2.7 Perfectly Lossless Fixed-Length to Variable-Length Block Source Codes

To obtain lower rates than perfectly lossless FFB source codes, in Sections 2.5 and 2.6 we relaxed the perfectly lossless requirement and considered almost lossless FFB source codes. In this section we maintain the perfectly lossless requirement, but relax the FFB requirement — allowing the codewords to have different lengths — again with the goal of obtaining lower rates. Specifically, we consider *fixed-length to variable-length block* (FVB) codes, which are similar to a *fixed-length to fixed-length block* (FFB) codes except that the codebook C contains codewords of varying lengths. Although the varying length nature of the codewords complicates the encoding and decoding somewhat, it turns out that perfectly lossless FVB codes with a given source length can perform as well as almost lossless FFB codes with much larger source lengths. And this ordinarily translates into much lower complexity and implementation cost.

Example 2.7.1 *Consider the encoding table shown below for an IID source* $\widetilde{U}$ *with alphabet* $A_U = \{a, b, c\}$*, probabilities* $p_a = 1/2$*,* $p_b = 1/4$*,* $p_c = 1/4$*, and entropy* $H = 1.5$*.*

u	$\underline{z} = f_e(u)$
a	*0*
b	*1 0*
c	*1 1*

For example with this encoding table, the source sequence $\underline{U} = aabcbac$ *is encoded into* $\underline{z} = 00101110011$*. It is easy to see that after encoding any source sequence, the bits produced by this encoding table can be decoded into the original source sequence; i.e., the code is perfectly lossless. It is also easy to see that on the average this code produces* 1.5 *bits per source symbol, which is its rate and which equals the entropy of the source. In comparison the best perfectly lossless FFB codes with source length one have rate* $\lceil \log_2 3 \rceil = 2$*, and the best perfectly lossless FFB codes with any source length have rates approaching* $\log_2 3 = 1.58$ *bits per source symbol. Although almost lossless FFB codes can attain rate arbitrarily close to the entropy, which is 1.5 bits per source symbol, they require a large source length and, consequently, a much larger codebook and much larger implementation complexity.* □

In general, a perfectly lossless FVB code is characterized by its source length K, its codebook $C = \{ \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_{Q^K} \}$, where the i^{th} codeword $\underline{v}_i =$

$(v_{i1}, v_{i2}, \ldots, v_{iL_i})$ is a binary sequence with length denoted L_i, its encoding rule f_e assigning codewords in C to source sequences of length K, and its decoding rule f_d assigning source sequences of length K to codewords. As with an FFB code, the encoder operates in "block fashion". It applies f_e to the first block, $\underline{U}_1 = (U_1, \ldots, U_K)$, produces a binary sequence denoted $\underline{Z}_1 = f_e(U_1, \ldots, U_K)$, then applies f_e to the next block, $\underline{U}_2 = (U_{K+1}, \ldots, U_{2K})$, produces the binary sequence $\underline{Z}_2 = f_e(U_{K+1}, \ldots, U_{2K})$, and so on.

Although the code is considered a "block" code, the decoder does not operate in the usual block fashion. For simplicity and, as it turns out, without loss of potential performance, we will assume that the codebook C has the *prefix-free* property that none of its codewords is the prefix of another. (A sequence $\underline{v} = (v_1, \ldots, v_m)$ is called a *prefix* of another sequence $\underline{w} = (w_1, \ldots, w_n)$ if $n \geq m$ and $w_i = v_i$, for $i = 1, \ldots, m$.) From now on we shall refer to C as a *prefix codebook* and to the resulting code as a *prefix code*.

The decoder of a prefix code operates as follows: Given an encoded sequence $\underline{Z}$, it begins by looking for the first codeword to appear in $\underline{Z}$. That is, it looks to see if Z_1 is a codeword, and if not it looks to see if Z_1, Z_2 is a codeword, and if not it looks to see if Z_1, Z_2, Z_3 is a codeword, and so forth. Eventually, it finds an integer J_1 such that $Z_1, \ldots, Z_{J_1}$ is a codeword in C. It then applies the decoding rule f_d, produces the reproduction $(\widehat{U}_1, \ldots, \widehat{U}_K) = f_d(Z_1, \ldots, Z_{J_1})$ and presents it to the user. Next the decoder examines the remainder of $\underline{Z}$, namely $Z_{J_1+1}, Z_{J_1+2}, \ldots$, until it finds a codeword, say $(Z_{J_1+1}, Z_{J_1+2}, \ldots, Z_{J_2})$. It then applies f_d and presents $(\widehat{U}_{K+1}, \ldots, \widehat{U}_{2K}) = f_d(Z_{J_1+1}, \ldots, Z_{J_2})$ to the user. Subsequent blocks of $\underline{\widehat{U}}$ are produced in the same fashion. The purpose of the prefix property is to insure that when the decoder discovers a codeword in $\underline{Z}$ it may immediately decode these bits, for it knows that they could not be the beginning of some longer codeword.

It is helpful to visualize the decoding with the aid of a binary tree. For example, see Figure 2.7.1. Upward branches of the tree are identified with 0's and downward branches with 1's. Each codeword, $\underline{v} = (v_1, \ldots, v_L)$, indicates a path through the tree from left to right, with the i^{th} component v_i indicating whether the ith branch of the path is up or down. As a result, each codeword is associated with the node reached at the end of its path. Once this tree is specified, one may envision the decoding process as using the bits $Z_1, Z_2, \ldots$ to generate a path through the tree. When a node associated with some codeword is reached, one has found $Z_1, \ldots, Z_{J_1}$ and applies the decoding rule f_d. One then returns to the root node of the tree and uses the remaining bits $Z_{J_1+1}, Z_{J_1+2}, Z_{J_1+3}, \ldots$ to generate a path through the tree to the next codeword, and so on.

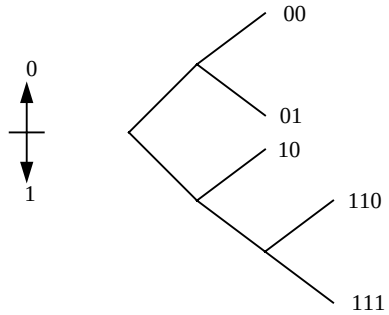


Figure 2.7.1: Tree diagram of the prefix code $C = \{00, 01, 10, 110, 111\}$.

A prefix code is perfectly lossless if and only if the encoding rule f_e is a one-to-one function, i.e. it assigns distinct codewords to distinct source sequences, and the decoding rule f_d is the inverse of f_e. The rate of such a code is the average codeword length divided by K; that is,

$$\overline{R} = \frac{\overline{L}}{K} = \frac{1}{K} \sum_{u^K} p(u^K) L(u^K) \ , \tag{2.7.1}$$

where $L(u^K)$ denotes the length of the codeword $f_e(u^K)$ assigned to u^K, and $p(u^K)$, as usual, denotes the probability of the source sequence u^K.

The principal goal of this section is to find

$$r^*_{VL,K} \stackrel{\Delta}{=} \min \left\{ r : \begin{array}{l} \text{there is a perfectly lossless FVB pre-} \\ \text{fix code with source length } K \text{ and} \\ \text{rate } r \end{array} \right\} \ , \tag{2.7.2}$$

which is the least rate of any FVB prefix code with source length K, and

$$r^*_{VL} \stackrel{\Delta}{=} \inf \left\{ r : \begin{array}{l} \text{there is a perfectly lossless FVB pre-} \\ \text{fix code (with any source length) and} \\ \text{rate } r \end{array} \right\} \tag{2.7.3}$$

$$= \inf \left\{ r^*_{VL,K} : K = 1, 2, \ldots \right\} \ , \tag{2.7.4}$$

which is the least rate of any perfectly lossless FVB code of any blocklength. We will also answer the following:

Question 2.7.1 *How does one design an FVB prefix code?*

The idea, of course, is to assign shorter codewords to source sequences with higher probability even if it means assigning longer codewords to source sequences with smaller probability. But how short and low long?

We first consider the simplest case wherein the source length K is 1 and, consequently, the code rate is the average length. It turns out that the key strategy for designing low rate prefix codes with source length 1 is to choose the code so that

$$L_q \cong -\log_2 p_q, \tag{2.7.5}$$

where p_q and L_q are shorthand for $p(a_q)$ and $L(a_q)$, respectively. To see the benefit of such a choice let us compute the average length:

$$\overline{L} \;=\; \sum_{q=1}^{Q} p_q L_q \;\cong\; -\sum_{q=1}^{Q} p_q \log_2 p_q = H. \tag{2.7.6}$$

Thus, the average length, and consequently the rate, is approximately equal to the entropy of the source. The result of the previous section suggests that this is very good and maybe even optimal performance. But two questions remain:

Question 2.7.2 *Does there actually exist a prefix code with lengths* $L_q \cong -\log_2 p_q$*?*

Question 2.7.3 *Could there be prefix codes with even smaller rates?*

Both of these questions may be answered using the following.

Theorem 2.7.1 (The Kraft inequality theorem) *There exists a binary prefix code with lengths* $\{L_1, L_2, \ldots, L_Q\}$ *if and only if*

$$\sum_{q=1}^{Q} 2^{-L_q} \leq 1. \tag{2.7.7}$$

That is, if the "Kraft inequality" holds for $\{L_1, \ldots, L_Q\}$*, then there exists a prefix code having these lengths. Conversely, the lengths of any prefix code satisfy the Kraft inequality.*

Proof

Let us first show that if $\{\underline{v}_1, \ldots, \underline{v}_Q\}$ is a prefix code with lengths $\{L_1, \ldots, L_Q\}$, then $\sum_{q=1}^{Q} 2^{-L_q} \leq 1$. Let L_{max} denote the length of the longest codeword. We proceed by counting the number of binary sequences of length L_{max} that are prefixed by one codeword or another, and by comparing this number to $2^{L_{max}}$, the total number of binary sequences of length L_{max}. Specifically, the q-th codeword $\underline{v}_q$ is a prefix of $2^{L_{max}-L_q}$ binary sequences of length L_{max}. Since the code has the prefix-free property, no

sequence of length L_{max} is prefixed by more than one codeword. Hence, the total number of sequences prefixed by some codeword is $\sum_{q=1}^{Q} 2^{L_{max}-L_q}$ and since this can be no larger than $2^{L_{max}}$, we have (after multiplying by $2^{-L_{max}}$)

$$\sum_{q=1}^{Q} 2^{-L_q} \leq 1 \ , \tag{2.7.8}$$

which is the Kraft inequality.

Now suppose that $\{L_1, \ldots, L_Q\}$ are a set of lengths satisfying the Kraft inequality. We will show there is a prefix code $\{\underline{v}_1, \ldots, \underline{v}_Q\}$ with these lengths. Let us assume for convenience that the lengths are arranged in increasing order, and let us begin by choosing $\underline{v}_1$ to be any binary sequence of length L_1. Next choose $\underline{v}_2$ to be any binary sequence of length L_2 that is not prefixed by $\underline{v}_1$, choose $\underline{v}_3$ to be any binary sequence of length L_3 that is not prefixed by $\underline{v}_1$ or $\underline{v}_2$, and so on. To demonstrate that this procedure will always work, we will show, using the Kraft inequality, that if after the n^{th} stage ($n < Q$) we have been able to choose codewords $\{\underline{v}_1, \ldots, \underline{v}_n\}$ so as to have lengths $\{L_1, \ldots, L_n\}$ and so that no codeword is the prefix of another, then there is at least one binary sequence of length L_{n+1} that is not prefixed by any of the codewords chosen so far, and this sequence can be chosen as $\underline{v}_{n+1}$.

For any $q, 1 \leq q \leq n$, there are $2^{L_{n+1}-L_q}$ binary sequences of length L_{n+1} that are prefixed by $\underline{v}_q$. Hence, the number of binary sequences of length L_{n+1} that cannot be selected as $\underline{v}_{n+1}$ is $\sum_{q=1}^{n} 2^{L_{n+1}-L_q}$. Is there one left that can be selected? The Kraft inequality shows

$$\sum_{q=1}^{n} 2^{L_{n+1}-L_q} = 2^{L_{n+1}} \sum_{q=1}^{n} 2^{-L_q} < 2^{L_{n+1}} \sum_{q=1}^{Q} 2^{-L_q} \leq 2^{L_{n+1}}; \tag{2.7.9}$$

i.e., the number of binary sequences of length L_{n+1} prefixed by codewords is strictly less than the total number of sequences of length L_{n+1}. Therefore, at least one such sequence remains that can be selected as $\underline{v}_{n+1}$. □

Let us now use the Kraft Inequality Theorem to answer Question 2.7.2 by showing there are prefix codes with lengths $L_q \cong -\log_2 p_q$. Since $-\log_2 p_q$ need not be an integer, let us choose

$$L_q = \lceil -\log_2 p_q \rceil \ , \ q = 1, \ldots, Q \ . \tag{2.7.10}$$

To see that there is indeed a prefix code with these lengths, we need only check that they satisfy the Kraft inequality (2.7.7). Using the fact that

$$\lceil -\log_2 p_q \rceil \geq -\log_2 p_q, \tag{2.7.11}$$

we find

$$\sum_{q=1}^{Q} 2^{-L_q} \;=\; \sum_{q=1}^{Q} 2^{-\lceil -\log_2 p_q \rceil} \;\leq\; \sum_{q=1}^{Q} 2^{\log_2 p_q} \;=\; \sum_{q=1}^{Q} p_q = 1 \;, \qquad (2.7.12)$$

which demonstrates that the Kraft inequality holds. Therefore, there does indeed exist a prefix code with lengths $L_q = \lceil -\log_2 p_q \rceil$, and this answers Question 2.7.2. One may find such a code simply by following the brute force procedure described in the second half of the proof of the Kraft inequality theorem. That is, choose $\underline{v}_1$ to be any binary sequence of length L_1, choose $\underline{v}_2$ be any binary sequence of length L_2 not prefixed by $\underline{v}$, and so on. The resulting codes are called *Shannon-Fano codes*.

We can now carefully bound the average length of the resulting code. Using the inequality,

$$\lceil -\log_2 p_q \rceil < -\log_2 \; p_q + 1, \qquad (2.7.13)$$

we find

$$\begin{aligned} \overline{L} \;&=\; \sum_{q=1}^{Q} p_q L_q \;=\; \sum_{q=1}^{Q} p_q \lceil -\log_2 p_q \rceil \;<\; -\sum_{q=1}^{Q} p_q \log_2 p_q + \sum_{q=1}^{Q} p_q \\ &=\; H + 1 \;. \end{aligned} \qquad (2.7.14)$$

Similarly, using the inequality

$$\lceil -\log_2 p_q \rceil \geq -\log_2 p_q \;, \qquad (2.7.15)$$

we find

$$\begin{aligned} \overline{L} \;&=\; \sum_{q=1}^{Q} p_q L_q \;=\; \sum_{q=1}^{Q} p_q \lceil -\log_2 p_q \rceil \;\geq\; \sum_{q=1}^{Q} -p_q \log_2 p_q \\ &=\; H \;. \end{aligned} \qquad (2.7.16)$$

Thus the average length $\overline{L}$ of a prefix code with lengths $L_q = \lceil -\log_2 p_q \rceil$ satisfies

$$H \leq \overline{L} < H + 1 \;. \qquad (2.7.17)$$

We now answer Question 2.7.3 by showing that no prefix code with source length 1 can have average length smaller than H. To do this we make use of the elementary inequality

$$\ln x \leq x - 1 \;, \qquad (2.7.18)$$

($\ln x$ denotes the natural logarithm of x), which is illustrated in Figure 2.7.2 and which is the basis of many important inequalities in information theory.

Let $\{L_1, \ldots, L_Q\}$ be the lengths of any prefix code whatsoever. To show that $\overline{L}$ must be larger than H, consider their difference. We find

$$\begin{aligned}
\overline{L} - H &= \sum_{q=1}^{Q} p_q L_q + \sum_{q=1}^{Q} p_q \log_2 p_q \\
&= -\sum_{q=1}^{Q} p_q \log_2 \left(\frac{2^{-L_q}}{p_q} \right) \quad = \quad -\sum_{q=1}^{Q} p_q \ln \left(\frac{2^{-L_q}}{p_q} \right) \frac{1}{\ln 2} \\
&\geq -\sum_{q=1}^{Q} p_q \left(\frac{2^{-L_q}}{p_q} - 1 \right) \frac{1}{\ln 2} \quad = \quad -\left(\sum_{q=1}^{Q} 2^{-L_q} - 1 \right) \frac{1}{\ln 2} \\
&\geq 0 \ , \qquad (2.7.19)
\end{aligned}$$

where the last inequality employed the Kraft inequality. This shows that $\overline{L} \geq H$ for any prefix code with source length 1.

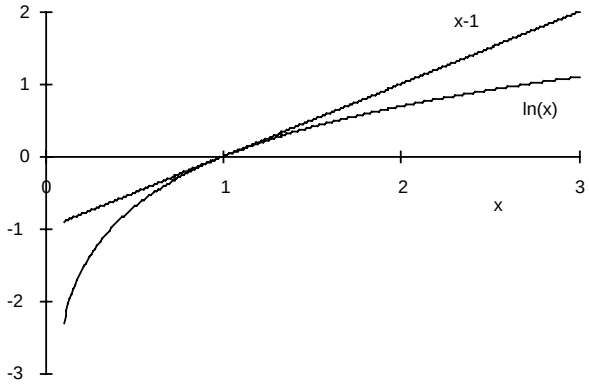


Figure 2.7.2: $\ln(x)$ and $x - 1$.

The following summarizes what we have learned so far about prefix codes with source length 1.

Lemma 2.7.2 *Given a set of probabilities* $\{p_1, p_2, \ldots, p_Q\}$*:*

(a) There exists a prefix code with lengths $\{L_1, L_2, \ldots, L_Q\}$ *such that*

$$\overline{L} < H + 1 \ , \qquad (2.7.20)$$

(b) For any prefix code whatsoever

$$\overline{L} \geq H \ . \qquad (2.7.21)$$

Equivalently, letting $\overline{L}^*$ denote the least average length of any prefix code, then

$$H \le \overline{L}^* < H + 1 \ . \tag{2.7.22}$$

Exercise 2.7.1 *Find an example of a source for which the Shannon-Fano code does not have the smallest possible average length; i.e. its average length is greater than $\overline{L}^*$. Hint: one need only consider a binary source.* □

Let us now turn our attention to prefix codes with source lengths K greater than 1. Since the rate of such a code is proportional to its average length, it has minimal rate if and only if it has minimal average length. So we need only apply what we have just learned, except that here we need a codeword for each source sequence of length K (i.e., Q^K codewords, one for each $u^K \in A_U^K$), and the relevant set of probabilities is $\left\{p(u^K) : u^K \in A_U^K\right\}$. We conclude that the codeword for u^K should have length approximately equal to $-\log_2 p(u^K)$. Specifically, there exists a prefix code with lengths $L(u^K) = \lceil -\log_2 p(u^K) \rceil$; this code has

$$H^K \le \overline{L} < H^K + 1 \ ; \tag{2.7.23}$$

and every prefix code with source length K has

$$\overline{L} \ge H^K \ , \tag{2.7.24}$$

where H^K denotes the entropy of the random vector $U^K = (U_1, \ldots, U_K)$,

$$H^K \stackrel{\Delta}{=} -\sum_{u^K \in A_U^K} p(u^K) \log_2 p(u^K) \ . \tag{2.7.25}$$

Using the IID nature of the source, we find that H^K simplifies:

$$\begin{aligned} H^K &= -\sum_{u^K \in A_U^K} p(u^K) \log_2 \prod_{k=1}^{K} p(u_k) &&= -\sum_{u^K \in A_U^K} p(u^K) \sum_{k=1}^{K} \log_2 p(u_k) \\ &= -\sum_{k=1}^{K} \sum_{u^K \in A_U^K} p(u^K) \log_2 p(u_k) &&= -\sum_{k=1}^{K} \sum_{u_k \in A_U} p(u_k) \log_2 p(u_k) \\ &= KH \ . \end{aligned} \tag{2.7.26}$$

Thus the least average length of prefix codes with source length K, henceforth denoted L_K^*, satisfies

$$KH \le L_K^* < KH + 1 \ . \tag{2.7.27}$$

As a consequence, the least rate, $R_{VL,K}^* = L_K^*/K$, is between H and $H + 1/K$. In effect, larger source lengths enable us to reduce the 1 in equation

(2.7.20) to $1/K$, which is especially important when H is small. In addition, we easily see that $R^*_{VL} \stackrel{\Delta}{=} \inf\left\{R^*_{VL,K} : K = 1, 2, \ldots\right\} = H$. We summarize in the following.

Theorem 2.7.3 (Coding Theorem for FVB Prefix Codes) *Let U be an IID source with finite alphabet and entropy H.*

(a) Positive statements:

$$R^*_{VL,K} < H + \frac{1}{K} \text{ , for every positive integer } K \text{ ;} \tag{2.7.28}$$

i.e. for every K there is an FVB prefix code with source length K and rate $R < H + \frac{1}{K}$; and

$$R^*_{VL} \leq H \text{ ;} \tag{2.7.29}$$

i.e. for every $\epsilon > 0$ there is an FVB prefix code with rate $R \leq H + \epsilon$.

(b) Converse Statement:

$$R^*_{VL,K} \geq R^*_{VL} \geq H \text{ , for every positive integer } K \text{ ;} \tag{2.7.30}$$

i.e. every prefix code (with any source length whatsoever) has rate $R \geq H$.

(c) Combined Statements: For any positive integer K,

$$H \leq R^*_{VL,K} < H + \frac{1}{K} \text{ ,} \tag{2.7.31}$$

and

$$R^*_{VL} = H \text{ .} \tag{2.7.32}$$

Exercise 2.7.2 *(a) Show that $R^*_{VL,K} \geq R^*_{VL,MK}$ for any positive integers M and K. (Hint: Consider a code with source length MK whose codebook consists of all possible concatenations of M codewords from the codebook of an optimal code with source length K.) (b) Find an example of a source for which $R^*_{VL,K+1} < R^*_{VL,K}$ for some K. (c) (Difficult) Find another example for which $R^*_{VL,K+1} > R^*_{VL,K}$.* □

Huffman's code design algorithm

Our final task is to answer Question 2.7.1, namely: How does one design prefix codes with the least possible average length and rate? The point of Exercise 2.7.1 was to show that the Shannon-Fano code does not always give the least average length. Optimal codes, i.e those with smallest rate, are

found by Huffman's algorithm, which we will now describe. The resulting codes are often called *Huffman codes*.

Given a set of probabilities $P_Q = \{p_1, \ldots, p_Q\}$, we must find an optimum codebook $C_Q = \{\underline{v}_1, \ldots, \underline{v}_Q\}$; i.e., one with $\overline{L}_Q = \sum_{q=1}^{Q} p_q L_q$ as small as possible. (Here, it helps to subscript C and $\overline{L}$ with the number of source symbols Q). The basic idea of Huffman's algorithm is that an optimum codebook can be formed by a simple "extension" of an optimum codebook C_{Q-1} for the "reduced" set of probabilities $P_{Q-1} = \{p'_1, \ldots, p'_{Q-1}\}$, where the p'_q's are the same as the p_q's except that the two smallest p_q's in P_Q have been added to form one of the p'_q's. It simplifies notation to assume $p_1 \geq p_2 \geq \ldots \geq p_Q$. Then $p'_1 = p_1$, $p'_2 = p_2$, ..., $p'_{Q-2} = p_{Q-2}$, $p'_{Q-1} = p_{Q-1} + p_Q$. The key observation, to be proved later, is:

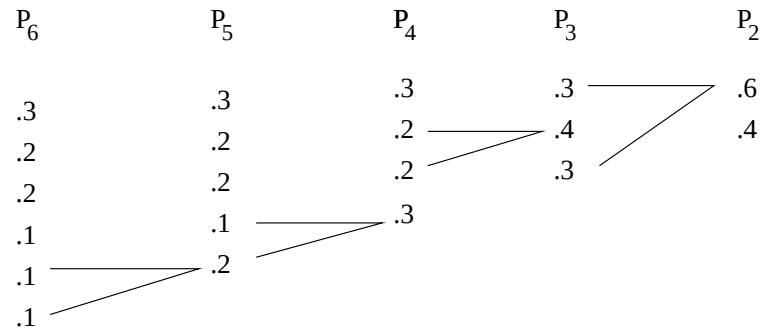
> If $C_{Q-1} = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-1}\}$ is an optimum codebook for P_{Q-1}, then $C_Q = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-2}, \underline{v}'_{Q-1}0, \underline{v}'_{Q-1}1\}$ is an optimum codebook for P_Q.

That is, an optimum code for P_Q is obtained by taking an optimum code C_{Q-1} for P_{Q-1}, using the first $Q-2$ codewords as they are, and "extending" the $(Q-1)^{th}$ codeword by adding "0" to obtain the codeword $\underline{v}_{Q-1} = (\underline{v}'_{Q-1}0)$ and then adding a "1" to obtain the codeword $\underline{v}_Q = (\underline{v}'_{Q-1}1)$.

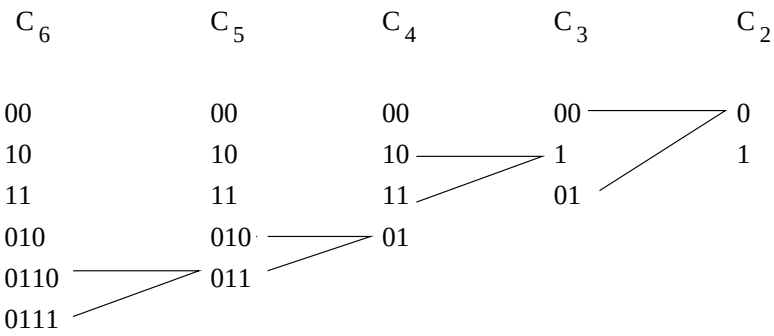
Next, an optimum codebook for P_{Q-1} can be constructed by extending an optimum codebook C_{Q-2} for the reduced set P_{Q-2}, formed by adding the two smallest probabilities in P_{Q-1}. We continue to reduce the set of probabilities in this way, until we need only find an optimum codebook for a set P_2 containing just two probabilities.

We now work our way backwards. An optimum codebook for the set P_2 is, obviously, $C_2 = \{0, 1\}$. An optimum codebook C_3 for P_3 (with three probabilities) is obtained by appending both 0 and 1 to the codeword in C_2 associated with the probability in P_2 that is the sum of the two smallest probabilities in P_3. An optimum codebook C_4 for P_4 is obtained by appending both 0 and 1 to the codeword in C_3 associated with the element of P_3 that is the sum of the two smallest elements of P_4, and so on until we find an optimum codebook C_Q for the original set of probabilities P_Q.

The process of reducing a set of probabilities and then expanding the codebooks is illustrated in Figure 2.7.3. Notice that at various stages there are three or more smallest probabilities, from which we arbitrarily choose to combine two. Consequently, the Huffman algorithm may be used to generate a number of optimum codebooks, even having different sets of lengths (see Exercise 2.7.6). Of course they all have the same average length, for otherwise they would not all be optimum.



(a) Reducing the sets of probabilities.



(b) Expanding the set of codewords.

Figure 2.7.3: Huffman design procedure.

It remains only to prove the key observation. Accordingly, let $C_{Q-1} = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-1}\}$ be an optimum codebook for P_{Q-1}, and let $C_Q = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-2}, \underline{v}'_{Q-1}0, \underline{v}'_{Q-1}1\}$ be a codebook for P_Q. The average length of C_Q is related to that of C_{Q-1} via

$$\begin{aligned}
\overline{L}_Q &= \sum_{q=1}^{Q} p_q L_q = \sum_{q=1}^{Q-2} p_q L'_q + p_{Q-1}(L'_{Q-1}+1) + p_Q(L'_{Q-1}+1) \\
&= \sum_{q=1}^{Q-1} p'_q L'_q + (p_{Q-1}+p_Q) \\
&= \overline{L}_{Q-1} + (p_{Q-1}+p_Q) \ . \qquad (2.7.33)
\end{aligned}$$

We will now use proof by contradiction. Suppose C_Q were not optimum for P_Q. Then an optimum code $C^*_Q = \{\underline{v}^*_1, \ldots, \underline{v}^*_Q\}$ for P_Q will have average

length $\overline{L}_Q^* < \overline{L}_Q$. Moreover, Exercise 2.7.7, below, shows that C_Q^* can be chosen so that the codewords associated with p_Q and p_{Q-1} are siblings in the sense of having the same length and differing only in the last bit. From C_Q^* we may in turn create a code $C_{Q-1}^* = \{\underline{v}_1^*, \ldots, \underline{v}_{Q-2}^*, \underline{v}'_{Q-1}\}$ for P_{Q-1}, where $\underline{v}'_{Q-1}$ is obtained by stripping the last bit from $\underline{v}_{Q-1}^*$ (or for that matter, from its sibling $\underline{v}_Q^*$). Notice that C_Q^* is, in fact, the direct extension of C_{Q-1}^*. Therefore using (2.7.33), the average length of C_{Q-1}^* is

$$\begin{aligned} \overline{L}_{Q-1}^* &= \overline{L}_Q^* - p_{Q-1} - p_Q \;<\; \overline{L}_Q - p_{Q-1} - p_Q \\ &= \overline{L}_{Q-1} \;, \end{aligned} \tag{2.7.34}$$

which contradicts the fact that C_{Q-1} is optimum for P_{Q-1}. Hence, our assumption that C_Q is not optimum must be false; i.e., C_Q is indeed optimum.

letter	prob.	codewd.	len.	letter	prob.	codewd.	len.
Space	.1859	1000	3	F	.0208	001100	6
E	.1031	100	3	M	.0198	001101	6
T	.0796	0010	4	W	.0175	001110	6
A	.0642	0100	4	Y	.0164	011100	6
O	.0632	0110	4	G	.0152	011101	6
I	.0575	1010	4	P	.0152	011110	6
N	.0574	1011	4	B	.0127	011111	6
S	.0514	1100	4	V	.0083	0011110	7
R	.0484	1101	4	K	.0049	00111110	8
H	.0467	1110	4	X	.0013	001111110	9
L	.0321	01010	5	J	.0008	0011111110	10
D	.0317	01011	5	Q	.0008	00111111110	11
U	.0228	11110	5	Z	.0005	00111111111	11
C	.0218	11111	5				

Figure 2.7.4: Huffman code for English

Example 2.7.2 *An optimal code (source length $K = 1$) for the probabilities of English letters given in Figure 2.5.4 is shown in Figure 2.7.4. Its rate is 4.12 bits per symbol which compares to the entropy of 4.08.* □

Exercise 2.7.3 *Find a set of probabilities $\{p_1, \ldots, p_Q\}$ for which there does not exist a prefix code with lengths $L_q = \lfloor -\log_2 p_q \rfloor$, $q = 1, \ldots, Q$. This explains why we conservatively rounded up rather than down.* □

Exercise 2.7.4 *Show that the minimum average length among prefix codes equals H, exactly, when and only when all p_q's are powers of 2.* □

Exercise 2.7.5 *Find an integer Q and a set of probabilities $\{p_1, \ldots, p_Q\}$ for which the minimum average length of prefix codes is at least .9 bits larger than H.* □

Exercise 2.7.6 *An IID source U has alphabet $A_U = \{a, b, c, d, e\}$ and probabilities $\{.4, .2, .2, .1, .1\}$. (a) Find two prefix codes with source length 1 whose average lengths are minimum and whose sets of lengths are different. (b) For each code compute the average and variance of its lengths. (c) Can you think of a reason why a code with smaller variance would be useful? (Hint: See the discussion below on buffering.) (d) Find the smallest source length K for which there exists a prefix code with rate $R \leq H + .1$.* □

Exercise 2.7.7 *Show there exists an optimum codebook C_Q for the set of probabilities $P_Q = \{p_1, \ldots, p_Q\}$ such that the codewords associated with the two smallest probabilities are siblings in the sense of having the same length and differing only in the last bit. Hint: First show that the longest codeword in any optimum codebook has another codeword as a sibling.* □

Exercise 2.7.8 *(From McEliece, Problem 10.22) Consider the game of "twenty questions" in which you are required to determine the outcome of one roll of a pair of dice by asking questions that can be answered "yes" or "no". The outcome to be guessed is one of the integers $2, 3, 4, \ldots, 12$. A question takes the form "Is $D \in S$?" where D is the outcome of the dice and S is a subset of the integers $\{2, 3, \ldots, 12\}$. The choice of a question, i.e the choice of S, may depend on the answers to the previous questions, and the number of questions until the outcome is determined need not be the same for all outcomes. Find a questioning strategy that, on the average, requires the fewest number of questions.*

Hints: (1) If you asked "Is it 2?" , "Is it 3?" etc., you would average a little under six questions. It is possible to do better, however. (2) Given an algorithm for questioning, the sequence of yes/no answers you get for a given value D might be considered a binary codeword for D. (3) What is the probability of a given value of D? □

Exercise 2.7.9 *A binary IID source U with alphabet $A_U = \{0, 1\}$ and $p_0 = .9$.*
(a) Find the smallest possible rate of any FVB lossless source code?
(b) Find a fixed-to-variable length block prefix code with rate .55 or less. Make it as simple and good as possible. And compute the rate of the code.
□

Exercise 2.7.10 *An IID source U has $L_2^* = 4$ and $L_3^* = 4.8$. What can be said about its entropy H? In other words, find upper lower bounds to H.* □

Remarks

Benefits of larger source lengths

For IID sources, we have seen that the benefit of making the source length K larger than 1 is to reduce the rate to no more than $(H^K+1)/K = H+1/K$, which is especially important when H is small. On the other hand, for sources with dependent random variables, we will show in a later chapter that H^K/K decreases with K, so that significantly larger reductions in rate will be possible. On the other hand, one should remember that the number of codewords and the corresponding complexity of implementation of the code increase exponentially with K.

Notice that although Theorem 2.7.3 finds R_{VL}^* exactly, it gives only bounds to $R_{VL,K}^*$. To find the latter exactly, one must apply the Huffman algorithm to find an optimum code with source length K for the probability distribution $p_{U^K}(u^k)$. By definition, the rate of this code is $R_{VL,K}^*$.

Complements

Synchronization and transmission errors

Although we have presumed that the decoder is always given the binary representation exactly as produced by the encoder, in practice, there may occasionally be mistakes. That is, bits may be deleted, inserted or changed, and if precautions are not taken, such perturbations may have large effects.

Let us first consider the situation in which a prefix code is used to encode an infinite sequence of source symbols, but for some reason, the first few bits of the binary representation become lost. Clearly, the decoder is not likely to be able to determine any of the source symbols whose codewords have missing bits. But it may also happen that subsequent source symbols are incorrectly decoded; that is, the errors caused by this loss may propagate. For example, suppose the codebook $\{01, 001, 101, 110\}$ is used for the alphabet $\{a, b, c, d\}$, suppose the codeword 110 for d is transmitted repeatedly; and suppose the first bit is lost, so the decoder is given only 10110110110110110 .... Instead of finding the codeword 110 repeated infinitely many times and decoding into $ddd\ldots$, the decoder finds 101 repeated infinitely and decodes into $ccc\ldots$ Basically, the loss of the initial bit caused the encoder to lose track of where the codewords began. We call this a loss of synchronization. Its effect on this code is disastrous. In

contrast, a loss of synchronization has very little effect on the codebook $\{1, 01, 001, 0001\}$ because the end of each codeword is so easily recognized.

A similar situation arises when bits are inserted, deleted or changed in the middle of the binary representation. The immediate effect is to incorrectly decode the affected codewords, but the more serious effect may be a loss of synchronization for decoding subsequent source symbol. Thus, in practice, if there is a realistic chance of the encoded bits being perturbed, it is advisable to use codes that permit rapid resynchronization. Usually, this entails making the codewords a little longer than would otherwise be necessary, so there is a price to pay for this kind of protection. The reader is referred to the book by Stiffler for a discussion of synchonizable codes.

Buffering

Suppose an FVB code is used in the situation where the source produces symbols at regular time intervals (say, one every T_s seconds) and a channel transmits bits at regular intervals (say, one every T_c seconds). If the encoder has rate $R = T_s/T_c$, then on the average the bit rate (in bits per second) produced by the encoder equals that which the channel can transmit, but the variable length nature of the codebook means that the actual number of bits produced in any given time interval may vary considerably from the average. To handle this situation, *buffering* is essential.

A buffer is a device capable of holding a large number of bits in their original order. The encoder feeds new bits into the buffer as it creates them, and independently, the channel removes the oldest bits at the time it transmits them. There are, however, two potential problems: overflow and underflow. The former arises when over some period of time, the encoder produces so many long codewords that the buffer fills to capacity, and new bits are lost rather than entered into the buffer. Generally, this is due to the source producing an unusually long sequence of unlikely sumbols. In effect, the rate produced by the encoder is much larger than the channel rate. In this case some of the bits will be lost. Moreover, if the loss of bits is not handled carefully, synchronization will be lost and the sort of error propagation described above may occur. To reduce the likelihood of overflow, one should choose the buffer to be large, but no matter how large the buffer, there is always some source sequence that will cause it to overflow.

Underflow is the reverse problem. Suppose over some period of time the source produces a sequence of very likely symbols, so that the encoder produces bits at a rate below that which the channel needs. At some point, the channel will find the buffer empty. Although there is nothing to transmit, the channel will nevertheless produce a bit (probably at random) at its

output, which the decoder will interpret as a real bit; i.e., it gets inserted into Z. This will cause at least one source symbol error and possibly more, if synchronization is lost. To prevent this sort of thing, whenever the buffer empties, one should immediately put in a specially designated codeword, called a *flag*, that indicates to the decoder that there was really nothing to send. The code, augmented by the flag, must be a prefix code, and this means that one or more of the codewords will be longer than they would otherwise need to be. Thus the rate of the code will be slightly larger. The flag will also add delay to the system, for once it is entered into the buffer, it must be transmitted in entirety, even if the encoder has already placed something in the buffer.

Separable codes

There are some perfectly lossless FVB codes that do not have the prefix property. For example, consider the codebook $\{1, 10, 100\}$. Since the codewords are distinct and since each begins with a 1, there will be no problem recognizing and decoding codewords in $Z_1, Z_2, \ldots$. Unlike prefix codes, however, the decoding will not be "instantaneous", in that when the codeword 10 is received by the decoder, it must wait for the next bit to determine whether the encoder sent 10 or 100.

A necessary condition for an FVB codebook to be perfectly lossless (presuming a one-to-one encoding rule and the corresponding inverse decoding rule) is that it be *separable* in the sense that the binary sequence formed by concatenating any finite number of codewords cannot also be formed by concatenating some other finite sequence of codewords. That is, if $\underline{Z}_1, \underline{Z}_2, \ldots, \underline{Z}_n$ and $\underline{Z}'_1, \underline{Z}'_2, \ldots, \underline{Z}'_m$ are codewords, then $(\underline{Z}_1\underline{Z}_2 \ldots \underline{Z}_n) = (\underline{Z}'_1\underline{Z}'_2 \ldots \underline{Z}'_m)$ if and only if $n = m$ and $\underline{Z}_i = \underline{Z}'_i$, for $i = 1, \ldots, n$. For example, the codebook $\{0, 01, 001\}$ is not separable because the binary sequence 001 corresponds to the both the codeword 001 and also the concatenation of codewords 0 and 01. There is a systematic method due to Sardinas and Patterson for determining if a codebook is separable in a finite number of steps.

Exercise 2.7.11 *Verify that prefix codes are always separable.* □

Exercise 2.7.12 *Show that any codebook with the suffix-free property that no codeword is the suffix of another is separable.* □

A result known as McMillan's Theorem shows that, just as with prefix codes, the lengths of any separable code satisfy the Kraft inequality. Accordingly, there must also be a prefix code with exactly the same lengths

and rate. This is why one can restrict attention to prefix codes with no loss in potential performance.

Separable codes are also called *uniquely decodable*. However, Exercise 2.7.13, below, suggests this is not such a good name, for although it is always possible to uniquely decode finite sequences of codewords from a separable code it is not always possible to uniquely decode infinite sequences. All the more reason to prefer prefix codes.

Exercise 2.7.13 *(a) Show that the codebook* $\{00, 001, 1010, 0101\}$ *is separable. (Hint: see Exercise 2.7.12.) (b) Show that the binary sequence* $001010101010101\ldots$ *can be decoded in two very different ways.* □

Infinite alphabet sources

It can be shown that Kraft inequality also holds for countably infinite sets of code lengths. Thus, although we restricted attention in this section to finite alphabet sources, in fact, the coding theorem for FVB prefix codes (Theorem 2.7.3) holds as stated for sources with countably infinite alphabets. Shannon-Fano coding works fine, as well. On the other hand, Huffman's code design algorithm depended greatly on the finite alphabet assumption and cannot be applied when the alphabet is countably infinite.