) \geq P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2) \tag{5.21}$$

for ML decoding. In fact the inequality on the right would also imply that $\underline{y} \in \mathcal{D}_1$ except that, when it is satisfied with equality, we are free to place that $\underline{y}$ in either $\mathcal{D}_1$ or $\mathcal{D}_2$. But (5.21) also implies that

$$\sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1)} \geq \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}$$

and hence that

$$P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2) \leq \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)} \tag{5.22}$$

where here and hereafter the positive square root is understood. Using (5.22) in (5.20) gives

$$P_{\mathrm{B}|2} \leq \sum_{\underline{y} \in \mathcal{D}_1} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}. \tag{5.23}$$

An entirely similar argument would have given

$$P_{B|1} \leq \sum_{\underline{y} \in \mathcal{D}_2} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}. \tag{5.24}$$

and indeed it was to make the summand symmetrical in $\underline{x}_1$ and $\underline{x}_2$ that we introduced the square root in (5.22). Combining (5.23) and (5.24), we obtain

$$P_{B|1} + P_{B|2} \leq \sum_{\underline{y}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}. \tag{5.25}$$

and we note with satisfaction that the sum is now over *all* $\underline{y}$ and hence is considerably easier to evaluate or to bound.

For the special case of a DMC, we can use (4.3) in (5.25) to obtain

$$P_{B|1} + P_{B|2} \leq \sum_{y_1} \cdots \sum_{y_N} \sqrt{\prod_{n=1}^{N} P_{Y|X}(y_n|x_{1n}) P_{Y|X}(y_n|x_{2n})},$$

where we have again used the notation of (5.14). Because the square root of a product equals the product of the square roots, this simplifies to

$$P_{B|1} + P_{B|2} \leq \prod_{n=1}^{N} \sum_{y} \sqrt{P_{Y|X}(y|x_{1n}) P_{Y|X}(y|x_{2n})} \tag{5.26}$$

where we have merely written the dummy variable of summation as y rather than y_n. Because $P_{B|1}$ and $P_{B|2}$ are nonnegative, the right side of (5.26) is an upper bound on each of these conditional error probabilities. Thus, we have proved the so-called:

Bhattacharyya Bound: When the code (x_1, x_2) of length N is used with ML decoding on a DMC, then

$$P_{B|i} \leq \prod_{n=1}^{N} \sum_{y} \sqrt{P_{Y|X}(y|x_{1n}) P_{Y|X}(y|x_{2n})}, \qquad i = 1, 2 \tag{5.27}$$

or, equivalently,

$$(P_B)_{wc} \leq \prod_{n=1}^{N} \sum_{y} \sqrt{P_{Y|X}(y|x_{1n}) P_{Y|X}(y|x_{2n})}. \tag{5.28}$$

The Bhattacharyya Bound gives the first justification of our claim that an ML decoder, which is guaranteed to minimize P_B only when the codewords are equally likely, will also perform well regardless of the codeword probabilities, i.e., that it will be "nearly minimax".

Before turning to specific examples, we now argue that we can generally expect the bound (5.28) to be quite tight. First, we observe that an ML decoder is most likely to err when the received N-tuple is "near the boundary" of the decoding regions, i.e., when $\underline{Y} = \underline{y}$ where $\underline{y}$ gives near equality in the inequality of (5.21). But it is just for these $\underline{y}$ that (5.22), or equivalently (5.25), is quite tight. But going from (5.26) to (5.27) weakens the bound by at most a factor of two for the larger of $P_{B|1}$ and $P_{B|2}$ so that we can expect (5.28) also to be quite tight.

In the special case of a *binary-input* DMC and the length N repeat code, i.e., the code $([00\dots0],[11\dots1])$, the bound (5.27) becomes simply

$$P_{B|i} \le \left(\sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)}\right)^N, \qquad i=1,2. \tag{5.29}$$

Defining

$$D_B = -\log_2 \sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)}, \tag{5.30}$$

which we shall call the *Bhattacharyya* distance between the two channel input digits, we can rewrite (5.29) as

$$P_{B|i} \le 2^{-ND_B}, \qquad i=1,2 \tag{5.31}$$

or equivalently as

$$(P_B)_{wc} \le 2^{-ND_B}. \tag{5.32}$$

Next, we note that, by Cauchy's inequality (cf. Problem 5.7), the summation in (5.30) satisfies

$$\sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)} \le \sqrt{\sum_y P_{Y|X}(y|0)} \cdot \sqrt{\sum_y P_{Y|X}(y|1)} = 1$$

with equality if and only if

$$P_{Y|X}(y|0) = P_{Y|X}(y|1), \qquad \text{all } y. \tag{5.33}$$

Thus, it follows that

$$D_B \ge 0$$

with equality if and only if (5.33) holds. But (5.33) can be written as

$$P_{Y|X}(y|x) = P_Y(y) \qquad \text{all } x, y$$

which is just the condition that X and Y be statistically independent regardless of the choice of P_X, and this in turn is just the condition that the capacity of the DMC be zero. Thus, we have shown that the *Bhattacharyya distance D_B of a binary-input DMC is positive if and only if the capacity C is positive.* It follows from (5.32) that by using a repeat code on such a channel with $C > 0$, we can make P_B arbitrarily small by choosing N sufficiently large. This is not especially surprising since the rate

$$R = \frac{1}{N}$$

of the repeat code approaches 0 as N increases, but it is at least a demonstration that P_B can be made as small as desired for codes with more than one codeword.

Example 5.2: For the BEC of Figure 4.1(b), we find

$$\sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)} = \delta$$

so that

$$D_B = -\log_2 \delta. \qquad \text{(BEC)} \tag{5.34}$$

Thus, the bound (5.32) for ML decoding of the length N repeat code becomes

$$(P_B)_{wc} \le 2^{N\log_2\delta} = \delta^N. \qquad \text{(BEC)} \tag{5.35}$$

But in fact, assuming $\delta < 1$, an ML decoder will always decode correctly for this code and channel unless $[\Delta\Delta\ldots\Delta]$ is received. But the probability of receiving $[\Delta\Delta\ldots\Delta]$ is δ^N, independent of which codeword is transmitted. Assuming that the ML decoder chooses $\hat{Z} = 1$ when $[\Delta\Delta\ldots\Delta]$ is received, it follows that

$$P_{\mathrm{B}} = \delta^N P_Z(2)$$

so that

$$(P_{\mathrm{B}})_{\mathrm{wc}} = \delta^N,$$

which shows that the Bhattacharyya bound (5.35) is exact for this code and channel.

Example 5.3: For the BSC of Figure 4.1(a), we find

$$\sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)} = 2\sqrt{\varepsilon(1-\varepsilon)}$$

so that

$$D_{\mathrm{B}} = -\log_2[2\sqrt{\varepsilon(1-\varepsilon)}]. \qquad \text{(BSC)} \tag{5.36}$$

Thus, the bound (5.32) for ML decoding of the length N repeat code becomes

$$\begin{aligned}(P_{\mathrm{B}})_{\mathrm{wc}} &\le 2^{N\log_2[2\sqrt{\varepsilon(1-\varepsilon)}]} = \\ &= 2^N[\varepsilon(1-\varepsilon)]^{\frac{N}{2}}. \qquad \text{(BSC)}\end{aligned} \tag{5.37}$$

In this case, there is no simple exact expression for P_{B}. However, one can get a fair idea of the tightness of the Bhattacharyya bound (5.37) by considering $\varepsilon \ll 1$. The results for $1 \le N \le 8$ are as follows:

N	Bound (5.37)	Actual $(P_{\mathrm{B}})_{\mathrm{wc}}$ for ML decoding
1	$\approx 2\varepsilon^{\frac{1}{2}}$	ε
2	$\approx 4\varepsilon$	2ε
3	$\approx 8\varepsilon^{\frac{3}{2}}$	$\approx 3\varepsilon^2$
4	$\approx 16\varepsilon^2$	$\approx 6\varepsilon^2$
5	$\approx 32\varepsilon^{\frac{5}{2}}$	$\approx 10\varepsilon^3$
6	$\approx 64\varepsilon^3$	$\approx 20\varepsilon^3$
7	$\approx 128\varepsilon^{\frac{7}{2}}$	$\approx 35\varepsilon^4$
8	$\approx 256\varepsilon^4$	$\approx 70\varepsilon^4$

[All ML decoders give the same P_{B} when the codewords are equally likely, but they can have different $(P_{\mathrm{B}})_{\mathrm{wc}}$ depending on how ties are resolved. For N odd, the ML decoder is unique and also minimax (see Problem 5.2). For N even, there are many ML decoders; that ML decoder giving the largest $(P_{\mathrm{B}})_{\mathrm{wc}}$ (which is the one that chooses $F(\underline{y})$ to have the same value for all $\underline{y}$'s where there is a choice) was used to construct the above table, its $(P_{\mathrm{B}})_{\mathrm{wc}}$ is twice that of the ML decoder that gives the smallest $(P_{\mathrm{B}})_{\mathrm{wc}}$ (which is the one that chooses $F(\underline{y}) = 1$ for half of the $\underline{y}$ where there is a choice, and $F(\underline{y}) = 2$ for the other half) and that is also minimax. The Bhattacharyya bound holds for *all* ML decoders so its tightness should properly be determined by a comparison to the "worst" ML decoder, as we have done here.]

5.5 Codes with Many Codewords – the Union Bhattacharyya Bound and the Gallager Bound

We first consider a straightforward way of generalizing the Bhattacharyya bound (5.27) [but not the slightly stronger bound (5.26)] to a code $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$ with many codewords. We begin with (5.19), which we write as

$$P_{\mathrm{B}|i} = \sum_{\substack{j=1\\ j\neq i}}^{M} \sum_{\underline{y}\in\mathcal{D}_j} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i). \tag{5.38}$$

Next, we note that, for ML decoding,

$$\underline{y} \in \mathcal{D}_j \Rightarrow P(\underline{y}|\underline{x}_j) \geq P(\underline{y}|\underline{x}_i), \tag{5.39}$$

from which we see [by comparison to (5.21)] that every $\underline{y}$ in $\mathcal{D}_j$ can also be put into the decoding region $\mathcal{D'}_2$ of an ML decoder for the code $(\underline{x}'_1, \underline{x}'_2) = (\underline{x}_i, \underline{x}_j)$ with only two codewords. Thus, we see that

$$\sum_{\underline{y}\in\mathcal{D}_j} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) \leq \sum_{\underline{y}\in\mathcal{D'}_2} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) = P'_{B|1} \tag{5.40}$$

where $P'_{B|1}$ denotes the block error probability for this second ML decoder when $\underline{x}'_1 = \underline{x}_i$ is sent. We can now make use of (5.24) in (5.40) to conclude that

$$\begin{aligned} \sum_{\underline{y}\in\mathcal{D}_j} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) &\leq \sum_{\underline{y}\in\mathcal{D'}_2} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)} \\ &\leq \sum_{\underline{y}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)}. \end{aligned} \tag{5.41}$$

Substituting (5.41) into (5.38), we obtain the so-called:

Union Bhattacharyya Bound: When the code $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$ of length N is decoded by an ML decoder, then

$$P_{\mathrm{B}|i} \leq \sum_{\substack{j=1\\ j\neq i}}^{M} \sum_{\underline{y}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)} \tag{5.42}$$

for $i = 1, 2, \ldots, M$. In particular, for ML decoding on a DMC,

$$P_{\mathrm{B}|i} \leq \sum_{\substack{j=1\\ j\neq i}}^{M} \prod_{n=1}^{N} \sum_{y} \sqrt{P_{Y|X}(y|x_{in}) P_{Y|X}(y|x_{jn})}. \tag{5.43}$$

(The reason for the name "union Bhattacharyya bound" to describe (5.42) stems from an alternative derivation in which the event $\hat{Z} \neq i$ for an ML decoder is first written as a union of the events that $P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j) \geq P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)$ for $j \neq i$, then the probability of this union of these events given that $Z = i$ is overbounded by the sum of their probabilities given that $Z = i$.) When the number of codewords is large, one cannot expect the bound (5.42) [or (5.43)] to be very tight for the reason that the decoding region $\mathcal{D'}_2$ considered above is much larger (in probability as well as in the number of N-tuples that it contains) than the actual decoding region $\mathcal{D}_j$ – thus, the bound (5.40) will be quite loose in this case.

We now consider a less straightforward way to generalize the Bhattacharyya bound (5.27). Again our starting point is (5.19), which we write here again for ease of reference.

$$P_{\mathrm{B}|i} = \sum_{\underline{y}\notin\mathcal{D}_i} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i).$$

For an ML decoder, we know that

$$\underline{y} \notin \mathcal{D}_i \Rightarrow P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j) \geq P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i) \qquad \text{for some } j \neq i \tag{5.44}$$

but, unfortunately, we cannot easily characterize such j. But we can certainly be sure that

$$\underline{y} \notin \mathcal{D}_i \Rightarrow \left\{ \sum_{\substack{j=1 \\ j \neq i}}^{M} \left[\frac{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)}{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)} \right]^s \right\}^\rho \geq 1, \qquad \text{all } s \geq 0, \text{all } \rho \geq 0 \tag{5.45}$$

because (5.44) guarantees that at least one term in the sum will itself be at least one and then, since the sum is at least one, raising the sum to a nonnegative power ρ will also yield a number at least equal to 1. (For the moment we postpone a discussion of the reasons for including the free parameters s and ρ in (5.45).) We now note that (5.45) can be written as

$$\underline{y} \notin \mathcal{D}_i \Rightarrow P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)^{-s\rho} \left[\sum_{\substack{j=1 \\ j \neq i}}^{M} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)^s \right]^\rho \geq 1, \qquad \text{all } s \geq 0, \text{all } \rho \geq 0. \tag{5.46}$$

We next overbound the sum in (5.19) by multiplying each of its terms by the left side of the inequality in (5.46) to obtain

$$P_{\mathrm{B}|i} \leq \sum_{\underline{y} \notin \mathcal{D}_i} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)^{1-s\rho} \left[\sum_{\substack{j=1 \\ j \neq i}}^{M} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)^s \right]^\rho \qquad \text{all } s \geq 0, \text{all } \rho \geq 0. \tag{5.47}$$

As we are free to choose both s and ρ to be nonnegative numbers, nothing prevents us from making the choice

$$s = \frac{1}{1+\rho} \tag{5.48}$$

which at least introduces some "symmetry" into (5.47). Making this choice and further weakening the bound (5.47) by extending the summation to all $\underline{y}$, we obtain finally the:

Gallager Bound: When the code $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$ of length N is decoded by an ML decoder, then

$$P_{\mathrm{B}|i} \leq \sum_{\underline{y}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)^{\frac{1}{1+\rho}} \left[\sum_{\substack{j=1 \\ j \neq i}}^{M} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)^{\frac{1}{1+\rho}} \right]^\rho, \qquad \text{all } \rho \geq 0 \tag{5.49}$$

for $i = 1, 2, \ldots, M$. In particular, for ML decoding on a DMC, the Gallager bound (5.49) becomes

$$P_{\mathrm{B}|i} \leq \prod_{n=1}^{N} \sum_{y} P_{Y|X}(y|x_{in})^{\frac{1}{1+\rho}} \left[\sum_{\substack{j=1 \\ j \neq i}}^{M} P_{Y|X}(y|x_{jn})^{\frac{1}{1+\rho}} \right]^\rho, \qquad \text{all } \rho \geq 0. \tag{5.50}$$

We now consider the tightness of, and the rationale behind, the Gallager bound. First, we observe that choosing $\rho = 1$ in (5.49) [or (5.50)] yields the bound (5.42) [or (5.43)]. Thus, when optimized by choice of ρ, *the Gallager bound is strictly tighter than the union Bhattacharyya bound*, except when the choice $\rho = 1$ minimizes the right side of (5.49). This shows that the choice of s in (5.48) had some merit. In fact, as the reader may check by a tedious differentiation, this choice of s minimizes the sum over i of the right side of (5.47) when the second sum in (5.47) is slightly enlarged by the inclusion of the term with $j = i$; but in any case we were free to choose s subject only to the constraint that $s \geq 0$. Next, we consider the rationale for introducing the parameters s and ρ in (5.45). Because, for certain j, the ratio $P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_j)/P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)$ can be much larger than 1, it makes sense to introduce an s in the range $0 \leq s \leq 1$ (which the choice (5.48) ensures) to reduce the contributions from such terms. Then, because the sum in (5.45) might far exceed 1 when M is large, it also makes sense to raise this sum to the power ρ for some ρ in the range $0 \leq \rho \leq 1$ (in which range the optimizing value of ρ will usually lie – but not always because larger values of ρ may better decrease the contribution from those $\underline{y}$ in $\mathcal{D}_i$ that were included in weakening (5.47) to obtain (5.49).)

The tighter Gallager bound is not substantially more difficult to compute than the union Bhattacharyya bound. But the reader should not be deceived by that assertion. Unless the code and channel possess a high degree of simplifying symmetry, *both* bounds are too complicated to calculate in most practical cases. Their utility lies rather in the fact that both bounds can conveniently be "averaged", as we are about to do.

5.6 Random Coding – Codes with Two Codewords and the Cut-off Rate of a DMC

Shannon's 1948 paper was so full of innovative ideas that it is difficult to say which one was the most significant, but the "trick" of so-called "random coding" would be near the top in anyone's list of Shannon's major innovations. It is also one of the most widely misunderstood ideas in Shannon's work, although it is one of the simplest when properly understood.

Shannon recognized clearly what the reader who has been solving the problems in this chapter by now already suspects, namely that it is computationally infeasible to calculate, or even closely bound, the block error probability P_B for ML decoding of practically interesting codes with many codewords. Shannon's insight, however, rested in his realizing that bounding the *average* P_B for all codes of a given rate and length was a much simpler matter. In this section, we bound the average P_B for codes with two codewords, both because the "random coding trick" can most easily be understood in this setting and also because this special case will lead us to a measure of channel quality, the "cut-off rate", that is a close rival to channel capacity in its usefulness and significance.

The following "thought experiment" should make clear the nature of a random coding argument. Suppose, for a given channel and given N, that one has, after several thousand years of calculation, calculated $(P_B)_{wc}$ with ML decoding for every length N block code with two codewords. To make the dependence on the code explicit, we write $(P_B)_{wc}(\underline{x}_1, \underline{x}_2)$ to denote $(P_B)_{wc}$ for some ML decoder for the particular code $(\underline{x}_1, \underline{x}_2)$. Next, suppose that we store an encoder for each such code in a very large warehouse and stamp the encoder for the code $(\underline{x}_1, \underline{x}_2)$ with the number $(P_B)_{wc}(\underline{x}_1, \underline{x}_2)$. Now consider the random experiment of going into the warehouse and selecting an encoder "randomly" in such a way that the probability of selecting the encoder for the code $(\underline{x}_1, \underline{x}_2)$ is $Q_{\underline{X}_1\underline{X}_2}(\underline{x}_1, \underline{x}_2)$ – where here we use $Q_{\underline{X}_1\underline{X}_2}$ to denote the probability distribution for the code, rather than the usual $P_{\underline{X}_1\underline{X}_2}$, to assist in reminding us that we are considering a new random experiment, not the random experiment that we had considered when we calculated $P_{Bwc}(\underline{x}_1, \underline{x}_2)$ for a particular code $(\underline{x}_1, \underline{x}_2)$. For our new random experiment, we could now calculate the expected value of $(P_B)_{wc}$ as

$$\mathrm{E}[(P_B)_{wc}] = \sum_{\underline{x}_1}\sum_{\underline{x}_2}(P_B)_{wc}(\underline{x}_1, \underline{x}_2)Q_{\underline{X}_1\underline{X}_2}(\underline{x}_1, \underline{x}_2), \tag{5.51}$$

which is just the average of the number stamped on all the encoders in our warehouse. Having calculated $\mathrm{E}[(P_B)_{wc}] = \alpha$ say, we would be sure that at least one encoder in our warehouse was stamped with a value of $(P_\mathrm{B})_\mathrm{wc}$ at most equal to α. In fact, the probability of selecting an encoder stamped with $(P_\mathrm{B})_\mathrm{wc} \ge 5\alpha$ would be at most $\frac{1}{5}$. And this is all there is to so-called "random coding"!

We now carry out the averaging called for in (5.51). To simplify things, we specify

$$Q_{\underline{X}_1\underline{X}_2}(\underline{x}_1, \underline{x}_2) = Q_{\underline{X}}(\underline{x}_1)Q_{\underline{X}}(\underline{x}_2), \tag{5.52}$$

which says simply that we make the probability of choosing (the encoder for) the code $(\underline{x}_1, \underline{x}_2)$ equal to the probability of choosing $\underline{x}_1$ and $\underline{x}_2$ independently according to a common probability distribution $Q_{\underline{X}}$ (in other words, we make $\underline{X}_1$ and $\underline{X}_2$ i.i.d. random variables for our random experiment of code selection.) Equation (5.51) then becomes

$$\mathrm{E}[(P_\mathrm{B})_\mathrm{wc}] = \sum_{\underline{x}_1}\sum_{\underline{x}_2}(P_\mathrm{B})_\mathrm{wc}(\underline{x}_1, \underline{x}_2)Q_{\underline{X}}(\underline{x}_1)Q_{\underline{X}}(\underline{x}_2). \tag{5.53}$$

But the bound (5.25), because of (5.16), implies the Bhattacharyya bound

$$(P_\mathrm{B})_\mathrm{wc}(\underline{x}_1, \underline{x}_2) \le \sum_{\underline{y}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1)P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}. \tag{5.54}$$

Substituting (5.54) into (5.53), we obtain

$$\begin{aligned}\mathrm{E}[(P_\mathrm{B})_\mathrm{wc}] &\le \sum_{\underline{x}_1}\sum_{\underline{x}_2}\sum_{\underline{y}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1)P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}Q_{\underline{X}}(\underline{x}_1)Q_{\underline{X}}(\underline{x}_2) = \\ &= \sum_{\underline{y}}\sum_{\underline{x}_1} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_1)}Q_{\underline{X}}(\underline{x}_1) \sum_{\underline{x}_2} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_2)}Q_{\underline{X}}(\underline{x}_2).\end{aligned} \tag{5.55}$$

But we now recognize that $\underline{x}_1$ and $\underline{x}_2$ are just dummy variables of summation so that (5.55) reduces simply to

$$\mathrm{E}[(P_\mathrm{B})_\mathrm{wc}] \le \sum_{\underline{y}} \left[\sum_{\underline{x}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}Q(\underline{x})\right]^2 \tag{5.56}$$

where, in accordance with our usual convention for probability distributions, we have dropped the subscript $\underline{X}$ from $Q_{\underline{X}}$.

To specialize (5.56) to a DMC, we specify that the codeword probability distribution satisfy

$$Q_{\underline{X}}(x_1, x_2, \ldots, x_N) = \prod_{n=1}^{N} Q_X(x_n) \tag{5.57}$$

where Q_X is some probability distribution over the channel input alphabet, i.e., the components $X_1, X_2, \ldots, X_N$ of a codeword are also i.i.d. random variables in our new random experiment for code selection. Then, for a DMC, we have

$$\sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})}Q(\underline{x}) = \prod_{n=1}^{N} \sqrt{P_{Y|X}(y_n|x_n)}Q(x_n). \tag{5.58}$$

Substituting (5.58) into (5.56), we obtain

$$\begin{aligned}\mathrm{E}[(P_\mathrm{B})_\mathrm{wc}] &\le \sum_{y_1}\cdots\sum_{y_N}\left[\prod_{n=1}^{N}\sum_{x_n} \sqrt{P_{Y|X}(y_n|x_n)}Q(x_n)\right]^2 = \\ &= \prod_{n=1}^{N}\sum_{y_n}\left[\sum_{x_n} \sqrt{P_{Y|X}(y_n|x_n)}Q(x_n)\right]^2.\end{aligned} \tag{5.59}$$

Recognizing that x_n and y_n are now dummy variables of summation, we can rewrite (5.59) as

$$\mathrm{E}[(P_{\mathrm{B}})_{\mathrm{wc}}] \leq \left\{ \sum_y \left[\sum_x \sqrt{P_{Y|X}(y|x)} Q(x) \right]^2 \right\}^N \tag{5.60}$$

where we have dropped the subscripts from $P_{Y|X}$ in accordance with our convention for probability distributions. Rewriting (5.60) to emphasize the exponential dependence on the codeword length N, we obtain the:

Random Coding Bound for Codes with Two Codewords: Over the ensemble of all codes with two codewords of length N for use on a given DMC, wherein each code is assigned the probability that it would be selected when each digit of each codeword was chosen independently according to a given probability distribution Q_X over the channel input alphabet, the average of the worst-case (over assignments of probabilities for using the two codewords) block error probability, assuming an ML decoder for each code, satisfies

$$\mathrm{E}[(P_{\mathrm{B}})_{\mathrm{wc}}] \leq 2^{-N\left\{ -\log_2 \sum_y \left[\sum_x \sqrt{P(y|x)} Q(x) \right]^2 \right\}}. \tag{5.61}$$

In particular,

$$\mathrm{E}[(P_{\mathrm{B}})_{\mathrm{wc}}] \leq 2^{-NR_0} \tag{5.62}$$

where

$$R_0 = \max_Q \left\{ -\log_2 \sum_y \left[\sum_x \sqrt{P(y|x)} Q(x) \right]^2 \right\} \tag{5.63}$$

or, equivalently,

$$R_0 = -\log_2 \left\{ \min_Q \sum_y \left[\sum_x \sqrt{P(y|x)} Q(x) \right]^2 \right\}. \tag{5.64}$$

The quantity that we have denoted R_0 is called the *cut-off rate* of the DMC for reasons that we shall soon consider. For the moment, we will content ourselves with noting that

(i) R_0 depends only on the channel itself since the optimization over Q_X removes the dependence on the assignment of probabilities to the codes in the ensemble, and

(ii)

$$R_0 \geq 0 \tag{5.65}$$

with equality if and only if the capacity C is also 0.

The proof of (5.65) is left as an exercise (Problem 5.6) in which one further shows in fact that the exponent in wavy brackets in (5.61) is also nonnegative for every choice of Q_X.

The optimization over Q_X needed to find R_0 fortunately becomes extremely simple in the two cases of greatest practical interest, namely for binary-input channels and for all symmetric channels. For these cases, it is easy to show (see Problem 5.5) that

$$Q_X(0) = Q_X(1) = \frac{1}{2} \text{ achieves } R_0 \text{ when } A = \{0, 1\} \tag{5.66}$$

where A is the input alphabet for the DMC, and that

$$Q(x) = \frac{1}{L}, \text{ all } x \in A, \text{ achieves } R_0 \text{ when the DMC is symmetric.} \tag{5.67}$$

Making use of (5.66) in (5.64), we see that for any *binary-input* DMC (whether symmetric or not)

$$\begin{aligned} R_0 &= -\log_2 \sum_y \left[\frac{1}{2}\sqrt{P_{Y|X}(y|0)} + \frac{1}{2}\sqrt{P_{Y|X}(y|1)}\right]^2 \\ &= 1 - \log_2 \left[1 + \sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)}\right]. \end{aligned} \tag{5.68}$$

Making use of (5.30), we see that (5.68) can also be written

$$R_0 = 1 - \log_2[1 + 2^{-D_{\mathrm{B}}}] \tag{5.69}$$

where D_{B} is the Bhattacharyya distance between the input letters of the binary-input DMC. Equation (5.69) shows that R_0 and D_{B} uniquely determine one another for a binary-input DMC. Using (5.34) and (5.36) in (5.69), we find

$$R_0 = 1 - \log_2[1 + \delta] \qquad \text{(BEC)} \tag{5.70}$$

and

$$R_0 = 1 - \log_2 \left[1 + 2\sqrt{\varepsilon(1-\varepsilon)}\right], \qquad \text{(BSC)} \tag{5.71}$$

respectively.

Next, making use of (5.67) in (5.64), we find that, for any *symmetric* DMC,

$$\begin{aligned} R_0 &= -\log_2 \sum_y \left[\frac{1}{L}\sum_x \sqrt{P(y|x)}\right]^2 \\ &= \log_2 L - \log_2 \left[1 + \frac{1}{L}\sum_x \sum_{\substack{x' \\ x' \neq x}} \sum_y \sqrt{P_{Y|X}(y|x)P_{Y|X}(y|x')}\right] \end{aligned} \tag{5.72}$$

where L is the size of the channel input alphabet.

Some insight into the random coding bound can be gotten by considering the case of a *noiseless* BSC, i.e., a BSC with $\varepsilon = 0$. From (5.71), we see that

$$R_0 = 1$$

and hence (5.62) implies that

$$\mathrm{E}\left[(P_{\mathrm{B}})_{\mathrm{wc}}\right] \leq 2^{-N}. \tag{5.73}$$

This bound seems at first rather weak since, after all, the BSC is noiseless. However, the probability of those codes in our ensemble having two identical codewords is just

$$P(\underline{X}_1 = \underline{X}_2) = 2^{-N}$$

since, regardless of the value $\underline{x}$ of $\underline{X}_1$, we have probability 2^{-N} of choosing $\underline{X}_2$ also to be $\underline{x}$ because $Q_X(0) = Q_X(1) = \frac{1}{2}$. But, for each such code with two identical codewords, the decoder that always decides $\hat{Z} = 1$ is ML and has $(P_\mathrm{B})_\mathrm{wc} = 1$. All other codes give $P_\mathrm{B} \equiv 0$ with ML decoding on this channel. Thus, we see that if we choose these ML decoders

$$\mathrm{E}\left[(P_\mathrm{B})_\mathrm{wc}\right] = 2^{-N}$$

so that the random coding bound (5.73) is in fact exact. The point is that there are two contributions to $\mathrm{E}\left[(P_\mathrm{B})_\mathrm{wc}\right]$, namely one stemming from the chance that the channel noise will be "strong" and the other stemming from the chance that a "bad code" will be chosen.

5.7 Random Coding – Codes with Many Codewords and the Interpretation of Cut-off Rate

It is a simple matter to extend the random coding bound of the previous section to many codewords via the union Bhattacharyya bound. Suppose that our "warehouse" now contains an encoder for each possible block code $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$ of length N with $M = 2^{NR}$ codewords and that we have somehow assigned probabilities to these encoders. Taking expectations in (5.42), we then obtain

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq \sum_{\substack{j=1 \\ j\neq i}}^{M} \mathrm{E}\left[\sum_{\underline{y}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{X}_i) P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{X}_j)}\right] \tag{5.74}$$

for $i = 1, 2, \ldots, M$ where we have merely interchanged the order of averaging and summing. But we now recognize the expectation on the right of (5.74) as just the expectation of the Bhattacharyya bound (5.54) for two codewords. Thus, provided only that our assignment of probabilities to the encoders satisfies

$$Q_{\underline{X}_i \underline{X}_j}(\underline{x}_i, \underline{x}_j) = Q_{\underline{X}}(\underline{x}_i) Q_{\underline{X}}(\underline{x}_j), \text{ all } j \neq i \tag{5.75}$$

for all choices of $\underline{x}_i$ and $\underline{x}_j$, it follows from (5.54)-(5.56) that

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq (M-1) \sum_{\underline{y}} \left[\sum_{\underline{x}} \sqrt{P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})} Q_{\underline{X}}(\underline{x})\right]^2 \tag{5.76}$$

for $i = 1, 2, \ldots, M$.

The condition of (5.75) is the condition of *pairwise independence of the codewords* and states only that we must assign probabilities to our encoders in such a way that the total probability of all the encoders with a given pair of channel input vectors as the i-th and j-th codewords must equal the probability of choosing these two vectors independently according to the common probability distribution $Q_{\underline{X}}$. One conceptually simple way to get such pairwise independence is to make *all* the codewords independent, i.e., to assign probability

$$Q_{\underline{X}_1 \underline{X}_2 \cdots \underline{X}_M}(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M) = \prod_{m=1}^{M} Q_{\underline{X}}(\underline{x}_m) \tag{5.77}$$

to the encoder for $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$. However, we shall see later in our discussion of "linear codes" that we can still satisfy (5.75) for such a small subset of the possible encoders that it is no longer possible to satisfy (5.77). This is important because it tells us that this smaller (and easily implementable) set of encoders is just as good as the larger set in the sense that we can prove the same upper bound on average error probability for both sets of encoders. We can then confine our search for a good encoder to those in the smaller set.

For the special case of a DMC and the choice of $Q_{\underline{X}}$ as in (5.57), it follows from (5.56)-(5.60) and the trivial bound

$$M - 1 < M = 2^{NR} \tag{5.78}$$

that (5.76) becomes

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] < 2^{-N(R_0 - R)} \tag{5.79}$$

for $i = 1, 2, \ldots, M$ when we choose Q_X as the minimizing distribution in (5.64). Taking expectations in (5.16), we see that (5.79) implies the:

> *Random Coding Union Bound for Block Codes:* Over any ensemble of block codes with $M = 2^{NR}$ codewords of length N for use on a given DMC, wherein each code is assigned a probability in such a way that the codewords are pairwise independent and that for $i = 1, 2, \ldots, M$ each N-tuple of channel input letters appears as the i-th codeword in codes whose total probability is the same as the probability for its selection when each digit is chosen independently according to the R_0-achieving probability distribution Q_X, then, regardless of the particular probability distribution P_Z for the codewords, the average block error probability of these codes for ML decoding satisfies
>
> $$\mathrm{E}\left[P_{\mathrm{B}}\right] < 2^{-N(R_0 - R)}. \tag{5.80}$$

In (5.80), we have our first rigorously justified result showing that, by using sufficiently long block codes, we can make P_{B} arbitrarily small without decreasing the code rate provided only that the code rate is not too large. In fact, we see that for a fixed code rate R with $R < R_0$, we can choose codes for various values of N such that the resulting P_{B} decreases exponentially fast with blocklength N (or, more precisely, such that P_{B} is overbounded by an exponentially decreasing function of N). We now introduce the notation

$$E_{\mathrm{B}}(R) = R_0 - R \tag{5.81}$$

for the exponent in (5.80) when the code rate is R, and we shall call $E_{\mathrm{B}}(R)$ the *Bhattacharyya exponent* to remind us that it arose from the averaging of the union Bhattacharyya bound. This exponent is plotted in Figure 5.2 from which two important facts are readily apparent:

(i) $E_{\mathrm{B}}(R) > 0$ if and only if $R < R_0$. This shows that R_0 has the same dimensions as the code rate, R, and justifies the inclusion of "rate" in our terminology of "cut-off rate" for R_0. It also shows that

$$R_0 \le C \tag{5.82}$$

because otherwise we would have a contradiction to the Converse of the Noisy Coding Theorem.

(ii) $E_{\mathrm{B}}(R)$ decreases linearly with R from its maximum value of R_0 at $R = 0$. In fact we already saw in (5.62) that R_0 is the error exponent for codes with $M = 2$ codewords, i.e., for codes with rate $R = \frac{1}{N}$ bits/use.

This dual character of R_0 as both (i) the upper limit of a rate region for which we can guarantee that P_{B} can be made arbitrarily small by choice of N and (ii) a complete determiner of a positive error exponent for every rate R in this region, makes R_0 perhaps the most important single parameter for characterizing the quality of a DMC. Even channel capacity, C, is in a sense less informative as it plays only the first of these two roles.

There is one sense in which the random coding union bound (5.80) is still unsatisfactory. For each assignment of probabilities to the codewords, the average P_{B} for all codes is good, but this does not imply that there is any one code that gives a good P_{B} for all choices of P_Z. We now remedy this deficiency in the bound (5.80).

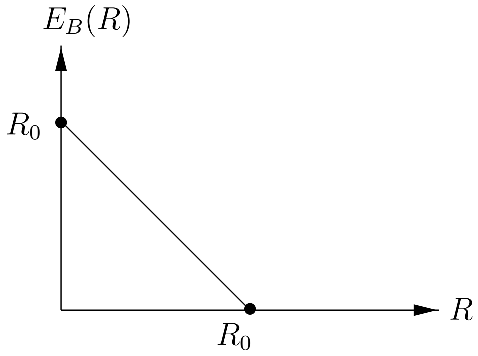


Figure 5.2: The Bhattacharyya Exponent $E_{\mathrm{B}}(R)$ of the Random Coding Union Bound.

We begin by considering the case of equally likely codewords (i.e. $P_Z(i) = 1/M$ for $i = 1, 2, \ldots, M$) so that P_{B} for each code is just the arithmetic mean of the conditional block error probabilities $P_{\mathrm{B}|i}, i = 1, 2, \ldots, M$. Next, we suppose that M is even, i.e., $M = 2M'$. Because at most half of the codewords can have $P_{\mathrm{B}|i}$ greater than or equal to twice the arithmetic mean P_{B} of these numbers, it follows that the $M' = M/2$ codewords with smallest $P_{\mathrm{B}|i}$ in each code must all correspond to i such that

$$P_{\mathrm{B}|i} \leq 2P_{\mathrm{B}}.$$

Hence, if we form a new code by discarding from each code the M' codewords for which $P_{\mathrm{B}|i}$ is greatest, then an ML decoder for this new code will have even larger decoding regions for its codewords than did the original code, and hence will give conditional error probabilities

$$P'_{B|i} \leq P_{\mathrm{B}|i} \leq 2P_{\mathrm{B}}$$

for $i = 1, 2, \ldots, M'$. This in turn implies that the worst-case (over assignments of probabilities to the M' codewords) block error probability for the new code satisfies

$$(P'_{\mathrm{B}})_{\mathrm{wc}} \leq 2P_{\mathrm{B}} \tag{5.83}$$

where P_{B} is still the block error probability for equally likely codewords in the original code. Assigning the same probability to the new code as to the original code, we see that (5.83) implies

$$\mathrm{E}\left[(P'_{\mathrm{B}})_{\mathrm{wc}}\right] < 2 \cdot 2^{-N(R_0 - R)}$$

where we have made use of (5.80) for the ensemble of original codes. But the rate of the new code is

$$R' = \frac{1}{N}\log_2 M' = \frac{1}{N}\log_2(\frac{M}{2}) = R - \frac{1}{N},$$

so the last inequality may be written

$$\mathrm{E}\left[(P'_{\mathrm{B}})_{\mathrm{wc}}\right] < 4 \cdot 2^{-N(R_0 - R')}$$

But there must be at least one new code whose worst-case error probability is no worse than average so that we have proved:

The Existence of Uniformly Good Codes: There exist block codes with $M = 2^{NR}$ codewords of length N for use on a given DMC such that the worst-case (over assignments of probabilities to the M codewords) block error probability with ML decoding satisfies

$$(P_{\mathrm{B}})_{\mathrm{wc}} < 4 \cdot 2^{-N(R_0 - R)}. \tag{5.84}$$

This result is one more verification of our claim that ML decoders (which minimize P_{B} for equally likely codewords) are "nearly minimax" (in the sense of nearly minimizing the worst-case P_{B}.)

5.8 Random Coding – Gallager's Version of the Coding Theorem for a Discrete Memoryless Channel

We saw in Section 5.5 that the Gallager bound (5.49) on $P_{\mathrm{B}|i}$, when optimized by choice of ρ, is strictly better than the union Bhattacharyya bound (5.42), except when $\rho = 1$ is the optimizing value, in which case the bounds coincide. Thus, averaging the Gallager bound over an ensemble of codes (as we are about to do) should in general yield a better bound on $\mathrm{E}\left[P_{\mathrm{B}|i}\right]$ than that which we obtained in the previous section by averaging the union Bhattacharyya bound.

As before, let $P_{\mathrm{B}|i}$ denote the block error probability of some ML decoder for the block code $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M)$ of length N used on a particular discrete channel, given that $Z = i$ so that $\underline{x}_i$ is the actual transmitted codeword. Rather than bounding $\mathrm{E}\left[P_{\mathrm{B}|i}\right]$ directly over the ensemble of codes, it turns out to be more convenient to begin by bounding the conditional expectation of $P_{\mathrm{B}|i}$ given that the i-th codeword is some particular vector $\underline{x}'$. The Gallager bound (5.49) gives immediately

$$\mathrm{E}\left[P_{\mathrm{B}|i}|\underline{X}_i = \underline{x}'\right] \leq \sum_{\underline{y}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}')^{\frac{1}{1+\rho}} \mathrm{E}\left[\left\{\sum_{\substack{j=1\\ j\neq i}}^{M} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{X}_j)^{\frac{1}{1+\rho}}\right\}^{\rho} \middle| \underline{X}_i = \underline{x}'\right] \tag{5.85}$$

for all $\rho \geq 0$. We see now the reason for conditioning on $\underline{X}_i = \underline{x}'$, namely so that the expectation operator could be moved inside the factor $P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}')$ in (5.85).

Next, we note that $f(\alpha) = \alpha^{\rho}$ is convex-$\cap$ in the interval $\alpha \geq 0$ when $0 \leq \rho \leq 1$, since then $f''(\alpha) = \rho(\rho-1)\alpha^{\rho-2} \leq 0$. Hence, Jensen's inequality implies that

$$\mathrm{E}\left[\{.\}^{\rho}|\underline{X}_i = \underline{x}'\right] \leq \{\mathrm{E}[.|\underline{X}_i = \underline{x}']\}^{\rho}, \qquad 0 \leq \rho \leq 1 \tag{5.86}$$

whenever the quantity at the places denoted by "." is nonnegative. We can thus use (5.86) in (5.85) to obtain

$$\mathrm{E}\left[P_{\mathrm{B}|i}|\underline{X}_i = \underline{x}'\right] \leq \sum_{\underline{y}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}')^{\frac{1}{1+\rho}} \left\{\sum_{\substack{j=1\\ j\neq i}}^{M} \mathrm{E}\left[P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{X}_j)^{\frac{1}{1+\rho}} \middle| \underline{X}_i = \underline{x}'\right]\right\}^{\rho}, \tag{5.87}$$

for $0 \leq \rho \leq 1$. To go further, we must specify our ensemble of codes. We choose the same ensemble as we did in the previous section; we assign probabilities to all possible encoders so that (5.75) is satisfied, i.e., so that the codewords enjoy pairwise independence. Now, because $\underline{X}_j$ is statistically independent of $\underline{X}_i$, the conditioning on $\underline{X}_i = \underline{x}'$ does not affect the expectation in (5.87). Thus,

$$\begin{aligned}\mathrm{E}\left[P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{X}_j)^{\frac{1}{1+\rho}} \middle| \underline{X}_i = \underline{x}'\right] &= \mathrm{E}\left[P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{X}_j)^{\frac{1}{1+\rho}}\right] \\ &= \sum_{\underline{x}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})^{\frac{1}{1+\rho}} Q_{\underline{X}}(\underline{x})\end{aligned} \tag{5.88}$$

Substituting (5.88) into (5.87), we find

$$\mathrm{E}\left[P_{\mathrm{B}|i}|\underline{X}_i = \underline{x}'\right] \leq \sum_{\underline{y}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}')^{\frac{1}{1+\rho}} (M-1)^{\rho} \left\{\sum_{\underline{x}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})^{\frac{1}{1+\rho}} Q_{\underline{X}}(\underline{x})\right\}^{\rho}, \tag{5.89}$$

for $0 \leq \rho \leq 1$. Finally, we make use of the theorem on total expectation to write

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] = \sum_{\underline{x}'} \mathrm{E}\left[P_{\mathrm{B}|i} | \underline{X} = \underline{x}'\right] Q_{\underline{X}}(\underline{x}'). \tag{5.90}$$

Using (5.89) in (5.90) now gives

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq \sum_{\underline{y}} \sum_{\underline{x}'} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}')^{\frac{1}{1+\rho}} Q_{\underline{X}}(\underline{x}')(M-1)^{\rho} \left\{ \sum_{\underline{x}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})^{\frac{1}{1+\rho}} Q_{\underline{X}}(\underline{x}) \right\}^{\rho},$$

for $0 \leq \rho \leq 1$. We then observe that, because $\underline{x}$ and $\underline{x}'$ are only dummy variables ranging over the set of possible codewords, the summations over $\underline{x}'$ and $\underline{x}$ in this last inequality are identical. Thus, we have shown that

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq (M-1)^{\rho} \sum_{\underline{y}} \left[\sum_{\underline{x}} P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x})^{\frac{1}{1+\rho}} Q_{\underline{X}}(\underline{x}) \right]^{1+\rho}, \qquad 0 \leq \rho \leq 1. \tag{5.91}$$

Inequality (5.91) is Gallager's celebrated random coding bound and applies for ML decoding of any block code on any discrete channel. For the special case of a DMC and the choice (5.57) for $Q_{\underline{X}}$, (5.91) reduces to

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq (M-1)^{\rho} \left\{ \sum_{y} \left[\sum_{x} P_{Y|X}(y|x)^{\frac{1}{1+\rho}} Q_X(x) \right]^{1+\rho} \right\}^{N}, \qquad 0 \leq \rho \leq 1. \tag{5.92}$$

If we now drop subscripts according to our usual convention and define *Gallager's function* by

$$E_0(\rho, Q) = -\log_2 \sum_{y} \left[\sum_{x} P(y|x)^{\frac{1}{1+\rho}} Q(x) \right]^{1+\rho}, \tag{5.93}$$

then we can rewrite (5.92) as

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq (M-1)^{\rho} 2^{-NE_0(\rho,Q)}, \qquad 0 \leq \rho \leq 1, \tag{5.94}$$

a form that we shall later find convenient in our study of tree and trellis codes. To make (5.94) even more suggestive, we recall the definition (5.1) of code rate and make use of the trivial inequality

$$(M-1)^{\rho} \leq M^{\rho} = 2^{\rho NR} \tag{5.95}$$

which we then use in (5.94) to obtain the fundamental inequality

$$\mathrm{E}\left[P_{\mathrm{B}|i}\right] \leq 2^{-N[E_0(\rho,Q)-\rho R]}, \qquad 0 \leq \rho \leq 1. \tag{5.96}$$

which holds for $i = 1, 2, \ldots, M$. We can now use (5.96) in (5.16) to establish the following result.

Gallager's Random Coding Bound for Block Codes: Over any ensemble of block codes with $M = 2^{NR}$ codewords of length N for use on a given DMC, wherein each code is assigned a probability in such a way that the codewords are pairwise independent and that the digits in the i-th word are statistically independent and each is distributed according to a given probability distribution Q over the channel input alphabet, then, regardless of the particular probability distribution P_Z for the codewords, the average block error probability of these codes for ML decoding satisfies

$$\mathrm{E}[P_{\mathrm{B}}] \leq 2^{-N[E_0(\rho,Q)-\rho R]} \tag{5.97}$$

for all ρ such that $0 \leq \rho \leq 1$. In particular,

$$\mathrm{E}[P_{\mathrm{B}}] \leq 2^{-NE_{\mathrm{G}}(R)} \tag{5.98}$$

where

$$E_{\mathrm{G}}(R) = \max_{Q} \max_{0\leq\rho\leq 1} [E_0(\rho, Q) - \rho R]. \tag{5.99}$$

The quantity $E_{\mathrm{G}}(R)$, which will be called the Gallager exponent in honor of its discoverer, has many interesting and important properties. Not unexpectedly, we can see easily that

$$E_{\mathrm{G}}(R) \geq E_{\mathrm{B}}(R) = R_0 - R \tag{5.100}$$

with equality if and only if $\rho = 1$ is maximizing in (5.99). This follows from the fact that, as a comparison of (5.93) and (5.63) shows,

$$\max_{Q} E_0(1, Q) = R_0 \tag{5.101}$$

It is similarly unsurprising in light of (5.67) that

$$Q(x) = \frac{1}{L}, \qquad \text{all } x \in A,$$

when the channel is symmetric, maximizes $E_0(\rho, Q)$ for all $\rho \geq 0$; but the reader is cautioned that, in spite of (5.66), the choice $Q_x(0) = Q_x(1) = \frac{1}{2}$ does *not* in general maximize $E_0(\rho, Q)$ for ρ such that $0 \leq \rho < 1$ for an unsymmetric binary-input channel. Other interesting properties of $E_{\mathrm{G}}(R)$ can be seen conveniently by writing

$$E_{\mathrm{G}}(R) = \max_{Q} E_{\mathrm{G}}(R, Q) \tag{5.102}$$

where

$$E_{\mathrm{G}}(R, Q) = \max_{0\leq\rho\leq 1} [E_0(\rho, Q) - \rho R]. \tag{5.103}$$

It is not particularly difficult to show that $E_{\mathrm{G}}(R, Q)$ has the form shown in Figure 5.3 as a function of R, that is:

(i) The $R = 0$ intercept is $E_0(1, Q)$, i.e.,

$$E_{\mathrm{G}}(0, Q) = E_0(1, Q).$$

(ii) $E_{\mathrm{G}}(R, Q)$ is linear with slope -1 in the interval $0 \leq R \leq R_C(Q)$, where

$$R_C(Q) = \left.\frac{\partial E_0(\rho, Q)}{\partial \rho}\right|_{\rho=1}. \tag{5.104}$$

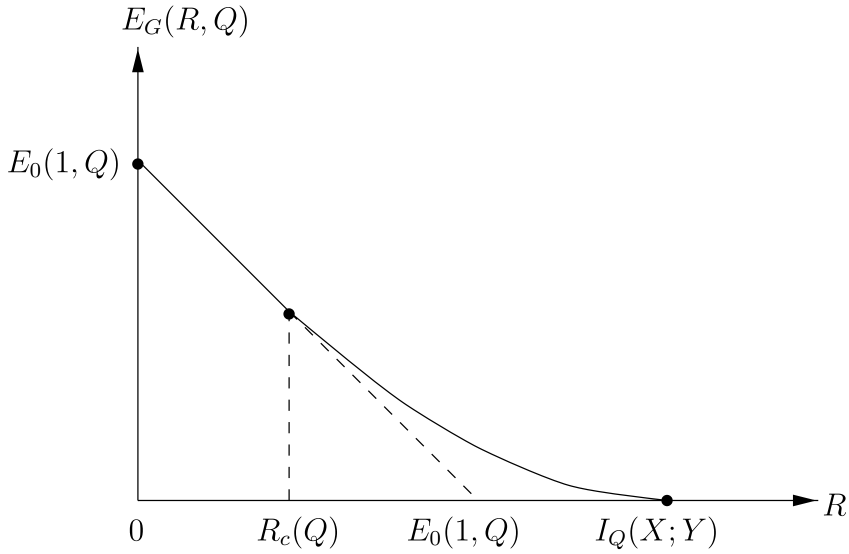


Figure 5.3: The general form of the exponent $E_G(R,Q)$ as a function of R.

(iii) $E_G(R,Q)$ is convex-∪ and positive in the interval $0 \le R \le I_Q(X;Y)$, where, by definition,

$$I_Q(X;Y) = I(X;Y)|_{P_X=Q} \tag{5.105}$$

is the average mutual information between the input X and output Y of the DMC when the input probability distribution is chosen as $P_X(x) = Q(x)$ for all $x \in A$.

It follows immediately from (5.102) that $E_G(R)$ is just the upper envelope of the curves $E_G(R,Q)$ taken over all Q. It is thus evident from Figure 5.3 that $E_G(R)$ has the form shown in Figure 5.4, that is:

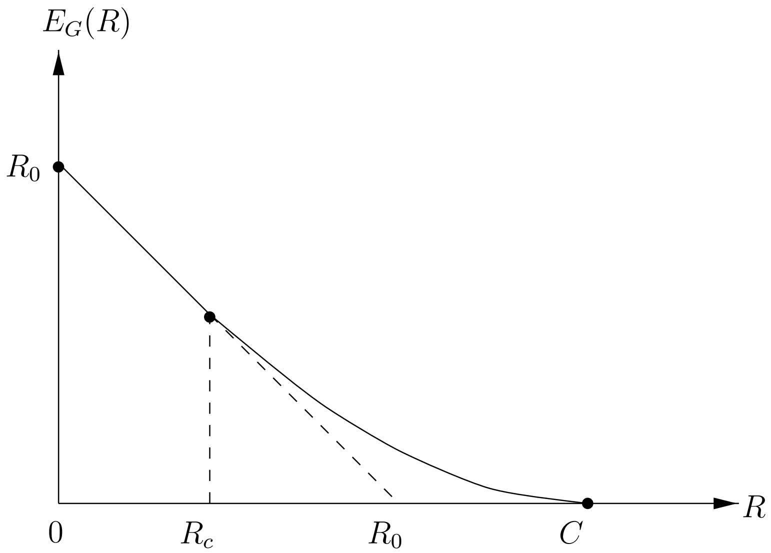


Figure 5.4: The general form of the Gallager exponent $E_G(R)$ for a DMC.

(i) The $R = 0$ intercept is R_0, i.e.,

$$E_G(0) = \max_Q E_0(1,Q) = R_0 \tag{5.106}$$

(ii) $E_{\mathrm{G}}(R)$ is linear with slope -1 in the interval $0 \le R \le R_{\mathrm{c}}$ where

$$R_{\mathrm{c}} = \max_{Q(R_0\text{-achieving})} \left. \frac{\partial E_0(\rho, Q)}{\partial \rho} \right|_{\rho=1} \tag{5.107}$$

where the maximum is taken only over those channel input probability distributions that achieve R_0, i.e., which maximize $E_0(1, Q)$. (It is the fashion to call R_{c} the "critical rate" and to denote it by "R_{crit}". However, it is "critical" only to designers of bounds on error probability, not to designers of communications systems, so we have shunned this terminology and will leave R_{c} nameless.)

(iii) $E_{\mathrm{G}}(R)$ is convex-$\cup$ and positive in the interval $0 \le R \le C$ where C is the capacity of the DMC.

The fact that $E_{\mathrm{G}}(R) > 0$ for $R < C$ not only establishes the coding theorem for a DMC as stated in Section 4.5, but also gives us the additional information that, for any fixed code rate R less than capacity C, we can make P_{B} go to zero exponentially fast in the block length N by the choice of an appropriate code, i.e., one for which P_{B} is no more than $\mathrm{E}[P_{\mathrm{B}}]$. Moreover, the same argument by which we proved the existence of "uniformly good" block codes in the previous section (i.e., codes for which $(P_{\mathrm{B}})_{\mathrm{wc}}$ is no more than 4 times $\mathrm{E}[P_{\mathrm{B}}]$ for equally likely codewords), can now be applied to Gallager's random coding bound for block codes and gives the following strong result.

Gallager's Coding Theorem for a DMC: For any code rate R and blocklength N, there exist block codes with $M = 2^{NR}$ codewords of length N for use on a given DMC such that the worst-case (over assignments of probabilities to the M codewords) block error probability with ML decoding satisfies

$$(P_{\mathrm{B}})_{\mathrm{wc}} < 4 \cdot 2^{-NE_{\mathrm{G}}(R)} \tag{5.108}$$

where $E_{\mathrm{G}}(R)$ is defined by (5.99) and satisfies

$$E_{\mathrm{G}}(R) > 0 \qquad \text{for} \qquad 0 \le R < C,$$

where C is the capacity of the DMC.

This is perhaps the best point to remind the reader that, as was mentioned in the previous section, channel capacity C in spite of its fundamental role is in some sense a less informative *single parameter* for characterizing a DMC than is the cut-off rate R_0. The reason is that knowledge of C alone tells us nothing about the magnitude of $E_{\mathrm{G}}(R)$ for some particular $R < C$ (whereas R_0 specifies $E_{\mathrm{B}}(R) = R_0 - R$ for all $R < R_0$). The problem with C alone is that both situations shown in Figure 5.5 are possible, namely

$$\begin{aligned} R_0 &= C \qquad \text{and} \\ R_0 &\ll C, \end{aligned}$$

as the reader can verify by solving Problem 5.11.

Finally, we should mention that, although block codes are easy to visualize and fit well into certain data formats commonly used in digital data communications, there is nothing fundamental that requires us to use block codes on noisy channels. As we shall see in the next chapter, there are certain real advantages in using "non-block" codes. Our efforts in this chapter, however, have hardly been wasted because we will make use of our error bounds for block codes in our derivation of the corresponding bounds for non-block codes.

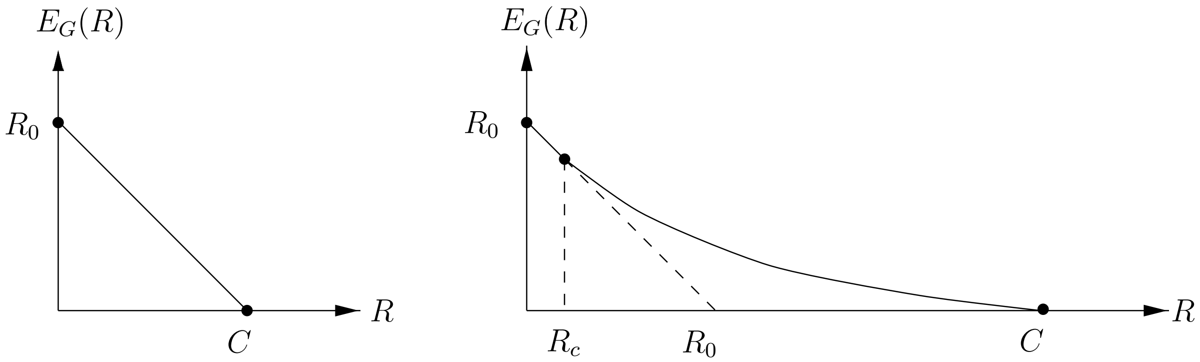


Figure 5.5: The two extreme forms of Gallager's error exponent for discrete memoryless channels.

Chapter 6

TREE AND TRELLIS CODING PRINCIPLES

6.1 Introduction

In this chapter, we consider two "non-block" coding techniques for noisy channels, namely tree coding and trellis coding. In one sense, each of these techniques is a generalization of block coding. But in another sense (and the one we will pursue), tree coding and trellis coding can both be viewed as special cases of block coding. While there is no clear analytical distinction between tree and trellis codes on the one hand and block codes on the other, there is an enormous difference in the design considerations that underlie these coding techniques. In many applications, tree codes and trellis codes have been found much superior to "traditional" block codes. As we shall see in this chapter, these "non-block" codes are superior in certain theoretical respects as well. We shall return to this comparison after we have developed a better understanding of "non-block" coding techniques.

6.2 A Comprehensive Example

Rather than beginning our discussion of "non-block" codes with precise definitions and a general analysis, we prefer to begin with a simple example that nevertheless embodies all the main features of non-block coding. Our starting point is the binary "convolutional encoder" of Figure 6.1. In this figure, the information symbols (U_i) and the encoded symbols (X_i) are all binary digits. We can of course use this encoder on any binary-input channel. The adders shown in Figure 6.1 are "modulo-two adders", i.e., their output is 1 if and only if an odd number of the inputs have value 1. During each "time-unit", one information bit enters the encoder and two channel input digits are emitted. The rate of this coding device is thus given by

$$R_{\mathrm{t}} = \frac{1}{2} \text{ bit/use} \tag{6.1}$$

where the subscript denotes "tree" or "trellis" – as we shall soon see, this same device can serve as an encoder for either kind of non-block code. The contents of the unit-delay cells as shown in Figure 6.1 are their current outputs, i.e., their inputs one time unit earlier. The "serializer" in Figure 6.1 merely interleaves its two input sequences to form the single output sequence $X_1, X_2, X_3, \ldots$ that is to be transmitted over the channel.

The input sequences to the serializer in Figure 6.1 can be written

$$X_{2i-1} = U_i \oplus U_{i-2}, \qquad i = 1, 2, \ldots \tag{6.2}$$

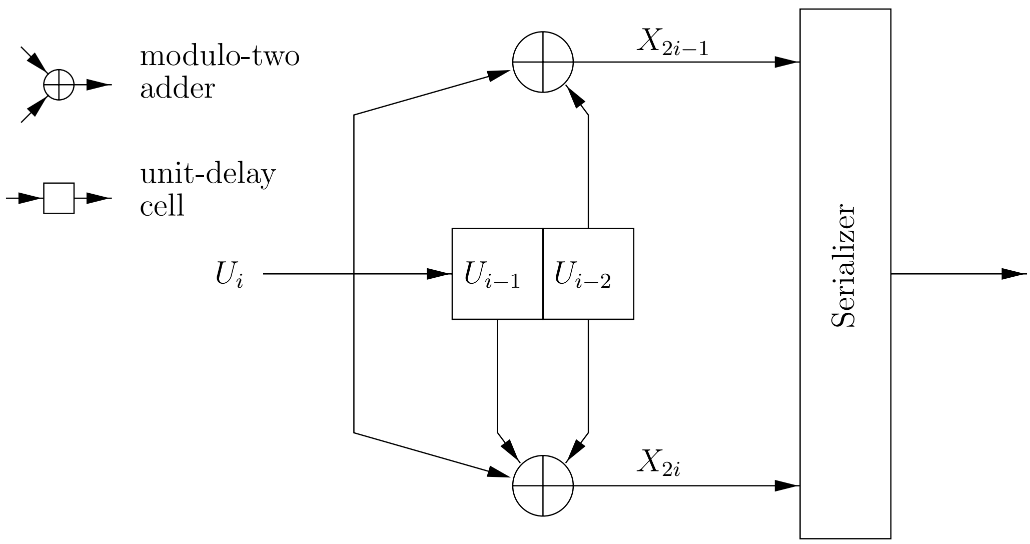


Figure 6.1: An $R_t = 1/2$ Convolutional Encoder.

and

$$X_{2i} = U_i \oplus U_{i-1} \oplus U_{i-2}, \qquad i = 1, 2, \ldots \tag{6.3}$$

where $\oplus$ denotes modulo-two addition. The right sides of (6.2) and (6.3) can be viewed as a "convolution" of the input sequence $U_1, U_2, U_3, \ldots$ with the sequences $1, 0, 1, 0, 0, \ldots$ and $1, 1, 1, 0, 0, \ldots$, respectively, which explains the nomenclature "convolutional encoder". We shall later have much more to say about the mathematical structure of such encoders.

The "state" of a system is a description of its past history which, together with specification of the present and future inputs, suffices to determine the present and future outputs. Thus, for the encoder of Figure 6.1, we can choose the *state* at time i to be the current contents of the delay cells, i.e.,

$$\sigma_i = [U_{i-1}, U_{i-2}]. \tag{6.4}$$

(In fact we can always choose the state to be the current contents of the delay cells in any *sequential circuit*, i.e., in any device built up from unit-delay cells and memoryless logical operations.) We have tacitly been assuming that the *initial state* of the device in Figure 6.1 was "zero", i.e., that

$$\sigma_1 = [0, 0] \tag{6.5}$$

or, equivalently, that

$$U_{-1} = U_0 = 0. \tag{6.6}$$

Thus, U_{-1} and U_0 are "dummy information bits" since their values are not free to be determined by the user of the coding system.

A. A Tree Code

Suppose that we decide to use the encoder of Figure 6.1 to encode true information bits for L_t time units, after which we encode "dummy information bits" (all of which are zero) during a tail of T time units. Choosing for our example $L_t = 3$ and $T = 2$, we would then have only 3 true information bits (U_1, U_2 and U_3) and we would have specified

$$U_4 = U_5 = 0. \tag{6.7}$$

Suppose further that we take the encoder output sequence over these $L_t + T = 5$ time units, namely

$$\underline{X} = [X_1, X_2, \dots, X_{10}], \tag{6.8}$$

as the codeword in what is now a block code with $K = 3$ information bits and codeword length $N = 10$. The fact that we have chosen to encode true information bits for only L_t out of the $L_t + T$ time units in which the codeword is formed has caused the rate R of the resulting block code to be related to the *nominal rate* R_t of the tree (or trellis) encoder by

$$R = \frac{L_t}{L_t + T} R_t. \tag{6.9}$$

For our example, we have

$$R = \frac{3}{5} \cdot \frac{1}{2} = \frac{3}{10} = \frac{K}{N} \text{ bits/use.} \tag{6.10}$$

In a practical tree (or trellis) coding system, one generally has $L_t \gg T$ (in fact, $L_t = \infty$ is not unusual for trellis coding, as we shall see later) so that R and R_t are virtually identical, unlike the case in our simple example here.

We now come to the heart of the matter, namely the special structure of the codewords in our block code. First, we note that X_1 and X_2 depend only on U_1. Next, we note that X_3 and X_4 depend only on U_1 and U_2. Finally, we note that $X_5, X_6, \dots, X_{10}$ are of course determined by the values of all three information bits U_1, U_2 and U_3. Thus, we can show the $2^3 = 8$ codewords in this block code by means of the binary rooted tree in Figure 6.2. Starting at the root node, we leave by the upper branch if $U_1 = 1$ but by the lower branch if $U_1 = 0$. These two branches are labelled with the corresponding values of X_1 and X_2. Arriving at the node at depth 1 determined by U_1, we leave again by the upper branch if $U_2 = 1$ but by the lower branch if $U_2 = 0$. These branches are labelled with the corresponding values of X_3 and X_4. Similar comments apply when we arrive at the nodes at depth 2 determined by U_1 and U_2. But when we arrive at the node at depth 3 determined by U_1, U_2 and U_3, the values of X_7 and X_8 are uniquely determined since we are now in the tail of the tree where we have demanded that $U_4 = 0$. Likewise, there is only one branch leaving the nodes at depth 5 since we have also demanded that $U_5 = 0$.

The fact that the $2^K = 8$ codewords in our block code can be placed on a rooted tree in the manner shown in Figure 6.2 is, of course, why we call this code a *tree code*. We shall see later how this special structure of the codewords can be used to simplify decoding.

B. A Trellis Code

We have not really discussed how we found the appropriate encoded digits to place on the branches of the rooted tree in Figure 6.2. We could of course have simply considered inserting each choice of the information sequence $[U_1, U_2, U_3]$ (with $U_{-1} = U_0 = U_4 = U_5 = 0$) into the encoder of Figure 6.1 and calculated the resulting codeword $[X_1, X_2, \dots, X_{10}]$. If we were trying to save ourselves effort, we would soon discover that it paid to remember the state of the encoder at each step so that after having found the codeword for, say, $[U_1, U_2, U_3] = [1, 1, 0]$, we would not have to start from the beginning to find the codeword for $[U_1, U_2, U_3] = [1, 1, 1]$.

In Figure 6.3, we have redrawn the tree of Figure 6.2, enlarging the nodes to show the state of the encoder at the time when the corresponding information bit is applied. Notice from (6.4) that this state depends only on the information sequence (and not at all on the encoded digits on the branches leaving the node.) Thus, the state labels for the nodes in Figure 6.3 can be inserted before we begin to worry about what encoded digits to put on each branch. To simplify calculation of these encoded digits, it is helpful to consider the *state-transition diagram* of the encoder as shown in Figure 6.4. The nodes in this diagram correspond to the states $[U_{i-1}, U_{i-2}]$ of the encoder. The branches are labelled with the encoder input U_i that causes this transition and with the resulting encoder output $[X_{2i-1}, X_{2i}]$ in the manner

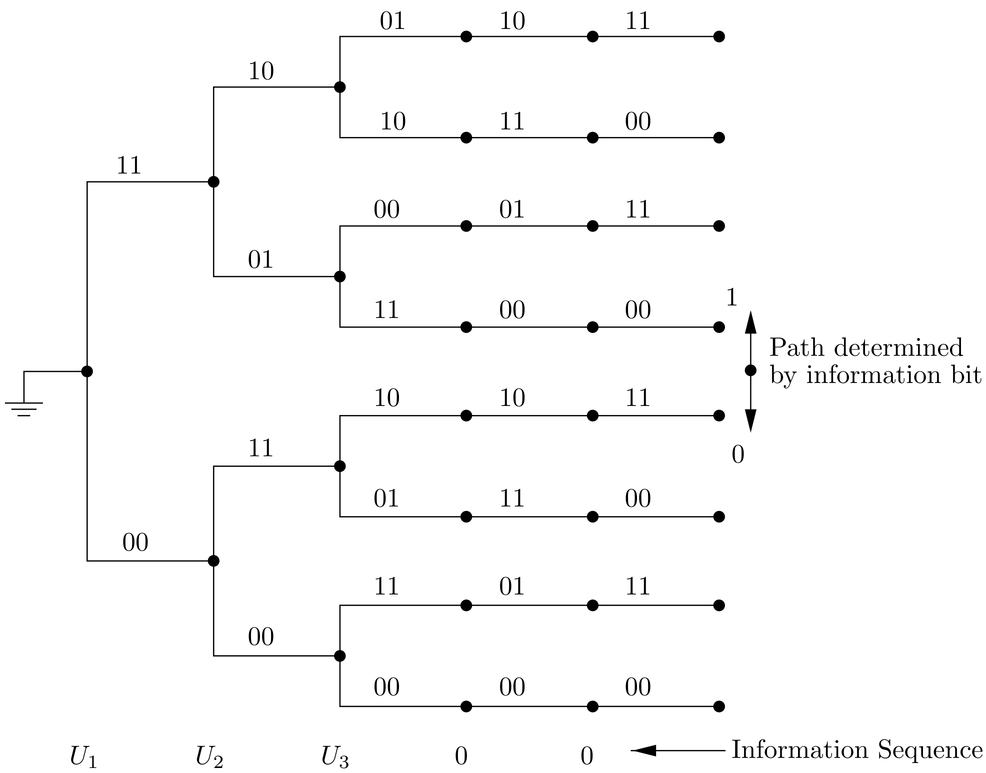


Figure 6.2: The $L = 3$, $T = 2$ Tree Code Encoded by the Convolutional Encoder of Figure 6.1.

$U_i/[X_{2i-1}, X_{2i}]$. From this diagram, it is a simple matter to read off the appropriate encoded digits to place on each branch of the tree in Figure 6.3.

Having now drawn our tree as in Figure 6.3 to show the encoder states, we notice that at depth 3, each of the 4 possible states appears twice. But since the present and future output of the encoder depends only on the present state and the present and future input, we can treat these two appearances of each state as the same node in a new diagram for showing the encoder output sequences (i.e., the codewords). We can likewise reduce each distinct state to one appearance at depths 4 and 5 in the tree. The result of our insistence that each state appear at most once at each depth is the diagram of Figure 6.5. The new diagram is, of course, not a tree, but rather a structure that has come to be called a *trellis*. In the tree, branches had a "natural direction" away from the root node. But we need to show this direction explicitly by arrows on the branches of the trellis – or by agreeing that we shall always draw our trellises (as in Figure 6.5) so that the direction is always from left to right. Notice that all the paths through the trellis end at the same node, which we shall call the *toor node* (toor = root spelled backwards) and which we indicate on the trellis by an upside-down ground symbol.

We shall call a code that can be placed on a trellis in the manner shown in Figure 6.5 a *trellis code*. The codewords are the sequences of digits on the paths from the root to the toor node. It is important to notice that the trellis code of Figure 6.5 is *exactly the same block code* as the tree code of Figure 6.2. The $2^K = 8$ codewords are exactly the same, and the same information sequence $[U_1, U_2, U_3]$ will give rise to the same codeword $[X_1, X_2, \ldots, X_{10}]$ in both codes. It just so happens that the codewords in the tree code of Figure 6.2 possess the special structure that permitted them to be displayed as the trellis code of Figure 6.5. This would not have been the case in general if we had formed the tree code by placing arbitrarily chosen binary digits on the tree in Figure 6.3 (see Problem 6.1). Thus, only some tree codes are also trellis codes. Which ones? The alert reader will have noticed that the tail of $T = 2$ dummy information bits that was used with the encoder of Figure 6.1 just sufficed to drive the encoder back to the zero state after the codeword had been formed – this is why the paths in the tree of Figure 6.3 all

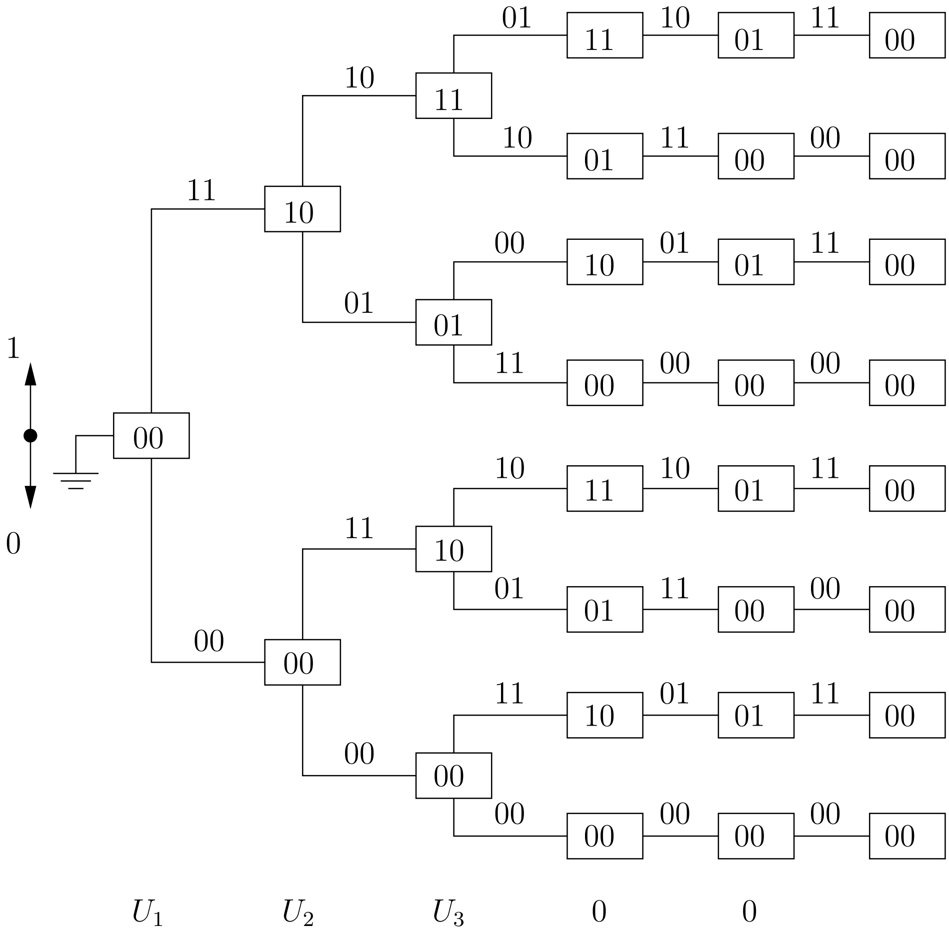


Figure 6.3: The $L_t = 3$, $T = 2$ Tree Code of Figure 6.2 Redrawn to Show the Encoder States at each Node.

ended in the zero state, and this in turn is why we were able to put this particular tree code on a trellis diagram.

We have already mentioned that the tree structure of the codewords in a tree code can be exploited in decoding. The reader is not wrong if he suspects that the further trellis structure can likewise be exploited in decoding.

C. Viterbi (ML) Decoding of the Trellis Code

The trellis code of Figure 6.5 can of course be used on any binary-input channel. Suppose then that we use this code on a BSC with crossover probability ε, where $0 < \varepsilon < \frac{1}{2}$, and that we wish to use ML decoding. From Example 4.1, we recall that for the BSC

$$P(\underline{y}|\underline{x}) = (1-\varepsilon)^N \left(\frac{\varepsilon}{1-\varepsilon}\right)^{d(\underline{x},\underline{y})}, \tag{6.11}$$

where $d(\underline{x}, \underline{y})$ is the Hamming distance between $\underline{x}$ and $\underline{y}$. But

$$0 < \varepsilon < \frac{1}{2} \qquad \Leftrightarrow \qquad 0 < \frac{\varepsilon}{1-\varepsilon} < 1. \tag{6.12}$$

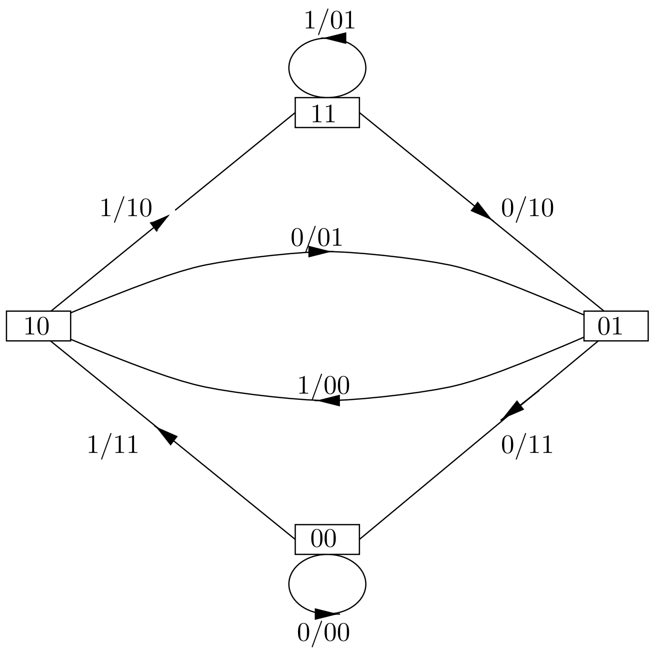


Figure 6.4: The State-Transition Diagram for the Convolutional Encoder of Figure 6.1.

It follows from (6.11) and (6.12) that choosing i to maximize $P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}_i)$ is equivalent to choosing i to minimize $d(\underline{x}, \underline{y})$. Thus, we have the following general result:

> *ML Decoding Rule for a Block Code on a BSC with* $0 < \varepsilon < \frac{1}{2}$*:* For each received N-tuple $\underline{y}$, choose $F(\underline{y}) = i$ where i is (one of) the index(es) that minimizes $d(\underline{x}_i, \underline{y})$ or, equivalently, that maximizes the number of positions in which $\underline{x}_i$ and $\underline{y}$ agree.

For our trellis code (which is also a block code of length $N = 10$), we can perform ML decoding by finding, when $\underline{Y} = \underline{y}$ is received, that path through the trellis of Figure 6.5 which agrees with $\underline{y} = [y_1, y_2, \ldots, y_{10}]$ in the most positions, then taking our decisions $[\hat{U}_1, \hat{U}_2, \hat{U}_3]$ as the corresponding information sequence for that path. (We have now tacitly assumed that our decoder should decide directly on the information sequence, rather than only on the index Z of the codeword as we considered previously.) A specific case, namely

$$\underline{y} = [0101010111] \tag{6.13}$$

will be used to illustrate how ML decoding can easily be implemented.

Our approach to ML decoding will be to move through the trellis of Figure 6.5, "remembering" all the subpaths that might turn out to be the prefix of the best path through the trellis, i.e., the one that agrees with $\underline{y}$ in the maximum number of positions. Our decoding metric, i.e., our measure of goodness of any subpath, is just the number of positions in which that subpath agrees with the received sequence.

In Figure 6.6, we show how ML decoding can be performed on the received sequence of (6.13). We begin at the root where the metric is zero; we show this in Figure 6.6 by placing 0 above the root node. The subpath from the root to state [10] at depth 1 agrees in 1 position with $\underline{y}$; we show this by placing 1 above the node [10] at depth 1 in the trellis. Similarly, the metric is 1 for state [00] at depth 1. The branch from state [10] at depth 1 to state [11] at depth 2 has no agreements with $\underline{y}$, so the metric is still 1 at the latter state. The branch between state [00] at depth 1 to state [10] at depth 2 has one agreement

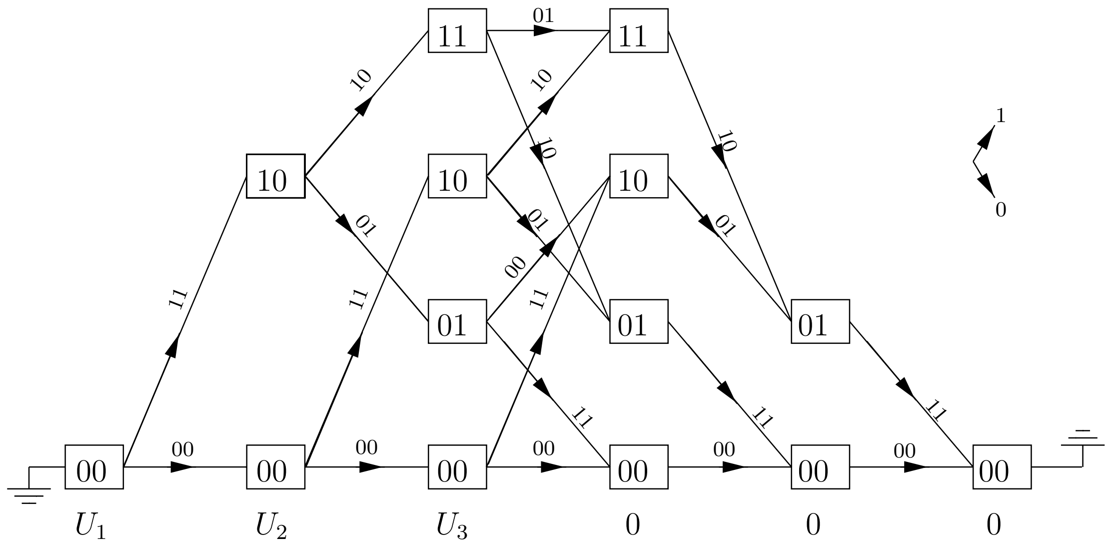
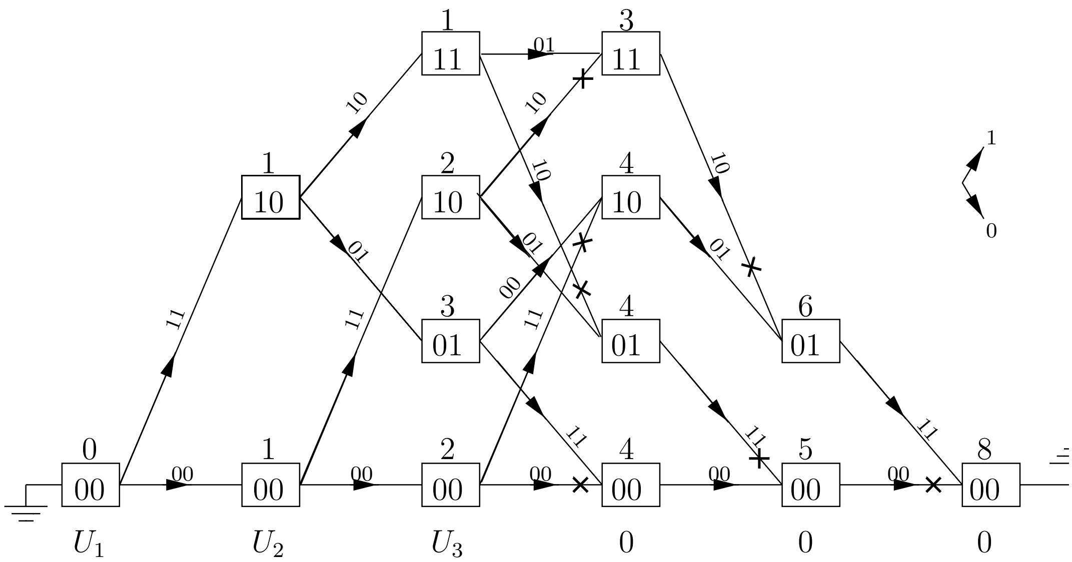


Figure 6.5: The $L_{\mathrm{t}} = 3$, $T = 2$ Trellis Code Encoded by the Convolutional Encoder of Figure 6.1.

with $\underline{y}$, so the metric is increased to $1 + 1 = 2$ at the latter state. Similarly, the metric for states [01] and [00] at depth 2 is found to be 3 and 2, respectively.



Received Word: 01 01 01 01 11

Decisions: $\hat{U}_1 = 1$ $\hat{U}_2 = 0$ $\hat{U}_3 = 1$

Figure 6.6: An Example of Viterbi (ML) Decoding on a BSC with $0 < \varepsilon < \frac{1}{2}$ for the Trellis Code of Figure 6.5.

At depth 3, something new and interesting begins to happen. There are two ways to reach state [11] at depth 3. Coming from state [11] at depth 2 would give a metric of $1 + 2 = 3$, but coming from state [10] at depth 2 would give a metric of only $2 + 0 = 2$ for state [11] at depth 3. But we now see that the latter subpath could not possibly be the prefix of the best path through the trellis because we could do still better if we replaced it by the former subpath. Thus, we *discard* the poorer subpath into state [11] at depth 3 (which we now show by a cross $\times$ on that subpath as it enters state [11] at depth 3) and, since we now are left with only one subpath into state [11] at depth 3, we show its metric, namely 3, above that state. We repeat this process of discarding the poorer subpath into each of the other three states at depth 3. Note that we are certain that we have not discarded the prefix of the best path through the trellis.

We now move to depth 4, where we first discard the poorer subpath into state [01], which then gives a metric $4+2=6$ for the better subpath. Something new again happens for state [00] at depth 4; the two subpaths into this state both have metric 5. But we are then free to discard either subpath and be sure that we have not eliminated *all* subpaths that are prefixes of best paths (in case there is a tie for best). We chose in Figure 6.6 quite arbitrarily to discard the upper of the two subpaths into state [00] at depth 4.

Finally, we reach depth 5, where we discard the poorer path into state [00], the only state at this depth. Our process of discarding subpaths was such that we know that there remains at least one best path through the trellis. But in fact there remains only one path through the trellis so this must be the best path! (It is also the unique best path because, in this example, the "winning path" was never involved in any ties along the way.) It is easiest to find this remaining path by moving backward from the toor node following the branches not "cut" by a cross $\times$. We find this path corresponds to the information sequence $[1,0,1]$ so our ML decoding decision is

$$[\hat{U}_1, \hat{U}_2, \hat{U}_3] = [1,0,1].$$

The above systematic procedure for performing ML decoding of a trellis code is called the *Viterbi algorithm*, in honor of its inventor. It was not noticed until several years after Viterbi introduced this decoding technique that the Viterbi algorithm can be viewed as the application of Bellman's *dynamic programming* method to the problem of decoding a trellis code.

6.3 Choosing the Metric for Viterbi Decoding

In the previous example, it was obvious from the form of the ML decoding rule how we could choose a convenient "metric" to use with the Viterbi algorithm. We now shall see how one can choose such a metric for any DMC. Our metric must possess two features:

(i) It must be an additive branch function in the sense that the metric for any subpath in the trellis must be the sum of the metrics for each branch in that subpath, and

(ii) it must have the property that the path through the trellis (from root to toor) with the largest metric must be the path that would be chosen by an ML decoder.

It is easy to see how to choose such a metric for any DMC. Let $\underline{y} = [y_1, y_2, \ldots, y_N]$ denote the received word and let $\underline{x} = [x_1, x_2, \ldots, x_N]$ denote an arbitrary path through the trellis, i.e., an arbitrary codeword. An ML decoder for a DMC must choose $\underline{x}$ to maximize

$$P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}) = \prod_{n=1}^{N} P_{Y|X}(y_n|x_n) \tag{6.14}$$

or, equivalently, to maximize

$$\log P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}) = \sum_{n=1}^{N} \log P_{Y|X}(y_n|x_n). \tag{6.15}$$

We see now that we could, in fact, simply take $\log P_{Y|X}(y_n|x_n)$ as the metric for each *digit* x_n in the path, the branch metric being just the sum of the metrics on each digit in the path. However, this is not usually a convenient metric. For convenience in implementing the VA, one usually prefers that the metric have two further properties:

(iii) All branch metrics are "small" nonnegative numbers , and

(iv) the smallest possible branch metric is 0 for each possible value of a received symbol.

To obtain a metric with these two additional properties, we begin by noting that maximization of (6.15) is equivalent to maximization by choice of $\underline{x}$ of

$$\alpha \log P_{\underline{Y}|\underline{X}}(\underline{y}|\underline{x}) + \alpha \sum_{n=1}^{N} f(y_n) = \sum_{n=1}^{N} \alpha \left[\log P_{Y|X}(y_n|x_n) + f(y_n)\right] \tag{6.16}$$

where α is any positive number and where f is a completely arbitrary real-valued function defined over the channel output alphabet. It follows from (6.16) that if we choose, for each y,

$$f(y) = -\log \left[\min_{x} P_{Y|X}(y|x)\right], \tag{6.17}$$

then the smallest digit metric, and hence also the smallest branch metric, will be 0. We are then free to choose α to make the digit metrics (and hence also the branch metric) integers – in fact we usually settle for the smallest α such that the exact branch metrics are *well-approximated* by integers. We summarize these observations as:

Viterbi Decoding Metric for a DMC: To perform ML decoding of the received word $\underline{y} = [y_1, y_2, \ldots, y_N]$ of a trellis code on a DMC, the metric for any subpath can be taken as the sum of the digit metrics, $\mu(x_n, y_n)$ for the digits x_n in that subpath where

$$\mu(x, y) = \alpha[\log P_{Y|X}(y|x) + f(y)] \tag{6.18}$$

for any positive α and any real-valued function f. Moreover, the choice (6.17) for f guarantees that, for each value of y, the minimum digit metric is 0.

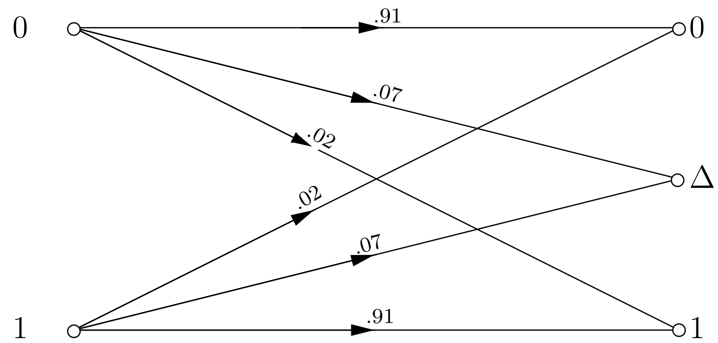


Figure 6.7: The Binary Symmetric Erasure Channel (BSEC) of Example 6.1.

Example 6.1: For the BSEC of Figure 6.7,

$$\min_{x,y} P_{Y|X}(y|x) = .02$$

so that (6.17) gives

$$f(0) = f(1) = -\log(.02) = 5.64$$
$$f(\Delta) = -\log(.07) = 3.84$$

where we have chosen the base 2 for our logarithms. We can thus construct the following table.

$y \backslash x$	0	1
0	5.5	0
Δ	0	0
1	0	5.5

Table for $\log P(y|x) + f(y)$.

By inspection of this table, we see that the choice $\alpha = 1/5.50$ will yield integer metrics as in Table 6.1.

$y \backslash x$	0	1
0	1	0
Δ	0	0
1	0	1

Table 6.1: $\mu(x, y)$ metric table.

In fact, the reader should see that this same final metric table will result for any BSEC for which $P_{Y|X}(0|0) > P_{Y|X}(1|0)$.

To check his or her understanding of the VA, the reader should now use the VA to perform ML decoding of the trellis code of Figure 6.5 when

$$\underline{y} = [000\Delta10\Delta01\Delta]$$

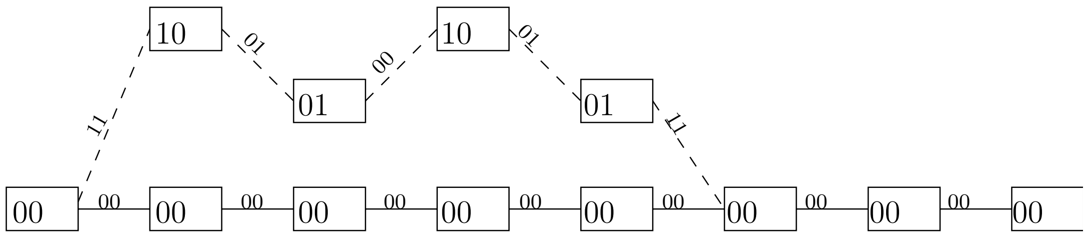
is received over the BSEC of Figure 6.7. Using the metrics of Example 6.1, he or she should find that

$$[\hat{U}_1, \hat{U}_2, \hat{U}_3] = [0, 1, 1]$$

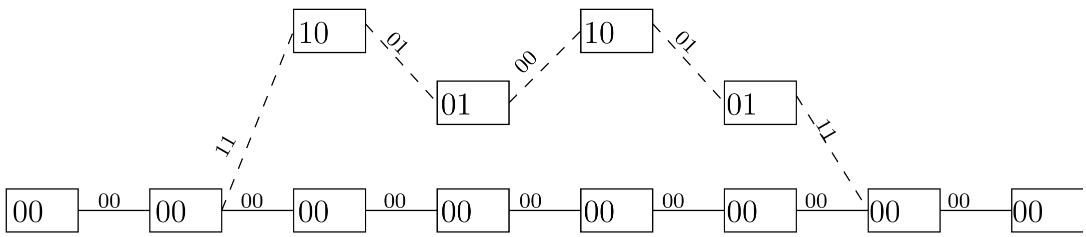
is the (unique) ML decision and that the total metric for the corresponding codeword is 6.

6.4 Counting Detours in Convolutional Codes

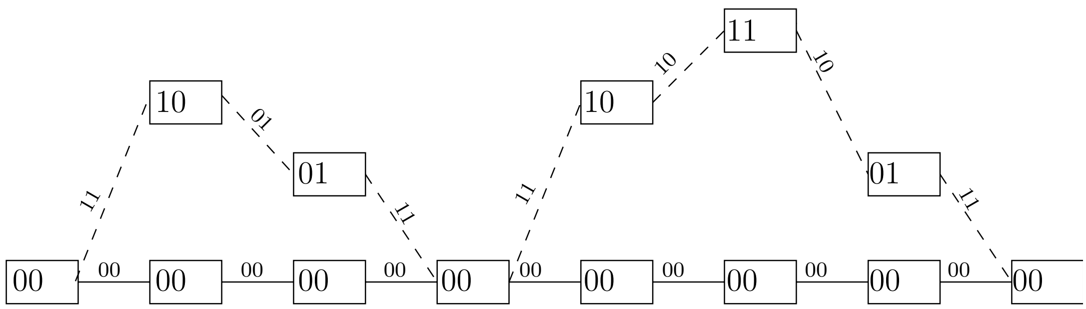
In the next section, we will develop a tight upper bound on the bit error probability of a convolutional code when used with Viterbi decoding on a DMC. The key to this bound is the enumeration of "detours" from the "correct path" in the trellis, which we now consider.



(a) A detour with distance $d = 6$ from the correct path containing $i = 2$ information bit errors, beginning at node 1.



(b) A detour with $d = 6$ and $i = 2$, beginning at node 2.



(c) Two detours: the first with $d = 5$ and $i = 1$ beginning at node 1, the second with $d = 6$ and $i = 2$ beginning at node 4.

Figure 6.8: Three possible decoded paths (shown dashed where different from the correct path) for the $L = 6$, $T = 2$ trellis code encoded by the convolutional encoder of Figure 6.1 when $\underline{0}$ is the correct path.

Suppose that we had decided to drive from Zürich to Paris and had indicated our intended route on a road map, but when we had actually made the trip we were forced to leave this route in several places because of road construction, floods, heavy snow, hunger, etc. Suppose then that after the trip we had indicated our actual route on the same road map. We would then say that we had taken a "detour" at each point where we had left our intended route, and that each "detour" had ended just when we returned to a point on the correct path – even if another "detour" immediately began! The concept of a "detour" in a trellis code is quite the same as that for a road map.

Suppose that we had used the $R_t = 1/2$ convolutional encoder of Figure 6.1 to encode the $L_t = 5$ information bits before inserting the tail of $T = 2$ dummy information bits to return the encoder to its zero state. Suppose further that, for a particular transmission, the information bits were all zero so that the all-zero path (which we denote hereafter by $\underline{0}$) through the trellis is the *correct path* in the sense that the decoder correctly decodes if and only if it chooses this path. In Figure 6.8, we show various possibilities for the *decoded path*, the decoded path being shown by dashed lines for those branches where it differs from the correct path. The decoded path of Figure 6.8(a) contains one *detour* from the correct path; the detour begins at node 1, has Hamming distance $d = 6$ from the corresponding segment of the correct path, and contains $i = 2$ information bit errors since the first and third decoded information bits are both 1. The decoded path of Figure 6.8(b) has a detour beginning at node 2 that also has $d = 6$ and $i = 2$; in fact we see that this detour is essentially the same as that in Figure 6.8(a), differing only in the point where the detour begins. Finally, the decoded path of Figure 6.8(c) contains two detours, one beginning at node 1 and the other at node 4. Notice that there are two separate detours even though the decoded path has no branches in common with the correct path. A detour begins at a node where the decoded path leaves the correct path and ends at that node where the decoded path again meets the correct path, even if another detour immediately begins!

We now wish to count the number of detours with a given distance d and given number i of information bit errors that begins at the j-th node of the correct path. We begin by assuming that $\underline{0}$ is the correct path and will show later that, because of the "linearity" of a convolutional encoder, we would get the same count had we considered any other path through the trellis as the correct path. It should be clear that the count will depend on L_t and that we will get an upper bound on the count for any finite trellis if we consider $L_t = \infty$. Moreover, when $L_t = \infty$, we see that the detours that begin at node 1 are essentially the same as those that begin at any other node [in the sense that the detour of Figure 6.8(b) is essentially the same as that of Figure 6.8(a)] because of the "time-invariant" nature of a convolutional encoder. Thus our interest is in the quantity $a(d, i)$ which we define as

$a(d, i) =$ number of detours beginning at node 1 that have Hamming distance d from the correct path, $\underline{0}$, and contain i information bit errors in the $L_t = \infty$ trellis for a given convolutional encoder.

It is not hard to see from the trellis of Figure 6.5 (extended conceptually to $L_t = \infty$) that $a(d, i) = 0$ for $d < 5$, that $a(5, 1) = 1$ and that $a(6, 2) = 2$. But we need a systematic approach if we are not going to exhaust ourselves in considering all d and i, even for such a simple convolutional encoder as that of Figure 6.1.

The secret to a simple evaluation of $a(d, i)$ lies in the state-transition diagram of the convolutional encoder, such as that of Figure 6.4 for the encoder of Figure 6.1. We have redrawn this state-transition diagram in Figure 6.9 after first removing the zero state with its self-loop, then relabelling each branch with $I^i D^d$ where i is the number of non-zero information bits for this transition (and hence is either 0 or 1 for a convolutional encoder with a single input), and d is the number of non-zero encoded digits (and hence is $0, 1$ or 2 for an encoder giving two encoded digits per time unit) or, equivalently, the Hamming distance of this branch from the all-zero branch. Thus, we obtain a flowgraph such that the possible paths from input terminal to output terminal correspond to the possible detours beginning at node 1 because the first branch of such a path corresponds to the immediate departure from the zero state and the last branch corresponds to the first return again to that state. Moreover, the product of the branch labels for such a path will equal $I^i D^d$ where i is the total number of non-zero information bits on this detour and d is the total Hamming distance of this detour from the corresponding segment of the correct path $\underline{0}$. We shall refer to this flowgraph as the *detour flowgraph* for the convolutional encoder. But the *transmission gain* of a flowgraph is, by definition, the sum over all paths[1] from the input terminal to the output terminal of the product of the branch labels on each path. Thus, we have proved:

Lemma 6.1: The transmission gain, $T(D, I)$, of the detour flowgraph for a binary convolutional encoder

[1] "paths", in the sense used here, can contain loops.

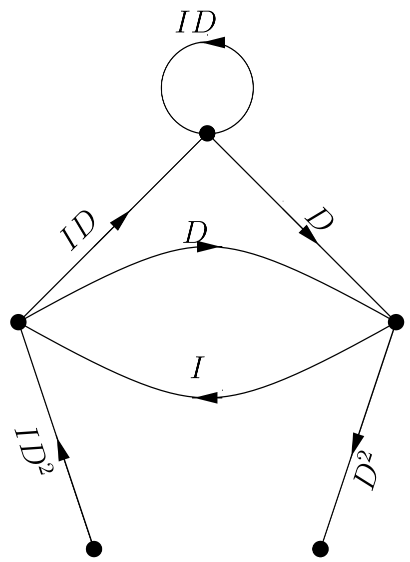


Figure 6.9: The Detour Flowgraph for the Convolutional Encoder of Figure 6.1.

is given by

$$T(D,I) = \sum_{i=1}^{\infty}\sum_{d=1}^{\infty} a(d,i) I^i D^d. \tag{6.19}$$

For instance, for the detour flowgraph of Figure 6.9 we find, by using any of the standard means for calculating the transmission gain of a flowgraph, that

$$T(D,I) = \frac{ID^5}{1-2ID} \tag{6.20}$$

or, equivalently,

$$T(D,I) = ID^5(1 + 2ID + 4I^2D^2 + \ldots). \tag{6.21}$$

Thus, we see that, for the convolutional encoder of Figure 6.1,

$$a(d,i) = \begin{cases} 2^{i-1}, & d = i+4, \qquad i = 1,2,3\ldots \\ 0 & \text{otherwise.} \end{cases} \tag{6.22}$$

It remains now to justify our earlier claim that $a(d,i)$ has the same value when defined for an arbitrary correct path rather than for $\underline{0}$. To show this, we let $g(\underline{u})$ denote the code word that is produced by the convolutional encoder when the information sequence is $\underline{u}$. But a convolutional encoder is linear in the sense that for any two information sequences, $\underline{u}$ and $\underline{u}'$,

$$g(\underline{u} \oplus \underline{u}') = g(\underline{u}) \oplus g(\underline{u}') \tag{6.23}$$

where the "addition" is component-by-component addition modulo-two. Suppose that $\underline{u}$ is the information sequence for the correct path and that $\underline{u}'$ is that for a decoded path with a detour beginning at node 1, i.e., $\underline{u}$ and $\underline{u}'$ differ in the first time unit. We then have for the Hamming distance between this detour and the corresponding segment of the correct path

$$\begin{aligned} d(g(\underline{u}), g(\underline{u}')) &= d(g(\underline{u}) \oplus g(\underline{u}), g(\underline{u}) \oplus g(\underline{u}')) \\ &= d(\underline{0}, g(\underline{u} \oplus \underline{u}')) \end{aligned} \tag{6.24}$$

where we have used the fact that adding a common vector modulo-two to two vectors does not change the Hamming distance between them, and then made use of (6.23). But $\underline{u} \oplus \underline{u}'$ is an information sequence that is non-zero in the first time unit, i.e., the information sequence for a detour beginning at node 1 when $\underline{0}$ is the correct path. The following result now follows from (6.24) and our previous observations about finite trellises.

Lemma 6.2: For any correct path and any $j, a(d,i)$ is the number of detours beginning at node j that have Hamming distance d from the corresponding segment of the correct path and contain i information bit errors in the $L_\mathrm{t} = \infty$ trellis for the given convolutional encoder, and is an upper bound on this number for every finite L_t.

6.5 A Tight Upper Bound on the Bit Error Probability of a Viterbi Decoder

We will shortly use $T(D, I)$ as the basis for a useful upper bound on the bit error probability, P_b, of a Viterbi decoder for a binary convolutional code. First, however, we recall and extend our understanding of two matters that arose in our discussion of block codes.

First, we recall the Bhattacharyya bound (5.28) for a block code with two codewords. We note that for a binary-input DMC, the summation on the right side of (5.28) equals 1 for all n where $x_{1n} = x_{2n}$ but equals

$$\sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)} = 2^{-D_\mathrm{B}}$$

for those n where $x_{1n} \neq x_{2n}$. But since the number of these latter n is just the Hamming distance, $d(\underline{x}_1, \underline{x}_2)$, between the two codewords, we see that (5.28) implies:

Lemma 6.3: For a binary-input DMC and the code $(\underline{x}_1, \underline{x}_2)$, the worst-case block error probability for an ML decoder satisfies

$$(P_\mathrm{B})_\mathrm{wc} \leq (2^{-D_\mathrm{B}})^{d(\underline{x}_1, \underline{x}_2)} \tag{6.25}$$

where D_B is the Bhattacharyya distance between the two channel input digits and $d(\underline{x}_1, \underline{x}_2)$ is the Hamming distance between the two codewords.

Next, we recall the definition (4.46) of the bit error probability of a decoder for a block code with K information bits, namely,

$$P_\mathrm{b} = \frac{1}{K}\sum_{i=1}^{K} P_{\mathrm{e},i}$$

where $P_{\mathrm{e},i}$ is the probability of a decoding error in the i-th information bit. We now define V_i to be the indicator random variable for an error in the i-th information bit, i.e., $V_i = 1$ when such an error occurs and $V_i = 0$ otherwise. But then

$$P_{V_i}(1) = P_{\mathrm{e},i}$$

so that

$$\mathrm{E}[V_i] = P_{\mathrm{e},i}\,. \tag{6.26}$$

Using (6.26) in (4.46), we find

$$\begin{aligned} P_\mathrm{b} &= \frac{1}{K}\sum_{i=1}^{K} \mathrm{E}[V_i] \\ &= \frac{1}{K}\mathrm{E}\left[\sum_{i=1}^{K} V_i\right] \\ &= \mathrm{E}\left[\frac{1}{K}\sum_{i=1}^{K} V_i\right]. \end{aligned} \tag{6.27}$$

But the random variable $\sum_{i=1}^{K} V_i$ is the total number of information bit errors made by the decoder so (6.27) can be stated as:

Lemma 6.4: The bit error probability of a decoder for a block code equals the average fraction of erroneously decoded information bits.

We are now ready to derive a tight upper bound on P_b for a Viterbi decoder for the trellis code of length $L_\text{t} + T$ branches generated by a convolutional encoder. We begin by defining the random variable W_j as the number of information bit errors made by the Viterbi decoder as it follows a detour beginning at node j. Naturally, $W_j = 0$ when the decoder does not follow a detour beginning at node j either because it is already on a detour that began earlier or because it follows the correct path from node j to node $j+1$. Then $\sum_{j=1}^{L_\text{t}} W_j$ is the total number of information bit errors made by the Viterbi decoder so that Lemma 6.4 gives

$$P_\text{b} = \mathrm{E}\left[\frac{1}{k_0 L_\text{t}} \sum_{j=1}^{L_\text{t}} W_j\right], \tag{6.28}$$

where k_0 is the *number of information bits per time unit* that enter the convolutional encoder. For the encoder of Figure 6.1, $k_0 = 1$ – but of course we wish our treatment here to be general.

Next, we define B_{jk} to be the event that the Viterbi decoder follows the k-th detour at node j on the correct path (where the detours are numbered in any order). Letting d_k be the Hamming distance between this detour and the corresponding segment of the correct path, we can invoke Lemma 6.3 to assert that

$$P(B_{jk}) \le (2^{-D_\text{B}})^{d_k}. \tag{6.29}$$

Letting i_k be the number of information bit errors on this detour, we see that

$$W_j = \begin{cases} i_k, & \text{if } B_{jk} \text{ occurs for some } k \\ 0, & \text{otherwise.} \end{cases} \tag{6.30}$$

Thus, (6.30) and the theorem on total expectation imply

$$\mathrm{E}[W_j] = \sum_k i_k P(B_{jk}). \tag{6.31}$$

Using (6.29) in (6.31) now gives

$$\mathrm{E}[W_j] \le \sum_k i_k \left(2^{-D_\text{B}}\right)^{d_k}. \tag{6.32}$$

But, by the definition of $a_j(d,i)$, the number of indices k in (6.32) for which $i_k = i$ and $d_k = d$ is just $a_j(d,i)$ so that (6.32) can be written

$$\mathrm{E}[W_j] \le \sum_i \sum_d i a_j(d,i) \left(2^{-D_\text{B}}\right)^d. \tag{6.33}$$

But Lemma 6.2 states that

$$a_j(d,i) \le a(d,i) \tag{6.34}$$

for all j. Using (6.34) in (6.33) and then (6.33) in (6.28), we have finally

$$P_\text{b} \le \frac{1}{k_0} \sum_i \sum_d i a(d,i) \left(2^{-D_\text{B}}\right)^d, \tag{6.35}$$

which is our desired bound. Notice that (6.35) holds for every L_t – even for $L_\text{t} = \infty$ when we never insert the tail of dummy information bits to return the encoder to the zero state and when R_t is the true information rate of the trellis code.

It remains only to express the right side of (6.35) in terms of the transmission gain $T(D, I)$ of the detour flowgraph. Differentiating with respect to I in (6.19) gives

$$\frac{\partial T(D, I)}{\partial I} = \sum_i \sum_d i a(d, i) I^{i-1} D^d. \tag{6.36}$$

Comparing (6.35) and (6.36), we obtain the following result:

Theorem 6.1: The bit error probability for Viterbi decoding of the trellis code generated by a binary convolutional encoder used on a DMC satisfies

$$P_\mathrm{b} \leq \frac{1}{k_0} \frac{\partial T(D, I)}{\partial I}\bigg|_{I=1, D=2^{-D_\mathrm{B}}} \tag{6.37}$$

provided that the power series on the right of (6.36) converges at $I = 1$ and $D = 2^{-D_\mathrm{B}}$, where k_0 is the number of information bits per time unit that enter the convolutional encoder, where $T(D, I)$ is the transmission gain of the detour flowgraph of the convolutional encoder, and where D_B is the Bhattacharyya distance between the two input letters of the DMC.

The bound (6.37) holds for all trellis lengths L_t (including $L_\mathrm{t} = \infty$) and all assignments of probabilities to the $k_0 \cdot L_\mathrm{t}$ "information bits".

For the encoder of Figure 6.1, we find by differentiation in (6.20) that

$$\frac{\partial T(D, I)}{\partial I} = \frac{D^5}{(1 - 2ID)^2}.$$

Suppose this convolutional code is used on a BEC with erasure probability δ. Then (5.34) gives

$$2^{-D_\mathrm{B}} = \delta$$

so that (6.37) becomes (because $k_0 = 1$ for this encoder)

$$P_\mathrm{b} \leq \frac{\delta^5}{(1 - 2\delta)^2}.$$

In particular, for $\delta = \frac{1}{10}$, our bound becomes

$$P_\mathrm{b} \leq 1.56 \times 10^{-5}.$$

It has been determined from simulations that the bound of (6.37) is so tight in general that it can be used to design convolutional decoders for particular applications without appreciable extra complexity resulting from the fact that an upper bound on P_b, rather than the true P_b, is used. This should not surprise us in light of the tightness of the Bhattacharyya bound on which (6.37) is based.

When the memory, T, of the convolutional encoder is very small, it is practical to calculate $T(D, I)$, as we have done here for the encoder of Figure 6.1. For larger values of T but still in the range where Viterbi decoding is practical (say, $k_0 T \leq 10$), it is impractical to calculate $T(D, I)$ as a function of D and I from the detour flowchart, although it is still easy enough to calculate $T(D, I)$ for particular values of D and I. In this case, we can still make use of the bound (6.37) by approximating the derivative, for instance as

$$\frac{\partial T(D, I)}{\partial I}\bigg|_{I=1, D=2^{-D_\mathrm{B}}} \approx \frac{T(2^{-D_\mathrm{B}}, 1.01) - T(2^{-D_\mathrm{B}}, .99)}{.02} \tag{6.38}$$

which requires the evaluation of $T(D, I)$ for only two particular cases.

Finally, we should mention that when $L_t = \infty$ (as is normally the case for practical systems that use Viterbi decoders) we do not want, of course, to wait until the end of the trellis before announcing the decoded information sequence. Theoretical considerations indicate, and simulations have confirmed, that P_b is virtually unchanged when the Viterbi decoder is forced to make its decisions with a delay of about 5 constraint lengths. In this case, the decoded information bits at time unit j are taken as their values in the information sequence for the currently best path to the state with the highest metric at time unit $j + 5(T + 1)$.

6.6 Random Coding – Trellis Codes and the Viterbi Exponent

We are about to develop a random coding bound for trellis codes which will show that, in an important way, trellis codes are superior to ordinary block codes. To do this, however, we must first suitably generalise our concept of a "trellis code".

The general form of what we shall call an "(n_0, k_0) trellis encoder with memory T" is shown in Figure 6.10. At each "time unit", k_0 information bits enter the encoder while n_0 channel input digits leave the encoder. Thus, the *rate* of the trellis code is

$$R_t = \frac{k_0}{n_0} \ \text{bits/use} \tag{6.39}$$

(which is of course a true information rate only if all the information bits are statistically independent and each is equally likely to be 0 or 1.) Notice that, although our information symbols are binary, the encoded symbols are digits in the channel input alphabet. The n_0 encoded digits formed each time unit depend only on the current k_0 information bits and on the k_0T information bits in the T most recent time units, but the functional form of this dependence is arbitrary – in particular this function need not be linear nor time-invariant as it was for the convolutional codes previously considered. By way of convention, we assume that the "past" information bits are all zeroes when encoding begins.

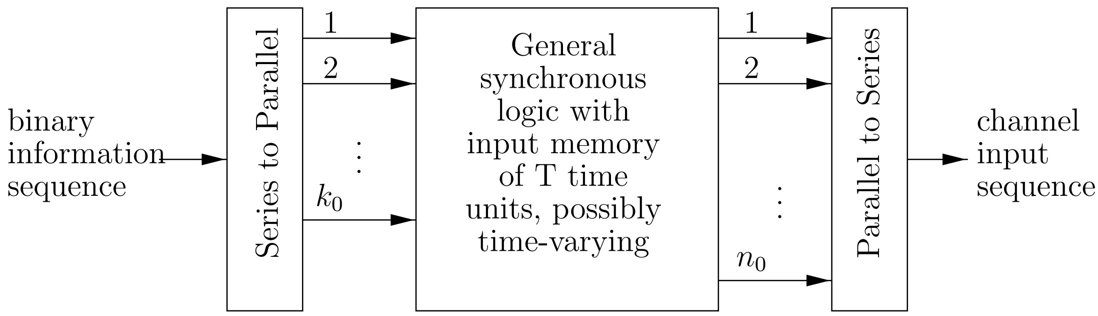


Figure 6.10: A general (n_0, k_0) trellis encoder with memory T and rate $R_t = \frac{k_0}{n_0}$.

It makes the notation much simpler if we write

$$\underline{U}_i = [U_{(i-1)k_0+1}, U_{(i-1)k_0+2}, \ldots, U_{ik_0}] \tag{6.40}$$

and

$$\underline{X}_i = [X_{(i-1)n_0+1}, X_{(i-1)n_0+2}, \ldots, X_{in_0}] \tag{6.41}$$

for the "byte" of k_0 information bits that enter the encoder and the "byte" of n_0 channel input symbols that leave the encoder, respectively, at time unit $i = 1, 2, 3, \ldots$. Then Figure 6.10 corresponds to the functional relationship

$$\underline{X}_i = f_i(\underline{U}_i, \underline{U}_{i-1}, \ldots, \underline{U}_{i-T}) \tag{6.42}$$

where, by way of convention,

$$\underline{U}_j = [0, 0, \ldots, 0], \qquad j < 1. \tag{6.43}$$

We will not in fact make much use of this functional description, except to observe that it shows clearly that

$$\sigma_i = [\underline{U}_{i-1}, \underline{U}_{i-2}, \ldots, \underline{U}_{i-T}] \tag{6.44}$$

is a valid choice for the encoder state at time i, since (6.42) can be written

$$\underline{X}_i = f_i(\underline{U}_i, \sigma_i). \tag{6.45}$$

Notice from (6.40) and (6.44) that there are exactly $2^{k_0 T}$ distinct values for the state σ_i, one of which is the all-zero state.

We now define the $(L_\mathrm{t}, T, n_0, k_0)$ *trellis code*, determined by a particular (n_0, k_0) trellis encoder with memory T, to be the block code whose list of codewords consists of the

$$M = 2^{k_0 L_\mathrm{t}} = 2^{n_0 R_\mathrm{t} L_\mathrm{t}} \tag{6.46}$$

output sequences of length

$$N = (L_\mathrm{t} + T) n_0 \tag{6.47}$$

from that encoder, corresponding to the M distinct choices of $\underline{U}_1, \underline{U}_2, \ldots, \underline{U}_{L_\mathrm{t}}$ followed by the "tail" $\underline{U}_{L_\mathrm{t}+1} = \underline{U}_{L_\mathrm{t}+2} = \cdots = \underline{U}_{L_\mathrm{t}+T} = [0, 0, \ldots, 0]$ which drives the encoder back to the all-zero state. Because the rate R of a block code is defined by $M = 2^{NR}$, we see from (6.46) and (6.47) that

$$R = \frac{L_\mathrm{t}}{L_\mathrm{t} + T} R_\mathrm{t} \tag{6.48}$$

so that $L_\mathrm{t} \gg T$ implies $R \approx R_\mathrm{t}$. Notice that the initial and final states, σ_1 and $\sigma_{L_\mathrm{t}+T+1}$, are both the all-zero state. If we now made a diagram (of which that in Figure 6.5 with the digits deleted from the branches, would be a special case) showing at each time instant $i, 1 \le i \le L_\mathrm{t} + T + 1$, the possible values of σ_i and then showing the possible successors of each state by branches from that state, we would obtain a directed graph with M distinct paths from the root node σ_1 to the toor node $\sigma_{L_\mathrm{t}+T+1}$. This graph is what we mean by a (k_0, L_t, T) *trellis.* The only property of this trellis that we have any real use for is the following.

Lemma 6.5: Regardless of which path from the root to the toor node is designated as the correct path in a (k_0, L_t, T) trellis, there are the same number $b(j, l)$ of detours that begin at node j and end at node $j + l$. Moreover, $b(j, l) = 0$ for $l \le T$ and

$$b(j, l) \le (2_0^k - 1) 2^{k_0 (l - T - 1)}, \qquad \text{all } l > T. \tag{6.49}$$

Proof: Inequality (6.49) follows from the facts that

(i) there are $2^{k_0} - 1$ choices for $\underline{U}_j$ different from its values on the correct path as is needed to begin a detour;

(ii) all the considered detours must merge to the same state σ_{j+l} as on the correct path and this, because of (6.44), determines $[\underline{U}_{j+l-1}, \underline{U}_{j+l-2}, \ldots, \underline{U}_{j+l-T}]$ for all these detours; and

(iii) there are $2^{k_0(l-T-1)}$ choices for the immediate information branches $\underline{U}_{j+1}, \underline{U}_{j+2}, \ldots, \underline{U}_{j+L-T-1}$ but not all of these choices will correspond to a detour ending at node $j + l$ because some choices may cause the detour to merge to the correct path before this node.

Notice that (i) and (ii) cannot simultaneously be satisfied unless $j + l - T > j$, which shows that $b(j, l) = 0$ if $l \le T$.

When we specify the encoding function f_i of (6.45), we are merely specifying the particular n_0 channel input digits that should be placed on the branches leading from the states in the trellis at depth $i - 1$

from the root node, i.e., specifying how the paths in the trellis from root to toor will be labelled with the M codewords in this particular trellis code. Thus, instead of defining an ensemble of trellis codes by the choice of these functions f_i for $1 \le i \le L_t + T$, we can equivalently define this ensemble by the choice of the n_0 digits that we put on each branch of the (k_0, L_t, T) trellis.

Consider an ensemble of (L_t, T, n_0, k_0) trellis codes with probabilities assigned to each code such that for some given probability distribution $Q(x)$ over the channel input alphabet the following two properties hold:

(1) Along any given path from the root to the toor node in the trellis, the corresponding codeword $\underline{X}$ appears with the same probability as if its digits were selected independently according to $Q(x)$, i.e.,

$$Q_{\underline{X}}(x_1, x_2, \ldots, x_N) = \prod_{n=1}^{N} Q(x_n) \tag{6.50}$$

where N of course is given by (6.47).

(2) The ensemble is *pairwise independent* in the sense that, given any choice of the correct path and any choice of a detour starting at node j and ending at node $j + l$ (where $l > T$), the encoded sequence along the detour and that along the corresponding segment of the correct path appear jointly with the same probability as if each of their digits was selected independently according to $Q(x)$.

We note that a conceptually simple way to satisfy both conditions (1) and (2) is to make all the digits on all of the branches in the trellis appear over the ensemble of codes as if they were selected independently according to $Q(x)$, i.e., to assign to every possible (L_t, T, n_0, k_0), trellis code (for the given channel input alphabet) a probability equal to the product of $Q(x)$ taken over every digit x on every branch of the trellis. We shall see later, however, that these conditions can also be satisfied by the appropriate assignment of probabilities to the much smaller ensemble of *linear* trellis codes. This is of practical importance since it shows that the linear codes, which are the easiest to implement, will achieve an upper bound on error probability that we can prove for the more general ensemble.

We now get down to the real task at hand, namely proving an upper bound on the average *bit* error probability, $\mathrm{E}[P_b]$, for the above ensemble of trellis codes. But first we should explain why we are going after $\mathrm{E}[P_b]$ rather than the average *block* error probability $\mathrm{E}[P_B]$. After all, trellis codes are block codes, at least when $L_t < \infty$. And didn't we promise in Section 4.5 to prove upper bounds for P_B rather than for P_b? That promise should and will now be broken because we want a bound that will apply as $L_t \to \infty$. Except for trivial noiseless channels, we must face the fact that $P_B \to 1$ as $L_t \to \infty$ for any given (n_0, k_0) encoder with memory T, i.e., if we force our coding system to work forever, then it must ultimately make some decoding error! Thus, the best upper bound we could possibly prove on $\mathrm{E}[P_B]$ that would hold for $L_t = \infty$ would be the trivial upper bound 1. When we wish to consider $L_t = \infty$, only P_b has practical significance!

It should be obvious to the reader that ML decoding on a DMC for an (L_t, T, n_0, k_0) trellis code can be implemented by a Viterbi decoder with $2^{k_0 T}$ states. We now proceed to overbound $\mathrm{E}[P_b]$ for such ML decoding on a DMC when the expectation is over the ensemble of codes satisfying the conditions (1) and (2) above.

Let E_{jl} be the event that the ML decoder chooses a detour from the correct path beginning at node j and ending at node $j+l$, $l > T$. This cannot happen unless one of the $b(j,l)$ such detours would be chosen by an ML decoder in preference to the corresponding segment of the correct path and this in turn, because of conditions (1) and (2) on our example, can be overbounded by Gallager's random coding bound (5.94) on $\mathrm{E}[P_B]$ for the ensemble of block codes with $M = b(j,l) + 1$ codewords of length ln_0 (the number of encoded digits along the detour). Thus, independently of the choice of the information sequence,

$$\mathrm{E}[P(E_{jl})] \le [b(j,l)]^{\rho}\, 2^{-ln_0 E_0(\rho, Q)}, \qquad 0 \le \rho \le 1. \tag{6.51}$$

The use of the bound (6.49) of Lemma 6.5 in (6.51) gives

$$\mathrm{E}[P(E_{jl})] \leq \left(2^{k_0}-1\right)^{\rho} 2^{\rho k_0(l-T-1)} 2^{-l n_0 E_0(\rho,Q)}. \tag{6.52}$$

If we now define the *constraint length*, N_t, of the trellis code as the maximum number of encoded digits that could be affected by a single information bit as it passes through the encoder, i.e., as

$$N_t = (T+1)n_0, \tag{6.53}$$

we can rewrite (6.52) as

$$\mathrm{E}[P(E_{jl})] \leq \left(2^{k_0}-1\right)^{\rho} 2^{-[n_0 E_0(\rho,Q)-\rho k_0](l-T-1)} 2^{-N_t E_0(\rho,Q)}.$$

for $0 \leq \rho \leq 1$. But $\left(2^{k_0}-1\right)^{\rho} < 2^{k_0}$ so we can weaken this last inequality to

$$\mathrm{E}[P(E_{jl})] < 2^{k_0} 2^{-n_0[E_0(\rho,Q)-\rho R_t](l-T-1)} 2^{-N_t E_0(\rho,Q)}, \qquad 0 \leq \rho \leq 1 \tag{6.54}$$

which holds for all $l > T$.

Now defining W_j as in the previous section, i.e., as the number of information bit errors made by the ML decoder as it follows a detour beginning at node j, we see that

$$W_j \begin{cases} \leq k_0(l-T), & \text{if the event } E_{jl} \text{ occurs for some } l > T \\ = 0, & \text{otherwise,} \end{cases} \tag{6.55}$$

where the inequality arises from the fact that, when E_{jl} occurs, then the only information bit errors that are possible along the detours are those in $\underline{U}_j, \underline{U}_{j+1}, \ldots, \underline{U}_{j+l-T-1}$ as follows from property (ii) in the proof of Lemma 6.5. Next, we note that because of (6.55) the theorem on total expectation gives for any one code

$$E^*[W_j] = \sum_{l=T+1}^{L_t+T+1-j} E^*[W_j|E_{jl}]P(E_{jl})$$

where the asterisk indicates that this expectation is over channel behavior for a given information sequence and a given code, the same as used to determine $P(E_{jl})$. Moreover, the inequality in (6.55) gives further

$$E^*[W_j] \leq \sum_{l=T+1}^{L_t+T+1-j} k_0(l-T)P(E_{jl}),$$

which we now further average over the ensemble of trellis codes to get

$$\mathrm{E}[W_j] \leq \sum_{l=T+1}^{L_t+T+1-j} k_0(l-T)\mathrm{E}[P(E_{jl})]. \tag{6.56}$$

Using (6.54) in (6.56) and changing the index of summation to $i = l - T$ now gives

$$\mathrm{E}[W_j] < k_0 2^{k_0} 2^{-N_t E_0(\rho,Q)} \sum_{i=0}^{L_t+1-j} i \left\{2^{-n_0[E_0(\rho,Q)-\rho R_t]}\right\}^{i-1}. \tag{6.57}$$

Further weakening inequality (6.57) by extending the summation to $i = \infty$, then using the fact that differentiation of the geometric series

$$\sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}, \qquad |\alpha| < 1 \tag{6.58}$$

gives

$$\sum_{i=1}^{\infty} i\alpha^{i-1} = \frac{1}{(1-\alpha)^2}, \qquad |\alpha| < 1 \tag{6.59}$$

we find that

$$\mathrm{E}[W_j] < k_0 c(R_\mathrm{t}, \rho, Q) 2^{-N_t E_0(\rho, Q)}, \tag{6.60}$$

provided that

$$R_\mathrm{t} < \frac{E_0(\rho, Q)}{\rho} \tag{6.61}$$

so that the magnitude of the bracketed expression in (6.57) is less than 1, where we have defined the coefficient function

$$c(R_\mathrm{t}, \rho, Q) = \frac{2^{k_0}}{\left\{1 - 2^{-n_0[E_0(\rho,Q) - \rho R_\mathrm{t}]}\right\}^2}, \tag{6.62}$$

which, while complicated, is unimportant since it has no dependence on the constraint length N_t of the trellis code. Notice that our upper bound (6.60) has no dependence on L_t – which is why we extended the summation in (6.57) to infinity! Notice further that it also has no dependence on j. But we now recall (6.28) which implies that

$$\mathrm{E}[P_\mathrm{b}] = \frac{1}{k_0 L_\mathrm{t}} \sum_{j=1}^{L_\mathrm{t}} \mathrm{E}[W_j]. \tag{6.63}$$

Using (6.60) in (6.63) gives finally our desired bound.

Viterbi's Random Coding Bound for Trellis Codes: Over any ensemble of $(L_\mathrm{t}, T, n_0, k_0)$ trellis codes for use on a given DMC, wherein each code is assigned a probability in such a way that the ensemble is pairwise independent as defined above for some given probability distribution $Q(x)$ over the channel input alphabet, then, regardless of the actual probability distribution for the information bits, the average over these codes of the bit error probability for ML decoding satisfies

$$\mathrm{E}[P_\mathrm{b}] < c(R_\mathrm{t}, \rho, Q) 2^{-N_t E_0(\rho, Q)}, \qquad 0 \le \rho \le 1 \tag{6.64}$$

provided that $R_\mathrm{t} = k_0/n_0$ satisfies

$$R_\mathrm{t} < \frac{E_0(\rho, Q)}{\rho}, \tag{6.65}$$

where $c(R_\mathrm{t}, \rho, Q)$ and N_t are defined by (6.62) and (6.53), respectively, and where $E_0(\rho, Q)$ is Gallager's function (5.93).

Naturally, we are interested in the best exponent that we can get in (6.60) for a given trellis code rate R_t. Because $E_0(\rho, Q)$ increases with ρ, we want to choose ρ as large as possible consistent with (6.65). The problem, however, is that the inequality (6.65) is strict so that we cannot take ρ to satisfy $R_\mathrm{t} = E_0(\rho, Q)/\rho$ as we would like. To avoid this problem, we pick a small positive number ε and replace (6.65) by the stronger condition

$$R_\mathrm{t} \le \frac{E_0(\rho, Q) - \varepsilon}{\rho}. \tag{6.66}$$

One happy consequence of (6.66) is that it simplifies (6.62) to

$$c(R_\mathrm{t}, \rho, Q) \le \frac{2^{k_0}}{(1 - 2^{\varepsilon n_0})^2} \tag{6.67}$$

in which the upper bound has no dependence on ρ or Q. Because R_0 is the maximum of $E_0(\rho, Q)$ over Q and ρ ($0 \le \rho \le 1$), we see that the best exponent in (6.64) equals R_0 over the range where (6.66) can be satisfied, i.e., over $R_\mathrm{t} \le R_0 - \varepsilon$. Denoting this best trellis code error exponent by $E_t(R, \varepsilon)$, we have then

$$E_t(R_\mathrm{t}, \varepsilon) = R_0 \qquad \text{for } R_\mathrm{t} \le R_0 - \varepsilon. \tag{6.68}$$

For higher rates, we can specify $E_t(R, \varepsilon)$ parametrically by

$$E_t(R_\mathrm{t}, \varepsilon) = \max_Q E_0(\rho^*, Q), \qquad R_0 - \varepsilon < R_\mathrm{t} \le C - \varepsilon \tag{6.69}$$

where ρ^*, which depends on Q, is the solution of

$$\frac{E_0(\rho^*, Q) - \varepsilon}{\rho^*} = R_\mathrm{t}, \tag{6.70}$$

and where

$$C = \lim_{\rho \to 0} \frac{E_0(\rho)}{\rho}$$

is the capacity of the DMC (see Problem 5.12). Thus, we have already proved most of the following result.

Viterbi's Coding Theorem for Trellis Codes: For any positive number ε, any assignment of probabilities to the information bits, any trellis code rate $R_\mathrm{t} = \frac{k_0}{n_0}$, any trellis code constraint length $N_t = (T+1)n_0$, and any trellis length L_t (including $L_\mathrm{t} = \infty$), there exist such trellis codes for use on a DMC such that their bit error probability for ML decoding satisfies

$$P_\mathrm{b} < c_v(\varepsilon) 2^{-N_t E_t(R_\mathrm{t}, \varepsilon)} \tag{6.71}$$

where $E_t(R_\mathrm{t}, \varepsilon)$, which is given by (6.68),(6.69) and (6.70), is positive for $R_\mathrm{t} \le C - \varepsilon$ (where C is the capacity of the DMC) and where

$$c_v(\varepsilon) = \frac{2^{k_0}}{(1 - 2^{-\varepsilon n_0})^2}. \tag{6.72}$$

The only unproved part of this theorem is the claim that $E_t(R_\mathrm{t}, \varepsilon) > 0$ for $R_\mathrm{t} \le C - \varepsilon$, which we leave as an exercise.

Since we may choose ε in the above theorem as small as we like, we see that the "true" best exponent that can be achieved at rate R_t is

$$E_\mathrm{V}(R_\mathrm{t}) = \lim_{\varepsilon \to 0} E_t(R_\mathrm{t}, \varepsilon), \tag{6.73}$$

which we call the *Viterbi exponent*. Equivalently to (6.73), we can define the Viterbi exponent by

$$E_\mathrm{V}(R_\mathrm{t}) = R_0, \qquad \text{for } R_\mathrm{t} \le R_0 \tag{6.74}$$

and

$$E_\mathrm{V}(R_\mathrm{t}) = \max_Q E_0(\rho^*, Q), \qquad \text{for } R_0 < R_\mathrm{t} < C \tag{6.75}$$

where ρ^*, which depends on Q, is the solution of

$$\frac{E_0(\rho^*, Q)}{\rho^*} = R_\mathrm{t}. \tag{6.76}$$

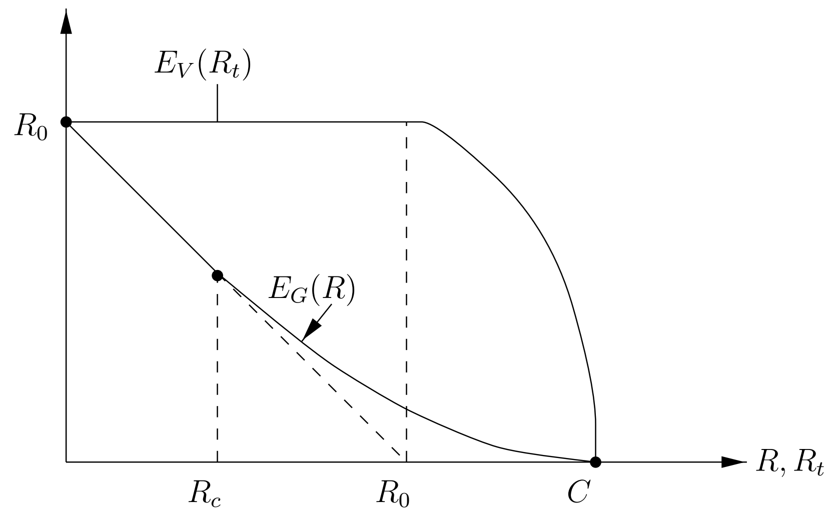


Figure 6.11: The general form of the Viterbi exponent $E_{\mathrm{V}}(R_{\mathrm{t}})$ and the Gallager exponent $E_{\mathrm{G}}(R)$.

In Figure 6.11, we give a sketch of the general form of the Viterbi exponent $E_{\mathrm{V}}(R_{\mathrm{t}})$, together with the Gallager exponent $E_{\mathrm{G}}(R)$ for block codes. The striking feature is that

$$E_{\mathrm{V}}(R_{\mathrm{t}}) > E_{\mathrm{G}}(R), \qquad \text{for } 0 \leq R_{\mathrm{t}} = R < C \tag{6.77}$$

where the inequality is great at high rates. This indicates that the constraint length N_t required with a good trellis code to get some desired ML decoding error probability P_{b} will be much smaller than the block length N of a good block code of the same rate. This suggests a natural superiority of the "non-block" trellis codes over block codes, but only if the complexity of ML decoding is comparable when $N_t = N$ and $R_{\mathrm{t}} = R$ so that our comparison of error exponents is a fair one. But ML decoding of the trellis code requires a Viterbi decoder with $2^{k_0 T} = 2^{n_0 R_{\mathrm{t}} T} \approx 2^{N_t R_{\mathrm{t}}}$ states. For a general block code, there seems to be no better way to do ML decoding than to compute $P(\underline{y}|\underline{x})$ for all $M = 2^{NR}$ codewords $\underline{x}$, then choose that $\underline{x}$ which gave the largest. But these two ways of performing ML decoding are indeed of roughly the same complexity so that the comparison made above is fair. But might not there be block codes much easier to decode than by the general method described above? Indeed there are! Remember that an $(L_{\mathrm{t}}, T, n_0, k_0)$ trellis code is also a block code (unless $L_{\mathrm{t}} = \infty$) with $N = (L_{\mathrm{t}} + T)n_0$ and $R \approx R_{\mathrm{t}}$ for $L_{\mathrm{t}} \gg T$, but these block codes can be ML decoded by a decoder whose complexity is of the order $2^{N_t R_{\mathrm{t}}} = 2^{(T+1)n_0 R_{\mathrm{t}}} \ll 2^{NR}$. The trellis codes themselves are the subclass of block codes for which ML decoding is apparently simplest!

There is a simple geometric method to find the function $E_{\mathrm{V}}(R_{\mathrm{t}})$ from $E_{\mathrm{G}}(R)$, or vice-versa (see Problem 6.9).

This is also the appropriate place to mention that the form of $E_{\mathrm{V}}(R)$ as in Figure 6.11 gives even greater credence to the claim that R_0 is the best single parameter description of a DMC. Not only does R_0 specify a range of rates where $E_{\mathrm{V}}(R_{\mathrm{t}})$ is positive, namely $R_{\mathrm{t}} < R_0$, but it also shows that we can achieve an error exponent equal to R_0 everywhere in this rate region, if we have the good sense to opt for trellis codes rather than block codes.