Elements of Information Theory
Thomas M. Cover, Joy A. Thomas
Copyright © 1991 John Wiley & Sons, Inc.
Print ISBN 0-471-06259-6 Online ISBN 0-471-20061-1

Chapter 5

Data Compression

We now put content in the definition of entropy by establishing the fundamental limit for the compression of information. Data compression can be achieved by assigning short descriptions to the most frequent outcomes of the data source and necessarily longer descriptions to the less frequent outcomes. For example, in Morse code, the most frequent symbol is represented by a single dot. In this chapter we find the shortest average description length of a random variable.

We first define the notion of an instantaneous code and then prove the important Kraft inequality, which asserts that the exponentiated codeword length assignments must look like a probability mass function. Simple calculus then shows that the expected description length must be greater than or equal to the entropy, the first main result. Then Shannon's simple construction shows that the expected description length can achieve this bound asymptotically for repeated descriptions. This establishes the entropy as a natural measure of efficient description length. The famous Huffman coding procedure for finding minimum expected description length assignments is provided. Finally, we show that Huffman codes are competitively optimal and that it requires roughly H fair coin flips to generate a sample of a random variable having entropy H.

Thus the entropy is the data compression limit as well as the number of bits needed in random number generation. And codes achieving H turn out to be optimal from many points of view.

5.1 EXAMPLES OF CODES

Definition: A *source code* C for a random variable X is a mapping from $\mathcal{X}$, the range of X, to $\mathcal{D}^*$, the set of finite length strings of symbols from a D-ary alphabet. Let $C(x)$ denote the codeword corresponding to x and let $l(x)$ denote the length of $C(x)$.

For example, $C(\text{Red}) = 00$, $C(\text{Blue}) = 11$ is a source code for $\mathcal{X} = \{\text{Red}, \text{Blue}\}$ with alphabet $\mathcal{D} = \{0, 1\}$.

Definition: The *expected length* $L(C)$ of a source code $C(x)$ for a random variable X with probability mass function $p(x)$ is given by

$$L(C) = \sum_{x \in \mathcal{X}} p(x) l(x), \tag{5.1}$$

where $l(x)$ is the length of the codeword associated with x.

Without loss of generality, we can assume that the D-ary alphabet is $\mathcal{D} = \{0, 1, \ldots, D-1\}$.

Some examples of codes follow.

Example 5.1.1: Let X be a random variable with the following distribution and codeword assignment:

$$\begin{array}{llll} \Pr(X=1) = 1/2, & \text{codeword} & C(1) = 0 \\ \Pr(X=2) = 1/4, & \text{codeword} & C(2) = 10 \\ \Pr(X=3) = 1/8, & \text{codeword} & C(3) = 110 \\ \Pr(X=4) = 1/8, & \text{codeword} & C(4) = 111. \end{array} \tag{5.2}$$

The entropy $H(X)$ of X is 1.75 bits, and the expected length $L(C) = El(X)$ of this code is also 1.75 bits. Here we have a code that has the same average length as the entropy. We note that any sequence of bits can be uniquely decoded into a sequence of symbols of X. For example, the bit string 0110111100110 is decoded as 134213.

Example 5.1.2: Consider another simple example of a code for a random variable:

$$\begin{array}{llll} \Pr(X=1) = 1/3, & \text{codeword} & C(1) = 0 \\ \Pr(X=2) = 1/3, & \text{codeword} & C(2) = 10 \\ \Pr(X=3) = 1/3, & \text{codeword} & C(3) = 11. \end{array} \tag{5.3}$$

Just as in the previous case, the code is uniquely decodable. However, in this case the entropy is $\log 3 = 1.58$ bits, while the average length of the encoding is 1.66 bits. Here $El(X) > H(X)$.

Example 5.1.3 (*Morse code*): The Morse code is a reasonably efficient code for the English alphabet using an alphabet of four symbols: a dot, a dash, a letter space and a word space. Short sequences represent frequent letters (e.g., a single dot represents E) and long sequences represent infrequent letters (e.g., Q is represented by "dash, dash, dot, dash"). This is not the optimal representation for the alphabet in four symbols—in fact, many possible codewords are not utilized because the codewords for letters do not contain spaces except for a letter space at the end of every codeword and no space can follow another space. It is an interesting problem to calculate the number of sequences that can be constructed under these constraints. The problem was solved by Shannon in his original 1948 paper. The problem is also related to coding for magnetic recording, where long strings of 0's are prohibited [2], [184].

We now define increasingly more stringent conditions on codes. Let x^n denote $(x_1, x_2, \ldots, x_n)$.

Definition: A code is said to be *non-singular* if every element of the range of X maps into a different string in $\mathcal{D}^*$, i.e.,

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j)\,. \tag{5.4}$$

Non-singularity suffices for an unambiguous description of a single value of X. But we usually wish to send a sequence of values of X. In such cases, we can ensure decodability by adding a special symbol (a "comma") between any two codewords. But this is an inefficient use of the special symbol; we can do better by developing the idea of self-punctuating or instantaneous codes. Motivated by the necessity to send sequences of symbols X, we define the extension of a code as follows:

Definition: The *extension* C^* of a code C is the mapping from finite length strings of $\mathcal{X}$ to finite length strings of $\mathcal{D}$, defined by

$$C(x_1x_2\cdots x_n) = C(x_1)C(x_2)\cdots C(x_n)\,, \tag{5.5}$$

where $C(x_1)C(x_2)\cdots C(x_n)$ indicates concatenation of the corresponding codewords.

Example 5.1.4: If $C(x_1) = 00$ and $C(x_2) = 11$, then $C(x_1x_2) = 0011$.

Definition: A code is called *uniquely decodable* if its extension is non-singular.

In other words, any encoded string in a uniquely decodable code has only one possible source string producing it. However, one may have to

look at the entire string to determine even the first symbol in the corresponding source string.

Definition: A code is called a *prefix code* or an *instantaneous code* if no codeword is a prefix of any other codeword.

An instantaneous code can be decoded without reference to the future codewords since the end of a codeword is immediately recognizable. Hence, for an instantaneous code, the symbol x_i can be decoded as soon as we come to the end of the codeword corresponding to it. We need not wait to see the codewords that come later. An instantaneous code is a "self-punctuating" code; we can look down the sequence of code symbols and add the commas to separate the codewords without looking at later symbols. For example, the binary string 01011111010 produced by the code of Example 5.1.1 is parsed as $0, 10, 111, 110, 10$.

The nesting of these definitions is shown in Figure 5.1. To illustrate the differences between the various kinds of codes, consider the following examples of codeword assignments $C(x)$ to $x \in \mathcal{X}$ in Table 5.1.

For the non-singular code, the code string 010 has three possible source sequences: 2 or 14 or 31, and hence the code is not uniquely decodable.

The uniquely decodable code is not prefix free and is hence not instantaneous. To see that it is uniquely decodable, take any code string and start from the beginning. If the first two bits are 00 or 10, they can be decoded immediately. If the first two bits are 11, then we must look at the following bits. If the next bit is a 1, then the first source symbol is a 3. If the length of the string of 0's immediately following the 11 is odd, then the first codeword must be 110 and the first source symbol must be

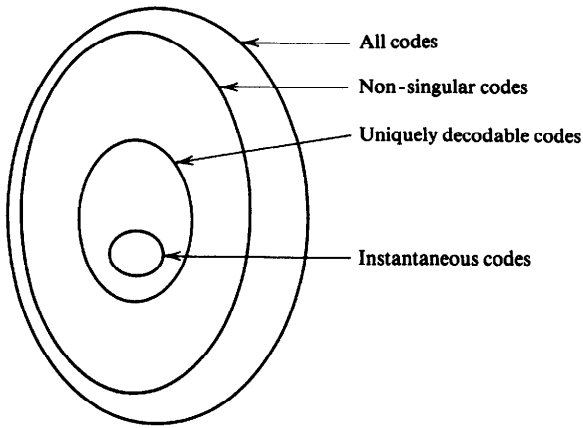


Figure 5.1. Classes of codes.

TABLE 5.1. Classes of Codes

X	Singular	Non-singular, but not uniquely decodable	Uniquely decodable, but not instantaneous	Instantaneous
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

4; if the length of the string of 0's is even, then the first source symbol is a 3. By repeating this argument, we can see that this code is uniquely decodable. Sardinas and Patterson have devised a finite test for unique decodability, which involves forming sets of possible suffixes to the codewords and systematically eliminating them. The test is described more fully in Problem 24 at the end of the chapter.

The fact that the last code in Table 5.1 is instantaneous is obvious since no codeword is a prefix of any other.

5.2 KRAFT INEQUALITY

We wish to construct instantaneous codes of minimum expected length to describe a given source. It is clear that we cannot assign short codewords to all source symbols and still be prefix free. The set of codeword lengths possible for instantaneous codes is limited by the following inequality:

Theorem 5.2.1 (*Kraft inequality*): *For any instantaneous code (prefix code) over an alphabet of size D, the codeword lengths* $l_1, l_2, \ldots, l_m$ *must satisfy the inequality*

$$\sum_i D^{-l_i} \leq 1 . \tag{5.6}$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.

Proof: Consider a D-ary tree in which each node has D children. Let the branches of the tree represent the symbols of the codeword. For example, the D branches arising from the root node represent the D possible values of the first symbol of the codeword. Then each codeword is represented by a leaf on the tree. The path from the root traces out the symbols of the codeword. A binary example of such a tree is shown in Figure 5.2.

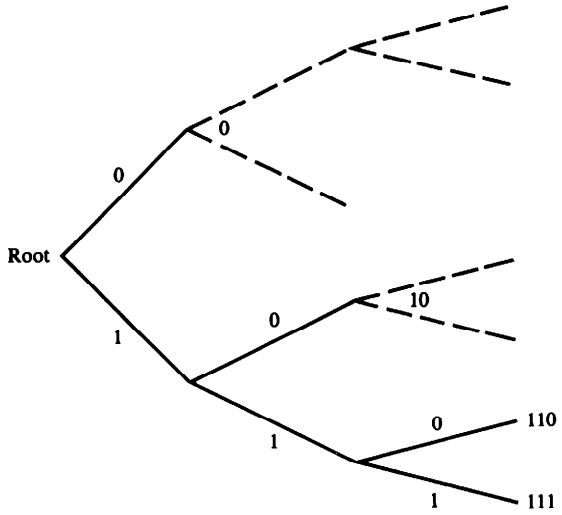


Figure 5.2. Code tree for the Kraft inequality.

The prefix condition on the codewords implies that no codeword is an ancestor of any other codeword on the tree. Hence, each codeword eliminates its descendants as possible codewords.

Let $l_{\max}$ be the length of the longest codeword of the set of codewords. Consider all nodes of the tree at level $l_{\max}$. Some of them are codewords, some are descendants of codewords, and some are neither. A codeword at level l_i has $D^{l_{\max}-l_i}$ descendants at level $l_{\max}$. Each of these descendant sets must be disjoint. Also, the total number of nodes in these sets must be less than or equal to $D^{l_{\max}}$. Hence, summing over all the codewords, we have

$$\sum D^{l_{\max}-l_i} \leq D^{l_{\max}} \tag{5.7}$$

or

$$\sum D^{-l_i} \leq 1\,, \tag{5.8}$$

which is the Kraft inequality.

Conversely, given any set of codeword lengths $l_1, l_2, \ldots, l_m$ which satisfy the Kraft inequality, we can always construct a tree like the one in Figure 5.2. Label the first node (lexicographically) of depth l_1 as codeword 1, and remove its descendants from the tree. Then label the first remaining node of depth l_2 as codeword 2, etc. Proceeding this way, we construct a prefix code with the specified $l_1, l_2, \ldots, l_m$. □

We now show that an infinite prefix code also satisfies the Kraft inequality.

Theorem 5.2.2 (*Extended Kraft Inequality*): *For any countably infinite set of codewords that form a prefix code, the codeword lengths satisfy the extended Kraft inequality,*

$$\sum_{i=1}^{\infty} D^{-l_i} \leq 1 . \tag{5.9}$$

Conversely, given any $l_1, l_2, \ldots$ *satisfying the extended Kraft inequality, we can construct a prefix code with these codeword lengths.*

Proof: Let the D-ary alphabet be $\{0, 1, \ldots, D-1\}$. Consider the ith codeword $y_1 y_2 \ldots y_{l_i}$. Let $0.y_1 y_2 \cdots y_{l_i}$ be the real number given by the D-ary expansion

$$0.y_1 y_2 \cdots y_{l_i} = \sum_{j=1}^{l_i} y_j D^{-j} . \tag{5.10}$$

This codeword corresponds to the interval

$$\left(0.y_1 y_2 \cdots y_{l_i}, 0.y_1 y_2 \cdots y_{l_i} + \frac{1}{D^{l_i}}\right), \tag{5.11}$$

the set of all real numbers whose D-ary expansion begins with $0.y_1 y_2 \cdots y_{l_i}$. This is a subinterval of the unit interval $[0, 1]$. By the prefix condition, these intervals are disjoint. Hence the sum of their lengths has to be less than or equal to 1.

This proves that

$$\sum_{i=1}^{\infty} D^{-l_i} \leq 1 . \tag{5.12}$$

Just as in the finite case, we can reverse the proof to construct the code for a given $l_1, l_2, \ldots$ that satisfies the Kraft inequality. First reorder the indexing so that $l_1 \geq l_2 \geq \ldots$. Then simply assign the intervals in order from the low end of the unit interval. □

In Section 5.5, we will show that the lengths of codewords for a uniquely decodable code also satisfy the Kraft inequality. Before we do that, we consider the problem of finding the shortest instantaneous code.

5.3 OPTIMAL CODES

In the previous section, we proved that any codeword set that satisfies the prefix condition has to satisfy the Kraft inequality and that the

Kraft inequality is a sufficient condition for the existence of a codeword set with the specified set of codeword lengths. We now consider the problem of finding the prefix code with the minimum expected length. From the results of the previous section, this is equivalent to finding the set of lengths $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality and whose expected length $L = \Sigma\, p_i l_i$ is less than the expected length of any other prefix code. This is a standard optimization problem: Minimize

$$L = \sum p_i l_i \tag{5.13}$$

over all integers $l_1, l_2, \ldots, l_m$ satisfying

$$\sum D^{-l_i} \leq 1 . \tag{5.14}$$

A simple analysis by calculus suggests the form of the minimizing l_i^*. We neglect the integer constraint on l_i and assume equality in the constraint. Hence, we can write the constrained minimization using Lagrange multipliers as the minimization of

$$J = \sum p_i l_i + \lambda\left(\sum D^{-l_i}\right) . \tag{5.15}$$

Differentiating with respect to l_i, we obtain

$$\frac{\partial J}{\partial l_i} = p_i - \lambda D^{-l_i} \log_e D . \tag{5.16}$$

Setting the derivative to 0, we obtain

$$D^{-l_i} = \frac{p_i}{\lambda \log_e D} . \tag{5.17}$$

Substituting this in the constraint to find λ, we find $\lambda = 1/\log_e D$ and hence

$$p_i = D^{-l_i} , \tag{5.18}$$

yielding optimal codelengths

$$l_i^* = -\log_D p_i . \tag{5.19}$$

This non-integer choice of codeword lengths yields expected codeword length

$$L^* = \sum p_i l_i^* = -\sum p_i \log_D p_i = H_D(X) . \tag{5.20}$$

But since the l_i must be integers, we will not always be able to set the codeword lengths as in (5.19). Instead, we should choose a set of codeword lengths l_i "close" to the optimal set. Rather than demonstrate by calculus that $l_i^* = -\log_D p_i$ is a global minimum, we will verify optimality directly in the proof of the following theorem.

Theorem 5.3.1: *The expected length L of any instantaneous D-ary code for a random variable X is greater than or equal to the entropy $H_D(X)$, i.e.,*

$$L \geq H_D(X) \tag{5.21}$$

with equality iff $D^{-l_i} = p_i$.

Proof: We can write the difference between the expected length and the entropy as

$$L - H_D(X) = \sum p_i l_i - \sum p_i \log_D \frac{1}{p_i} \tag{5.22}$$

$$= -\sum p_i \log_D D^{-l_i} + \sum p_i \log_D p_i \,. \tag{5.23}$$

Letting $r_i = D^{-l_i}/\Sigma_j D^{-l_j}$ and $c = \Sigma D^{-l_i}$, we obtain

$$L - H = \sum p_i \log_D \frac{p_i}{r_i} - \log_D c \tag{5.24}$$

$$= D(\mathbf{p}\|\mathbf{r}) + \log_D \frac{1}{c} \tag{5.25}$$

$$\geq 0 \tag{5.26}$$

by the non-negativity of relative entropy and the fact (Kraft inequality) that $c \leq 1$. Hence $L \geq H$ with equality iff $p_i = D^{-l_i}$, i.e., iff $-\log_D p_i$ is an integer for all i. □

Definition: A probability distribution is called *D-adic* with respect to D if each of the probabilities is equal to D^{-n} for some n.

Thus we have equality in the theorem if and only if the distribution of X is D-adic.

The preceding proof also indicates a procedure for finding an optimal code: find the D-adic distribution that is closest (in the relative entropy sense) to the distribution of X. This distribution provides the set of codeword lengths. Construct the code by choosing the first available node as in the proof of the Kraft inequality. We then have an optimal code for X.

However, this procedure is not easy, since the search for the closest D-adic distribution is not obvious. In the next section, we give a good suboptimal procedure (Shannon-Fano coding). In Section 5.6, we describe a simple procedure (Huffman coding) for actually finding the optimal code.

5.4 BOUNDS ON THE OPTIMAL CODELENGTH

We now demonstrate a code that achieves an expected description length L within 1 bit of the lower bound, that is,

$$H(X) \leq L < H(X) + 1 . \tag{5.27}$$

Recall the setup of the last section: we wish to minimize $L = \sum p_i l_i$ subject to the constraint that $l_1, l_2, \ldots, l_m$ are integers and $\sum D^{-l_i} \leq 1$. We proved that the optimal codeword lengths can be found by finding the D-adic probability distribution closest to the distribution of X in relative entropy i.e., finding the D-adic $\mathbf{r}$ ($r_i = D^{-l_i}/\sum_j D^{-l_j}$) minimizing

$$L - H_D = D(\mathbf{p}\|\mathbf{r}) - \log\left(\sum D^{-l_i}\right) \geq 0 . \tag{5.28}$$

The choice of word lengths $l_i = \log_D \frac{1}{p_i}$ yields $L = H$. Since $\log_D \frac{1}{p_i}$ may not equal an integer, we round it up to give integer word length assignments,

$$l_i = \left\lceil \log_D \left(\frac{1}{p_i}\right) \right\rceil , \tag{5.29}$$

where $\lceil x \rceil$ is the smallest integer $\geq x$. These lengths satisfy the Kraft inequality since

$$\sum D^{-\left\lceil \log \frac{1}{p_i} \right\rceil} \leq \sum D^{-\log \frac{1}{p_i}} = \sum p_i = 1 . \tag{5.30}$$

This choice of codeword lengths satisfies

$$\log_D \frac{1}{p_i} \leq l_i < \log_D \frac{1}{p_i} + 1 . \tag{5.31}$$

Multiplying by p_i and summing over i, we obtain

$$H_D(X) \leq L < H_D(X) + 1 . \tag{5.32}$$

Since the optimal code can only be better than this code, we have the following theorem:

Theorem 5.4.1: *Let $l_1^*, l_2^*, \ldots, l_m^*$ be the optimal codeword lengths for a source distribution* **p** *and a D-ary alphabet, and let L^* be the associated expected length of the optimal code ($L^* = \sum p_i l_i^*$). Then*

$$H_D(X) \leq L^* < H_D(X) + 1 . \tag{5.33}$$

Proof: Let $l_i = \lceil \log_D \frac{1}{p_i} \rceil$. Then l_i satisfies the Kraft inequality and from (5.32) we have

$$H_D(X) \leq L = \sum p_i l_i < H_D(X) + 1 . \tag{5.34}$$

But since L^*, the expected length of the optimal code, is less than $L = \sum p_i l_i$, and since $L^* \geq H_D$ from Theorem 5.3.1, we have the theorem. □

In the preceding theorem, there is an overhead which is at most 1 bit, due to the fact that $\log \frac{1}{p_i}$ is not always an integer. We can reduce the overhead per symbol by spreading it out over many symbols. With this in mind, let us consider a system in which we send a sequence of n symbols from X. The symbols are assumed to be drawn i.i.d. according to $p(x)$. We can consider these n symbols to be a supersymbol from the alphabet $\mathcal{X}^n$.

Define L_n to be the expected codeword length per input symbol, i.e., if $l(x_1, x_2, \ldots, x_n)$ is the length of the codeword associated with $(x_1, x_2, \ldots, x_n)$, then

$$L_n = \frac{1}{n} \sum p(x_1, x_2, \ldots, x_n) l(x_1, x_2, \ldots, x_n) = \frac{1}{n} El(X_1, X_2, \ldots, X_n) . \tag{5.35}$$

We can now apply the bounds derived above to the code:

$$H(X_1, X_2, \ldots, X_n) \leq El(X_1, X_2, \ldots, X_n) < H(X_1, X_2, \ldots, X_n) + 1 . \tag{5.36}$$

Since $X_1, X_2, \ldots, X_n$ are i.i.d., $H(X_1, X_2, \ldots, X_n) = \sum H(X_i) = nH(X)$. Dividing (5.36) by n, we obtain

$$H(X) \leq L_n < H(X) + \frac{1}{n} . \tag{5.37}$$

Hence by using large block lengths we can achieve an expected codelength per symbol arbitrarily close to the entropy.

We can also use the same argument for a sequence of symbols from a stochastic process that is not necessarily i.i.d. In this case, we still have the bound

$$H(X_1, X_2, \ldots, X_n) \leq El(X_1, X_2, \ldots, X_n) < H(X_1, X_2, \ldots, X_n) + 1 . \tag{5.38}$$

Dividing by n again and defining L_n to be the expected description length per symbol, we obtain

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n} . \tag{5.39}$$

If the stochastic process is stationary, then $H(X_1, X_2, \ldots, X_n)/n \to H(\mathcal{X})$, and the expected description length tends to the entropy rate as $n \to \infty$. Thus we have the following theorem:

Theorem 5.4.2: *The minimum expected codeword length per symbol satisfies*

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n^* < \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n} . \tag{5.40}$$

Moreover, if $X_1, X_2, \ldots, X_n$ is a stationary stochastic process,

$$L_n^* \to H(\mathcal{X}) , \tag{5.41}$$

where $H(\mathcal{X})$ is the entropy rate of the process.

This theorem provides another justification for the definition of entropy rate—it is the expected number of bits per symbol required to describe the process.

Finally, we ask what happens to the expected description length if the code is designed for the wrong distribution. For example, the wrong distribution may be the best estimate that we can make of the unknown true distribution.

We consider the Shannon code assignment $l(x) = \lceil \log \frac{1}{q(x)} \rceil$ designed for the probability mass function $q(x)$. Suppose the true probability mass function is $p(x)$. Thus we will not achieve expected length $L \approx H(p) = -\sum p(x) \log p(x)$. We now show that the increase in expected description length due to the incorrect distribution is the relative entropy $D(p \| q)$. Thus $D(p \| q)$ has a concrete interpretation as the increase in descriptive complexity due to incorrect information.

Theorem 5.4.3: *The expected length under $p(x)$ of the code assignment $l(x) = \lceil \log \frac{1}{q(x)} \rceil$ satisfies*

$$H(p) + D(p \| q) \leq E_p l(X) < H(p) + D(p \| q) + 1 . \tag{5.42}$$

Proof: The expected codelength is

$$El(X) = \sum_x p(x)\left\lceil \log \frac{1}{q(x)} \right\rceil \tag{5.43}$$

$$< \sum_x p(x)\left(\log \frac{1}{q(x)} + 1 \right) \tag{5.44}$$

$$= \sum_x p(x) \log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1 \tag{5.45}$$

$$= \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \tag{5.46}$$

$$= D(p\|q) + H(p) + 1 . \tag{5.47}$$

The lower bound can be derived similarly. □

Thus using the wrong distribution incurs a penalty of $D(p\|q)$ in the average description length.

5.5 KRAFT INEQUALITY FOR UNIQUELY DECODABLE CODES

We have proved that any instantaneous code must satisfy the Kraft inequality. The class of uniquely decodable codes is larger than the class of instantaneous codes, so one expects to achieve a lower expected codeword length if L is minimized over all uniquely decodable codes. In this section, we prove that the class of uniquely decodable codes does not offer any further possibilities for the set of codeword lengths than do instantaneous codes. We now give Karush's elegant proof of the following theorem.

Theorem 5.5.1 (*McMillan*): *The codeword lengths of any uniquely decodable code must satisfy the Kraft inequality*

$$\sum D^{-l_i} \leq 1 . \tag{5.48}$$

Conversely, given a set of codeword lengths that satisfy this inequality, it is possible to construct a uniquely decodable code with these codeword lengths.

Proof: Consider C^k, the kth extension of the code, i.e., the code formed by the concatenation of k repetitions of the given uniquely decodable code C. By the definition of unique decodability, the kth extension of the code is non-singular. Since there are only D^n different D-ary strings of length n, unique decodability implies that the number

of code sequences of length n in the kth extension of the code must be no greater than D^n. We now use this observation to prove the Kraft inequality.

Let the codeword lengths of the symbols $x \in \mathcal{X}$ be denoted by $l(x)$. For the extension code, the length of the code-sequence is

$$l(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} l(x_i) . \tag{5.49}$$

The inequality that we wish to prove is

$$\sum_{x \in \mathcal{X}} D^{-l(x)} \leq 1 . \tag{5.50}$$

The trick is to consider the kth power of this quantity. Thus

$$\left(\sum_{x \in \mathcal{X}} D^{-l(x)} \right)^k = \sum_{x_1 \in \mathcal{X}} \sum_{x_2 \in \mathcal{X}} \cdots \sum_{x_k \in \mathcal{X}} D^{-l(x_1)} D^{-l(x_2)} \cdots D^{-l(x_k)} \tag{5.51}$$

$$= \sum_{x_1, x_2, \ldots, x_k \in \mathcal{X}^k} D^{-l(x_1)} D^{-l(x_2)} \cdots D^{-l(x_k)} \tag{5.52}$$

$$= \sum_{x^k \in \mathcal{X}^k} D^{-l(x^k)} , \tag{5.53}$$

by (5.49). We now gather the terms by word lengths to obtain

$$\sum_{x^k \in \mathcal{X}^k} D^{-l(x^k)} = \sum_{m=1}^{kl_{\max}} a(m) D^{-m} , \tag{5.54}$$

where $l_{\max}$ is the maximum codeword length and $a(m)$ is the number of source sequences x^k mapping into codewords of length m. But the code is uniquely decodable, so there is at most one sequence mapping into each code m-sequence and there are at most D^m code m-sequences. Thus $a(m) \leq D^m$, and we have

$$\left(\sum_{x \in \mathcal{X}} D^{-l(x)} \right)^k = \sum_{m=1}^{kl_{\max}} a(m) D^{-m} \tag{5.55}$$

$$\leq \sum_{m=1}^{kl_{\max}} D^m D^{-m} \tag{5.56}$$

$$= kl_{\max} \tag{5.57}$$

and hence

$$\sum_j D^{-l_j} \leq \left(kl_{\max} \right)^{1/k} . \tag{5.58}$$

Since this inequality is true for all k, it is true in the limit as $k \to \infty$. Since $(kl_{\max})^{1/k} \to 1$, we have

$$\sum_j D^{-l_j} \leq 1 , \tag{5.59}$$

which is the Kraft inequality.

Conversely, given any set of $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality, we can construct an instantaneous code as proved in Section 5.2. Since every instantaneous code is uniquely decodable, we have also constructed a uniquely decodable code. □

Corollary: *A uniquely decodable code for an infinite source alphabet $\mathcal{X}$ also satisfies the Kraft inequality.*

Proof: The point at which the preceding proof breaks down for infinite $|\mathcal{X}|$ is at (5.58), since for an infinite code $l_{\max}$ is infinite. But there is a simple fix to the proof. Any subset of a uniquely decodable code is also uniquely decodable; hence, any finite subset of the infinite set of codewords satisfies the Kraft inequality. Hence,

$$\sum_{i=1}^{\infty} D^{-l_i} = \lim_{N \to \infty} \sum_{i=1}^{N} D^{-l_i} \leq 1 . \tag{5.60}$$

Given a set of word lengths $l_1, l_2, \ldots$ that satisfy the Kraft inequality, we can construct an instantaneous code as in the last section. Since instantaneous codes are uniquely decodable, we have constructed a uniquely decodable code with an infinite number of codewords. So the McMillan theorem also applies to infinite alphabets. □

The theorem implies a rather surprising result—that the class of uniquely decodable codes does not offer any further choices for the set of codeword lengths than the class of prefix codes. The set of achievable codeword lengths is the same for uniquely decodable and instantaneous codes. Hence the bounds derived on the optimal codeword lengths continue to hold even when we expand the class of allowed codes to the class of all uniquely decodable codes.

5.6 HUFFMAN CODES

An optimal (shortest expected length) prefix code for a given distribution can be constructed by a simple algorithm discovered by Huffman [138]. We will prove that any other code for the same alphabet cannot have a lower expected length than the code constructed by the

algorithm. Before we give any formal proofs, let us introduce Huffman codes with some examples:

Example 5.6.1: Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively. We expect the optimal binary code for X to have the longest codewords assigned to the symbols 4 and 5. Both these lengths must be equal, since otherwise we can delete a bit from the longer codeword and still have a prefix code, but with a shorter expected length. In general, we can construct a code in which the two longest codewords differ only in the last bit. For this code, we can combine the symbols 4 and 5 together into a single source symbol, with a probability assignment 0.30. Proceeding this way, combining the two least likely symbols into one symbol, until we are finally left with only one symbol, and then assigning codewords to the symbols, we obtain the following table:

Codeword length	Codeword	X	Probability
2	01	1	0.25 → 0.3 → 0.45 → 0.55 → 1
2	10	2	0.25 → 0.25 → 0.3 → 0.45
2	11	3	0.2 → 0.25 → 0.25
3	000	4	0.15 → 0.2
3	001	5	0.15

This code has average length 2.3 bits.

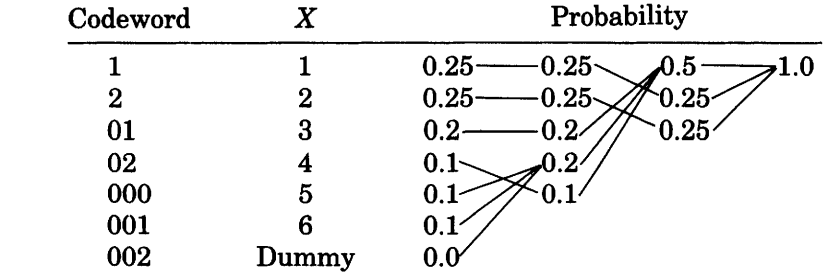
Example 5.6.2: Consider a ternary code for the same random variable. Now we combine the three least likely symbols into one supersymbol and obtain the following table:

Codeword	X	Probability
1	1	0.25 → 0.5 → 1
2	2	0.25 → 0.25
00	3	0.2 → 0.25
01	4	0.15
02	5	0.15

This code has an average length of 1.5 ternary digits.

Example 5.6.3: If $D \geq 3$, we may not have a sufficient number of symbols so that we can combine them D at a time. In such a case, we add dummy symbols to the end of the set of symbols. The dummy symbols have probability 0 and are inserted to fill the tree. Since at each

stage of the reduction, the number of symbols is reduced by $D-1$, we want the total number of symbols to be $1+k(D-1)$, where k is the number of levels in the tree. Hence, we add enough dummy symbols so that the total number of symbols is of this form. For example:



This code has an average length of 1.7 ternary digits.

A proof of the optimality of Huffman coding will be given in Section 5.8.

5.7 SOME COMMENTS ON HUFFMAN CODES

1. **Equivalence of source coding and 20 questions.** We now digress to show the equivalence of coding and the game of 20 questions.

   Supposing we wish to find the most efficient series of yes-no questions to determine an object from a class of objects. Assuming we know the probability distribution on the objects, can we find the most efficient sequence of questions?

   We first show that a sequence of questions is equivalent to a code for the object. Any question depends only on the answers to the questions before it. Since the sequence of answers uniquely determines the object, each object has a different sequence of answers, and if we represent the yes-no answers by 0's and 1's, we have a binary code for the set of objects. The average length of this code is the average number of questions for the questioning scheme.

   Also, from a binary code for the set of objects, we can find a sequence of questions that correspond to the code, with the average number of questions equal to the expected codeword length of the code. The first question in this scheme becomes "Is the first bit equal to 1 in the object's codeword?"

   Since the Huffman code is the best source code for a random variable, the optimal series of questions is that determined by the Huffman code. In Example 5.6.1, the optimal first question is "Is X

equal to 2 or 3?" The answer to this determines the first bit of the Huffman code. Assuming the answer to the first question is "Yes," the next question should be "Is X equal to 3?" which determines the second bit. However, we need not wait for the answer to the first question to ask the second. We can ask as our second question "Is X equal to 1 or 3?" determining the second bit of the Huffman code independently of the first.

The expected number of questions EQ in this optimal scheme satisfies

$$H(X) \leq EQ < H(X) + 1 . \tag{5.61}$$

2. **Huffman coding for weighted codewords.** Huffman's algorithm for minimizing $\Sigma\, p_i l_i$ can be applied to any set of numbers $p_i \geq 0$, regardless of $\Sigma\, p_i$. In this case, the Huffman code minimizes the sum of weighted codelengths $\Sigma\, w_i l_i$ rather than the average codelength.

Example 5.7.1: We perform the weighted minimization using the same algorithm.

X	Codeword	Weights
1	00	5 8 10 18
2	01	5 5 8
3	10	4 5
4	11	4

In this case the code minimizes the weighted sum of the codeword lengths, and the minimum weighted sum is 36.

3. **Huffman coding and "slice" questions.** We have described the equivalence of source coding with the game of 20 questions. The optimal sequence of questions corresponds to an optimal source code for the random variable. However, Huffman codes ask arbitrary questions of the form "Is $X \in A$?" for any set $A \subseteq \{1, 2, \ldots, m\}$.

Now we consider the game of 20 questions with a restricted set of questions. Specifically, we assume that the elements of $\mathcal{X} = \{1, 2, \ldots, m\}$ are ordered so that $p_1 \geq p_2 \geq \cdots \geq p_m$ and that the only questions allowed are of the form "Is $X > a$?" for some a.

The Huffman code constructed by the Huffman algorithm may not correspond to "slices" (sets of the form $\{x : x < a\}$). If we take the codeword lengths ($l_1 \leq l_2 \leq \cdots \leq l_m$, by Lemma 5.8.1) derived from the Huffman code and use them to assign the symbols to the

code tree by taking the first available node at the corresponding level, we will construct another optimal code. However, unlike the Huffman code itself, this code is a "slice" code, since each question (each bit of the code) splits the tree into sets of the form $\{x : x > a\}$ and $\{x : x < a\}$.

We illustrate this with an example.

Example 5.7.2: Consider the first example of Section 5.6. The code that was constructed by the Huffman coding procedure is not a "slice" code. But using the codeword lengths from the Huffman procedure, namely, $\{2, 2, 2, 3, 3\}$, and assigning the symbols to the first available node on the tree, we obtain the following code for this random variable:

$$1 \rightarrow 00, \quad 2 \rightarrow 01, \quad 3 \rightarrow 10, \quad 4 \rightarrow 110, \quad 5 \rightarrow 111$$

It can be verified that this code is a "slice" code. These "slice" codes are known as *alphabetic codes* because the codewords are alphabetically ordered.

4. **Huffman codes and Shannon codes.** Using codeword lengths of $\lceil \log \frac{1}{p_i} \rceil$ (which is called Shannon coding) may be much worse than the optimal code for some particular symbol. For example, consider two symbols, one of which occurs with probability 0.9999 and the other with probability 0.0001. Then using codeword lengths of $\lceil \log \frac{1}{p_i} \rceil$ implies using codeword lengths of 1 bit and 14 bits respectively. The optimal codeword length is obviously 1 bit for both symbols. Hence, the code for the infrequent symbol is much longer in the Shannon code than in the optimal code.

   Is it true that the codeword lengths for an optimal code are always less than $\lceil \log \frac{1}{p_i} \rceil$? The following example illustrates that this is not always true.

Example 5.7.3: Consider a random variable X with a distribution $(\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{12})$. The Huffman coding procedure results in codeword lengths of $(2, 2, 2, 2)$ or $(1, 2, 3, 3)$ (depending on where one puts the merged probabilities, as the reader can verify). Both these codes achieve the same expected codeword length. In the second code, the third symbol has length 3, which is greater than $\lceil \log \frac{1}{p_3} \rceil$. Thus the codeword length for a Shannon code could be less than the codeword length of the corresponding symbol of an optimal (Huffman) code.

   This example also illustrates the fact that the set of codeword lengths for an optimal code is not unique (there may be more than one set of lengths with the same expected value).

Although either the Shannon code or the Huffman code can be shorter for individual symbols, the Huffman code is shorter on the average. Also, the Shannon code and the Huffman code differ by less than one bit in expected codelength (since both lie between H and $H + 1$.)

5. **Fano codes.** Fano proposed a suboptimal procedure for constructing a source code, which is similar to the idea of slice codes. In his method, we first order the probabilities in decreasing order. Then we choose k such that $|\Sigma_{i=1}^{k} p_i - \Sigma_{i=k+1}^{m} p_i|$ is minimized. This point divides the source symbols into two sets of almost equal probability. Assign 0 for the first bit of the upper set and 1 for the lower set. Repeat this process for each subset. By this recursive procedure, we obtain a code for each source symbol. This scheme, though not optimal in general, achieves $L(C) \leq H(X) + 2$. (See [137].)

5.8 OPTIMALITY OF HUFFMAN CODES

We prove by induction that the binary Huffman code is optimal. It is important to remember that there are many optimal codes: inverting all the bits or exchanging two codewords of the same length will give another optimal code. The Huffman procedure constructs one such optimal code. To prove the optimality of Huffman codes, we first prove some properties of a particular optimal code.

Without loss of generality, we will assume that the probability masses are ordered, so that $p_1 \geq p_2 \geq \cdots \geq p_m$. Recall that a code is optimal if $\Sigma\, p_i l_i$ is minimal.

Lemma 5.8.1: *For any distribution, there exists an optimal instantaneous code (with minimum expected length) that satisfies the following properties:*

1. *If $p_j > p_k$, then $l_j \leq l_k$.*
2. *The two longest codewords have the same length.*
3. *The two longest codewords differ only in the last bit and correspond to the two least likely symbols.*

Proof: The proof amounts to swapping, trimming and rearranging, as shown in Figure 5.3. Consider an optimal code C_m:

- *If $p_j > p_k$, then $l_j \leq l_k$.* Here we swap codewords.

  Consider C_m', with the codewords j and k of C_m interchanged. Then

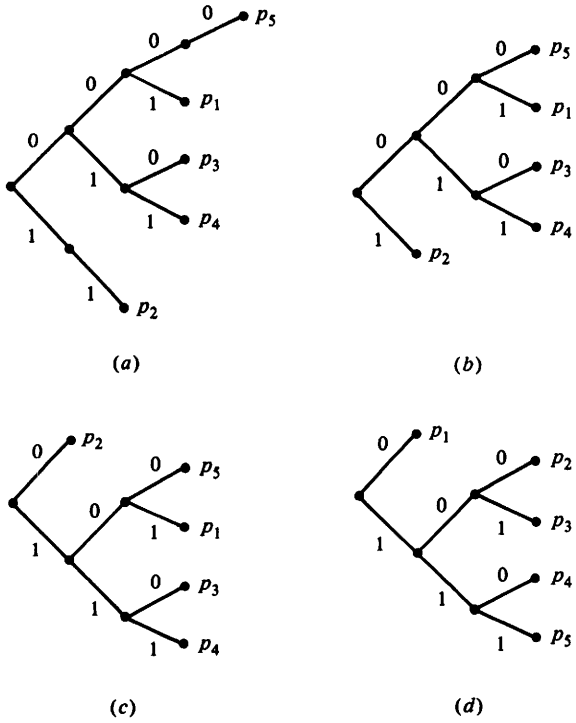


Figure 5.3. Properties of optimal codes. We will assume that $p_1 \geq p_2 \geq \cdots \geq p_m$. A possible instantaneous code is given in (*a*). By trimming branches without siblings, we improve the code to (*b*). We now rearrange the tree as shown in (*c*) so that the word lengths are ordered by increasing length from top to bottom. Finally, we swap probability assignments to improve the expected depth of the tree as shown in (*d*). Thus every optimal code can be rearranged and swapped into the canonical form (*d*). Note that $l_1 \leq l_2 \leq \cdots \leq l_m$, that $l_{m-1} = l_m$, and the last two codewords differ only in the last bit.

$$L(C'_m) - L(C_m) = \sum p_i l'_i - \sum p_i l_i \tag{5.62}$$

$$= p_j l_k + p_k l_j - p_j l_j - p_k l_k \tag{5.63}$$

$$= (p_j - p_k)(l_k - l_j)\,. \tag{5.64}$$

But $p_j - p_k > 0$, and since C_m is optimal, $L(C'_m) - L(C_m) \geq 0$. Hence we must have $l_k \geq l_j$. Thus C_m itself satisfies property 1.

- *The two longest codewords are of the same length.* Here we trim the codewords.

  If the two longest codewords are not of the same length, then one can delete the last bit of the longer one, preserving the prefix property and achieving lower expected codeword length. Hence the

two longest codewords must have the same length. By property 1, the longest codewords must belong to the least probable source symbols.

- *The two longest codewords differ only in the last bit and correspond to the two least likely symbols.* Not all optimal codes satisfy this property, but by rearranging, we can find a code that does.

  If there is a maximal length codeword without a sibling, then we can delete the last bit of the codeword and still satisfy the prefix property. This reduces the average codeword length and contradicts the optimality of the code. Hence every maximal length codeword in any optimal code has a sibling.

  Now we can exchange the longest length codewords so the two lowest probability source symbols are associated with two siblings on the tree. This does not change the expected length $\Sigma\, p_i l_i$. Thus the codewords for the two lowest probability source symbols have maximal length and agree in all but the last bit.

Summarizing, we have shown that if $p_1 \geq p_2 \geq \cdots \geq p_m$, then there exists an optimal code with $l_1 \leq l_2 \leq \cdots \leq l_{m-1} = l_m$, and codewords $C(x_{m-1})$ and $C(x_m)$ that differ only in the last bit. □

Thus we have shown that there exists an optimal code satisfying the properties of the lemma. We can now restrict our search to codes that satisfy these properties.

For a code C_m satisfying the properties of the lemma, we now define a "merged" code C_{m-1} for $m-1$ symbols as follows: take the common prefix of the two longest codewords (corresponding to the two least likely symbols), and allot it to a symbol with probability $p_{m-1} + p_m$. All the other codewords remain the same. The correspondence is shown in the following:

$$
\begin{array}{cccccc}
 & C_{m-1} & & C_m & \\
p_1 & w_1' & l_1' & w_1 = w_1' & l_1 = l_1' \\
p_2 & w_2' & l_2' & w_2 = w_2' & l_2 = l_2' \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
p_{m-2} & w_{m-2}' & l_{m-2}' & w_{m-2} = w_{m-2}' & l_{m-2} = l_{m-2}' \\
p_{m-1} + p_m & w_{m-1}' & l_{m-1}' & w_{m-1} = w_{m-1}'0 & l_{m-1} = l_{m-1}' + 1 \\
 & & & w_m = w_{m-1}'1 & l_m = l_{m-1}' + 1
\end{array}
\tag{5.65}
$$

where w denotes a binary codeword and l denotes its length. The expected length of the code C_m is

$$
L(C_m) = \sum_{i=1}^{m} p_i l_i \tag{5.66}
$$

$$= \sum_{i=1}^{m-2} p_i l'_i + p_{m-1}(l'_{m-1} + 1) + p_m(l'_{m-1} + 1) \tag{5.67}$$

$$= \sum_{i=1}^{m-1} p_i l'_i + p_{m-1} + p_m \tag{5.68}$$

$$= L(C_{m-1}) + p_{m-1} + p_m \,. \tag{5.69}$$

Thus the expected length of the code C_m differs from the expected length of C_{m-1} by a fixed amount independent of C_{m-1}. Thus minimizing the expected length $L(C_m)$ is equivalent to minimizing $L(C_{m-1})$. Thus we have reduced the problem to one with $m-1$ symbols and probability masses $(p_1, p_2, \ldots, p_{m-2}, p_{m-1} + p_m)$. This step is illustrated in Figure 5.4. We again look for a code which satisfies the properties of Lemma 5.8.1 for these $m-1$ symbols and then reduce the problem to finding the optimal code for $m-2$ symbols with the appropriate probability masses obtained by merging the two lowest probabilities on the previous merged list. Proceeding this way, we finally reduce the problem to two symbols, for which the solution is obvious, i.e., allot 0 for one of the symbols and 1 for the other. Since we have maintained optimality at

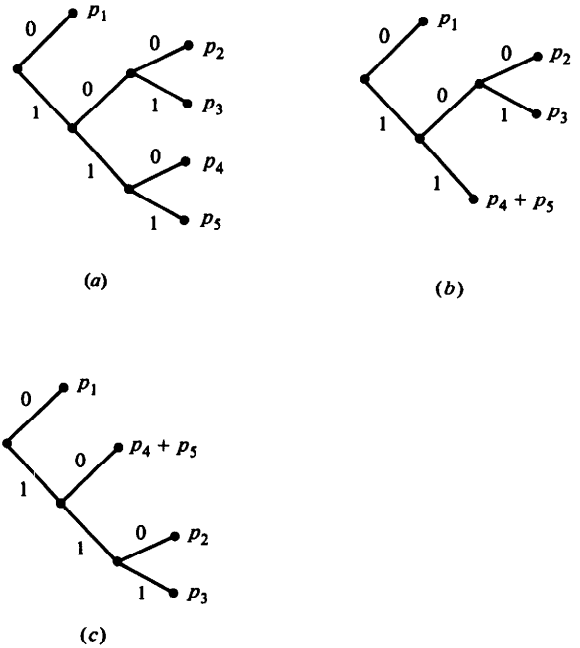


Figure 5.4. Induction step for Huffman coding. Let $p_1 \geq p_1 \geq \cdots \geq p_5$. A canonical optimal code is illustrated in (*a*). Combining the two lowest probabilities, we obtain the code in (*b*). Rearranging the probabilities in decreasing order, we obtain the canonical code in (*c*) for $m-1$ symbols.

every stage in the reduction, the code constructed for m symbols is optimal. Thus we have proved the following theorem for binary alphabets.

Theorem 5.8.1: *Huffman coding is optimal, i.e., if C^* is the Huffman code and C' is any other code, then $L(C^*) \leq L(C')$.*

Although we have proved the theorem for a binary alphabet, the proof can be extended to establishing optimality of the Huffman coding algorithm for a D-ary alphabet as well. Incidentally, we should remark that Huffman coding is a "greedy" algorithm in that it coalesces the two least likely symbols at each stage. The above proof shows that this local optimality ensures a global optimality of the final code.

5.9 SHANNON-FANO-ELIAS CODING

In Section 5.4, we showed that the set of lengths $l(x) = \lceil \log \frac{1}{p(x)} \rceil$ satisfies the Kraft inequality and can therefore be used to construct a uniquely decodable code for the source. In this section, we describe a simple constructive procedure which uses the cumulative distribution function to allot codewords.

Without loss of generality we can take $\mathcal{X} = \{1, 2, \ldots, m\}$. Assume $p(x) > 0$ for all x. The cumulative distribution function $F(x)$ is defined as

$$F(x) = \sum_{a \leq x} p(a) . \tag{5.70}$$

This function is illustrated in Figure 5.5. Consider the modified cumulative distribution function

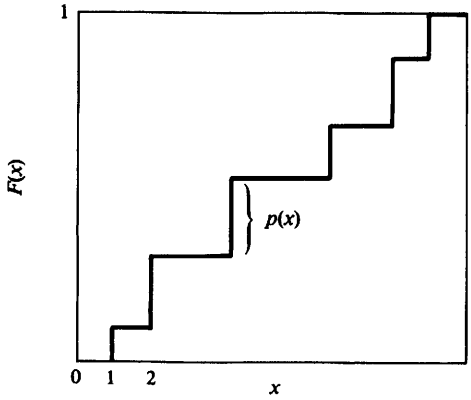


Figure 5.5. Cumulative distribution function and Shannon-Fano-Elias coding.

$$\bar{F}(x) = \sum_{a<x} p(a) + \frac{1}{2}\, p(x), \tag{5.71}$$

where $\bar{F}(x)$ denotes the sum of the probabilities of all symbols less than x plus half the probability of the symbol x. Since the random variable is discrete, the cumulative distribution function consists of steps of size $p(x)$. The value of the function $\bar{F}(x)$ is the midpoint of the step corresponding to x.

Since all the probabilities are positive, $F(a) \neq F(b)$ if $a \neq b$, and hence we can determine x if we know $\bar{F}(x)$. Merely look at the graph of the cumulative distribution function and find the corresponding x. Thus the value of $\bar{F}(x)$ can be used as a code for x.

But in general $\bar{F}(x)$ is a real number expressible only by an infinite number of bits. So it is not efficient to use the exact value of $\bar{F}(x)$ as a code for x. If we use an approximate value, what is the required accuracy?

Assume that we round off $\bar{F}(x)$ to $l(x)$ bits (denoted by $\lfloor \bar{F}(x) \rfloor_{l(x)}$). Thus we use the first $l(x)$ bits of $\bar{F}(x)$ as a code for x. By definition of rounding off, we have

$$\bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{l(x)} < \frac{1}{2^{l(x)}}. \tag{5.72}$$

If $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$, then

$$\frac{1}{2^{l(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x-1), \tag{5.73}$$

and therefore $\lfloor \bar{F}(x) \rfloor_{l(x)}$ lies within the step corresponding to x. Thus $l(x)$ bits suffice to describe x.

In addition to requiring that the codeword identify the corresponding symbol, we also require the set of codewords to be prefix-free. To check whether the code is prefix-free, we consider each codeword $z_1 z_2 \ldots z_l$ to represent not a point but the interval $[0.z_1 z_2 \ldots z_l, 0.z_1 z_2 \ldots z_l + \frac{1}{2^l}]$. The code is prefix-free if and only if the intervals corresponding to codewords are disjoint.

We now verify that the code above is prefix-free. The interval corresponding to any codeword has length $2^{-l(x)}$, which is less than half the height of the step corresponding to x by (5.73). The lower end of the interval is in the lower half of the step. Thus the upper end of the interval lies below the top of the step, and the interval corresponding to any codeword lies entirely within the step corresponding to that symbol in the cumulative distribution function. Therefore the intervals corresponding to different codewords are disjoint and the code is prefix-free.

Note that this procedure does not require the symbols to be ordered

in terms of probability. Another procedure that uses the ordered probabilities is described in Problem 25 at the end of the chapter.

Since we use $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$ bits to represent x, the expected length of this code is

$$L = \sum_x p(x)l(x) = \sum_x p(x)\left(\left\lceil \log \frac{1}{p(x)} \right\rceil + 1\right) < H(X) + 2 . \quad (5.74)$$

Thus this coding scheme achieves an average codeword length that is within two bits of the entropy.

Example 5.9.1: We first consider an example where all the probabilities are dyadic. We construct the code in the following table:

x	$p(x)$	$F(x)$	$\bar{F}(x)$	$\bar{F}(x)$ in binary	$l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.5	0.75	0.5	0.10	2	10
3	0.125	0.875	0.8125	0.1101	4	1101
4	0.125	1.0	0.9375	0.1111	4	1111

In this case, the average codeword length is 2.75 bits while the entropy is 1.75 bits. The Huffman code for this case achieves the entropy bound. Looking at the codewords, it is obvious that there is some inefficiency—for example, the last bit of the last two codewords can be omitted. But if we remove the last bit from all the codewords, the code is no longer prefix free.

Example 5.9.2: We now give another example for the construction for the Shannon-Fano-Elias code. In this case, since the distribution is not dyadic, the representation of $F(x)$ in binary may have an infinite number of bits. We denote 0.01010101 . . . by $0.\overline{01}$.

We construct the code in the following table:

x	$p(x)$	$F(x)$	$\bar{F}(x)$	$\bar{F}(x)$ in binary	$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.25	0.5	0.375	0.011	3	011
3	0.2	0.7	0.6	$0.1\overline{0011}$	4	1001
4	0.15	0.85	0.775	$0.11\overline{00011}$	4	1100
5	0.15	1.0	0.925	$0.11\overline{10110}$	4	1110

The above code is 1.2 bits longer on the average than the Huffman code for this source (Example 5.6.1).

In the next section, we extend the concept of Shannon-Fano-Elias coding and describe a computationally efficient algorithm for encoding and decoding called arithmetic coding.

5.10 ARITHMETIC CODING

From the discussion of the previous sections, it is apparent that using a codeword length of $\log \frac{1}{p(x)}$ for the codeword corresponding to x is nearly optimal in that it has an expected length within 1 bit of the entropy. The optimal codes are Huffman codes, and these can be constructed by the procedure described in Section 5.6.

For small source alphabets, though, we have efficient coding only if we use long blocks of source symbols. For example, if the source is binary, and we code each symbol separately, we must use 1 bit per symbol irrespective of the entropy of the source. If we use long blocks, we can achieve an expected length per symbol close to the entropy rate of the source.

It is therefore desirable to have an efficient coding procedure that works for long blocks of source symbols. Huffman coding is not ideal for this situation, since it is a bottom-up procedure that requires the calculation of the probabilities of all source sequences of a particular block length and the construction of the corresponding complete code tree. We are then limited to using that block length. A better scheme is one which can be easily extended to longer block lengths without having to redo all the calculations. Arithmetic coding, a direct extension of the Shannon-Fano-Elias coding scheme of the last section, achieves this goal.

The essential idea of arithmetic coding is to efficiently calculate the probability mass function $p(x^n)$ and the cumulative distribution function $F(x^n)$ for the source sequence x^n. Using the ideas of Shannon-Fano-Elias coding, we can use a number in the interval $(F(x^n) - p(x^n), F(x^n)]$ as the code for x^n. For example, expressing $F(x^n)$ to an accuracy of $\lceil \log \frac{1}{p(x^n)} \rceil$ will give us a code for the source. Using the same arguments as in the discussion of the Shannon-Fano-Elias code, it follows that the codeword corresponding to any sequence lies within the step in the cumulative distribution function (Figure 5.5) corresponding to that sequence. So the codewords for different sequences of length n are different. However, the procedure does *not* guarantee that the set of codewords is prefix-free. We can construct a prefix-free set by using $\bar{F}(x)$ rounded off to $\lceil \log \frac{1}{p(x)} \rceil + 1$ bits as in Section 5.9. In the algorithm described below, we will keep track of both $F(x^n)$ and $p(x^n)$ in the course of the algorithm, so we can calculate $\bar{F}(x)$ easily at any stage.

We now describe a simplified version of the arithmetic coding algorithm to illustrate some of the important ideas. We assume that we have a fixed block length n that is known to both the encoder and the decoder. With a small loss of generality, we will assume that the source alphabet is binary. We assume that we have a simple procedure to calculate $p(x_1, x_2, \ldots, x_n)$ for any string $x_1, x_2, \ldots, x_n$. We will use the natural lexicographic order on strings, so that a string x is greater than

a string y if $x_i = 1$, $y_i = 0$ for the first i such that $x_i \neq y_i$. Equivalently, $x > y$ if $\Sigma_i x_i 2^{-i} > \Sigma_i y_i 2^{-i}$, i.e., if the corresponding binary numbers satisfy $0.x > 0.y$. We can arrange the strings as the leaves of a tree of depth n, where each level of the tree corresponds to one bit. Such a tree is illustrated in Figure 5.6. In this figure, the ordering $x > y$ corresponds to the fact that x is to the right of y on the same level of the tree.

From the discussion of the last section, it appears that we need to find $p(y^n)$ for all $y^n \leq x^n$ and use that to calculate $F(x^n)$. Looking at the tree, we might suspect that we need to calculate the probabilities of all the leaves to the left of x^n to find $F(x^n)$. The sum of these probabilities is the sum of the probabilities of all the subtrees to the left of x^n. Let $T_{x_1 x_2 \cdots x_{k-1} 0}$ be a subtree starting with $x_1 x_2 \cdots x_{k-1} 0$. The probability of this subtree is

$$p(T_{x_1 x_2 \cdots x_{k-1} 0}) = \sum_{y_{k+1} \cdots y_n} p(x_1 x_2 \cdots x_{k-1} 0 y_{k+1} \cdots y_n) \tag{5.75}$$

$$= p(x_1 x_2 \cdots x_{k-1} 0), \tag{5.76}$$

and hence can be calculated easily. Therefore we can rewrite $F(x^n)$ as

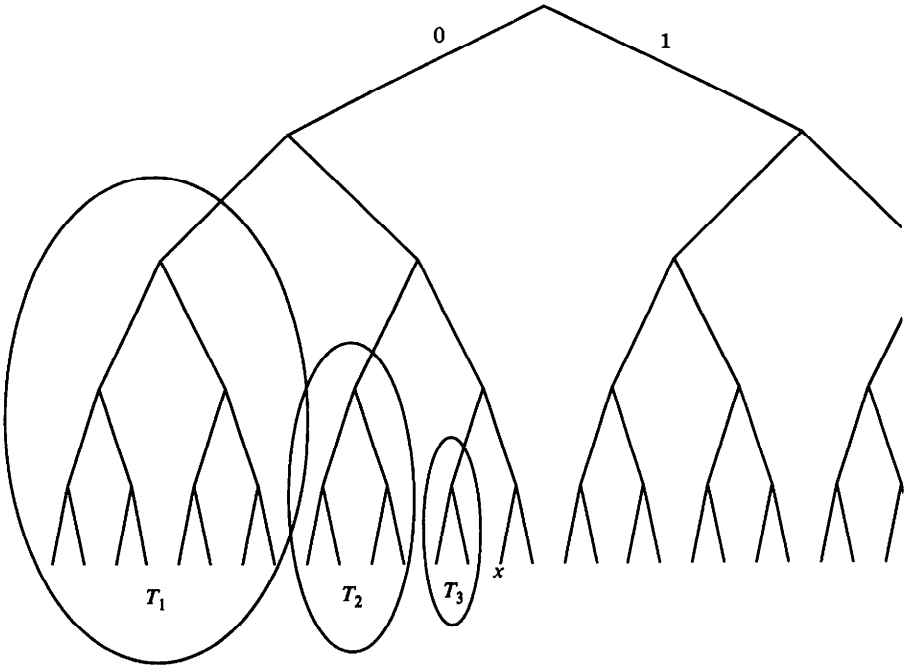


Figure 5.6. Tree of strings for arithmetic coding.

$$F(x^n) = \sum_{y^n \le x^n} p(y^n) \tag{5.77}$$

$$= \sum_{T : T \text{ is to the left of } x^n} p(T) \tag{5.78}$$

$$= \sum_{k : x_k = 1} p(x_1 x_2 \cdots x_{k-1} 0) . \tag{5.79}$$

Thus we can calculate $F(x^n)$ quickly from $p(x^n)$.

Example 5.10.1: If $X_1, X_2, \ldots, X_n$ are Bernoulli(θ) in Figure 5.6, then

$$F(01110) = p(T_1) + p(T_2) + p(T_3) = p(00) + p(010) + p(0110) \tag{5.80}$$

$$= (1-\theta)^2 + \theta(1-\theta)^2 + \theta^2(1-\theta)^2 . \tag{5.81}$$

Note that these terms can be calculated recursively. For example, $\theta^3(1-\theta)^3 = (\theta^2(1-\theta)^2)\theta(1-\theta)$.

To encode the next bit of the source sequence, we need only calculate $p(x^i x_{i+1})$ and update $F(x^i x_{i+1})$ using the above scheme. Encoding can therefore be done sequentially, by looking at the bits as they come in.

To decode the sequence, we use the same procedure to calculate the cumulative distribution function and check whether it exceeds the value corresponding to the codeword. We then use the tree in Figure 5.6 as a decision tree. At the top node, we check to see if the received codeword $F(x^n)$ is greater than $p(0)$. If it is, then the subtree starting with 0 is to the left of x^n and hence $x_1 = 1$. Continuing this process down the tree, we can decode the bits in sequence. Thus we can compress and decompress a source sequence in a sequential manner.

The above procedure depends on a model for which we can easily compute $p(x^n)$. Two examples of such models are i.i.d. sources, where

$$p(x^n) = \prod_{i=1}^{n} p(x_i) . \tag{5.82}$$

and Markov sources, where

$$p(x^n) = p(x_1) \prod_{i=2}^{n} p(x_i | x_{i-1}) . \tag{5.83}$$

In both cases, we can easily calculate $p(x^n x_{n+1})$ from $p(x^n)$.

Note that it is not essential that the probabilities used in the encoding be equal to the true distribution of the source. In some cases, such as in image compression, it is difficult to describe a "true" distribution for the source. Even then, it is possible to apply the above

arithmetic coding procedure. The procedure will be efficient only if the model distribution is close to the empirical distribution of the source (Theorem 5.4.3). A more sophisticated use of arithmetic coding is to change the model dynamically to adapt to the source. Adaptive models work well for large classes of sources. The adaptive version of arithmetic coding is a simple example of a universal code, that is, a code that is designed to work with an arbitrary source distribution. Another example is the Lempel-Ziv code, which is discussed in Section 12.10.

The foregoing discussion of arithmetic coding has avoided discussion of the difficult implementation issues of computational accuracy, buffer sizes, etc. An introduction to some of these issues can be found in the tutorial introduction to arithmetic coding by Langdon [170].

5.11 COMPETITIVE OPTIMALITY OF THE SHANNON CODE

We have shown that Huffman coding is optimal in that it has minimum expected length. But what does that say about its performance on any particular sequence? For example, is it always better than any other code for all sequences? Obviously not, since there are codes which assign short codewords to infrequent source symbols. Such codes will be better than the Huffman code on those source symbols.

To formalize the question of competitive optimality, consider the following two-person zero sum game: Two people are given a probability distribution and are asked to design an instantaneous code for the distribution. Then a source symbol is drawn from this distribution and the payoff to player A is 1 or -1 depending on whether the codeword of player A is shorter or longer than the codeword of player B. The payoff is 0 for ties.

Dealing with Huffman codelengths is difficult, since there is no explicit expression for the codeword lengths. Instead, we will consider the Shannon code with codeword lengths $l(x) = \lceil \log \frac{1}{p(x)} \rceil$. In this case, we have the following theorem:

Theorem 5.11.1: *Let $l(x)$ be the codeword lengths associated with the Shannon code and let $l'(x)$ be the codeword lengths associated with any other code. Then*

$$\Pr(l(X) \geq l'(X) + c) \leq \frac{1}{2^{c-1}} . \tag{5.84}$$

Thus, for example, the probability that $l'(X)$ is 5 or more bits shorter than $l(X)$ is less than $\frac{1}{16}$.

Proof:

$$\Pr(l(X) \geq l'(X) + c) = \Pr\left(\left\lceil \log \frac{1}{p(X)} \right\rceil \geq l'(X) + c\right) \tag{5.85}$$

$$\leq \Pr\left(\log \frac{1}{p(X)} \geq l'(X) + c - 1\right) \tag{5.86}$$

$$= \Pr(p(X) \leq 2^{-l'(X)-c+1}) \tag{5.87}$$

$$= \sum_{x\,:\,p(x) \leq 2^{-l'(x)-c+1}} p(x) \tag{5.88}$$

$$\leq \sum_{x\,:\,p(x) \leq 2^{-l'(x)-c+1}} 2^{-l'(x)-(c-1)} \tag{5.89}$$

$$\leq \sum_{x} 2^{-l'(x)} 2^{-(c-1)} \tag{5.90}$$

$$\leq 2^{-(c-1)}, \tag{5.91}$$

since $\Sigma\, 2^{-l'(x)} \leq 1$ by the Kraft inequality. □

Hence, no other code can do much better than the Shannon code most of the time.

We now strengthen this result in two ways. First, there is the term $+1$ that has been added, which makes the result non-symmetric. Also, in a game theoretic setting, one would like to ensure that $l(x) < l'(x)$ more often than $l(x) > l'(x)$. The fact that $l(x) \leq l'(x) + 1$ with probability $\geq \frac{1}{2}$ does not ensure this. We now show that even under this stricter criterion, Shannon coding is optimal. Recall that the probability mass function $p(x)$ is dyadic if $\log \frac{1}{p(x)}$ is an integer for all x.

Theorem 5.11.2: *For a dyadic probability mass function $p(x)$, let $l(x) = \log \frac{1}{p(x)}$ be the word lengths of the binary Shannon code for the source, and let $l'(x)$ be the lengths of any other uniquely decodable binary code for the source. Then*

$$\Pr(l(X) < l'(X)) \geq \Pr(l(X) > l'(X)), \tag{5.92}$$

with equality iff $l'(x) = l(x)$ for all x. Thus the code length assignment $l(x) = \log \frac{1}{p(x)}$ is uniquely competitively optimal.

Proof: Define the function $\operatorname{sgn}(t)$ as follows:

$$\operatorname{sgn}(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t = 0 \\ -1 & \text{if } t < 0 \end{cases}. \tag{5.93}$$

Then it is easy to see from Figure 5.7 that

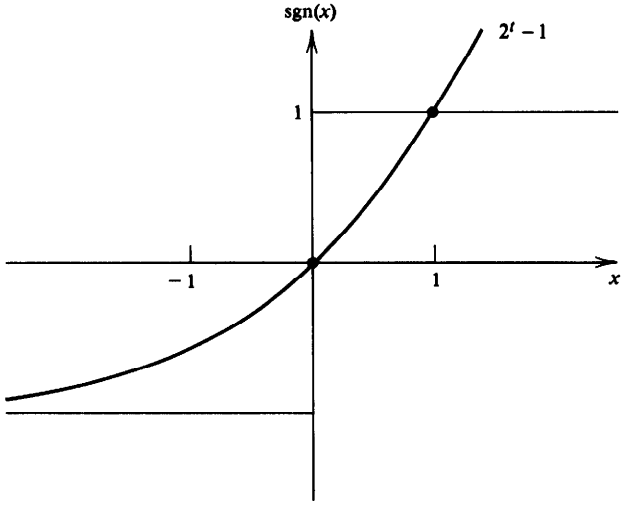


Figure 5.7. The sgn function and a bound.

$$\operatorname{sgn}(t) \le 2^t - 1 \quad \text{for } t = 0, \pm 1, \pm 2, \ldots. \tag{5.94}$$

Note that though this inequality is not satisfied for all t, it is satisfied at all integer values of t.

We can now write

$$\Pr(l'(X) < l(X)) - \Pr(l'(X) > l(X)) = \sum_{x : l'(x) < l(x)} p(x) - \sum_{x : l'(x) > l(x)} p(x) \tag{5.95}$$

$$= \sum_x p(x) \operatorname{sgn}(l(x) - l'(x)) \tag{5.96}$$

$$= E \operatorname{sgn}(l(X) - l'(X)) \tag{5.97}$$

$$\overset{(a)}{\le} \sum_x p(x)(2^{l(x)-l'(x)} - 1) \tag{5.98}$$

$$= \sum_x 2^{-l(x)}(2^{l(x)-l'(x)} - 1) \tag{5.99}$$

$$= \sum_x 2^{-l'(x)} - \sum_x 2^{-l(x)} \tag{5.100}$$

$$= \sum_x 2^{-l'(x)} - 1 \tag{5.101}$$

$$\overset{(b)}{\le} 1 - 1 \tag{5.102}$$

$$= 0, \tag{5.103}$$

where (a) follows from the bound on $\operatorname{sgn}(x)$ and (b) follows from the fact that $l'(x)$ satisfies the Kraft inequality.

We have equality in the above chain only if we have equality in (a) and (b). We have equality in the bound for $\operatorname{sgn}(t)$ only if t is 0 or 1, i.e., $l(x) = l'(x)$ or $l(x) = l'(x) + 1$. Equality in (b) implies that $l'(x)$ satisfy the Kraft inequality with equality. Combining these two facts implies that $l'(x) = l(x)$ for all x. □

Corollary: *For non-dyadic probability mass functions,*

$$E \operatorname{sgn}(l(X) - l'(X) - 1) \leq 0 \tag{5.104}$$

where $l(x) = \lceil \log \frac{1}{p(x)} \rceil$ *and* $l'(x)$ *is any other code for the source.*

Proof: Along the same lines as the preceding proof. □

Hence we have shown that Shannon coding is optimal under a variety of criteria; it is robust with respect to the payoff function. In particular, for dyadic p, $E(l - l') \leq 0$, $E \operatorname{sgn}(l - l') \leq 0$, and by use of inequality (5.94), $Ef(l - l') \leq 0$, for any function f satisfying $f(t) \leq 2^t - 1$, $t = 0, \pm 1, \pm 2, \ldots$.

5.12 GENERATION OF DISCRETE DISTRIBUTIONS FROM FAIR COINS

In the early sections of this chapter, we considered the problem of representing a random variable by a sequence of bits such that the expected length of the representation was minimized. It can be argued (Problem 26) that the encoded sequence is essentially incompressible, and therefore has an entropy rate close to 1 bit per symbol. Therefore the bits of the encoded sequence are essentially fair coin flips.

In this section, we will take a slight detour from our discussion of source coding and consider the dual question. How many fair coin flips does it take to generate a random variable X drawn according to some specified probability mass function $\mathbf{p}$? We first consider a simple example:

Example 5.12.1: Given a sequence of fair coin tosses (fair bits), suppose we wish to generate a random variable X with distribution

$$X = \begin{cases} a & \text{with probability } \frac{1}{2}, \\ b & \text{with probability } \frac{1}{4}, \\ c & \text{with probability } \frac{1}{4}. \end{cases} \tag{5.105}$$

It is easy to guess the answer. If the first bit is 0, we let $X = a$. If the first two bits are 10, we let $X = b$. If we see 11, we let $X = c$. It is clear that X has the desired distribution.

We calculate the average number of fair bits required for generating the random variable in this case as $\frac{1}{2}1 + \frac{1}{4}2 + \frac{1}{4}2 = 1.5$ bits. This is also the entropy of the distribution. Is this unusual? No, as the results of this section indicate.

The general problem can now be formulated as follows. We are given a sequence of fair coin tosses $Z_1, Z_2, \ldots$, and we wish to generate a discrete random variable $X \in \mathcal{X} = \{1, 2, \ldots, m\}$ with probability mass function $\mathbf{p} = (p_1, p_2, \ldots, p_m)$. Let the random variable T denote the number of coin flips used in the algorithm.

We can describe the algorithm mapping strings of bits $Z_1, Z_2, \ldots$, to possible outcomes X by a binary tree. The leaves of the tree are marked by output symbols X and the path to the leaves is given by the sequence of bits produced by the fair coin. For example, the tree for the distribution $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ is shown in Figure 5.8.

The tree representing the algorithm must satisfy certain properties:

1. The tree should be complete, i.e., every node is either a leaf or has two descendants in the tree. The tree may be infinite, as we will see in some examples.
2. The probability of a leaf at depth k is 2^{-k}. Many leaves may be labeled with the same output symbol—the total probability of all these leaves should equal the desired probability of the output symbol.
3. The expected number of fair bits ET required to generate X is equal to the expected depth of this tree.

There are many possible algorithms that generate the same output distribution. For example, the mapping: $00 \to a, 01 \to b, 10 \to c, 11 \to a$ also yields the distribution $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$. However, this algorithm uses two fair bits to generate each sample, and is therefore not as efficient as the mapping given earlier, which used only 1.5 bits per sample. This brings up the question: What is the most efficient algorithm to generate a given distribution and how is this related to the entropy of the distribution?

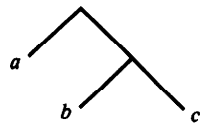


Figure 5.8. Tree for generation of the distribution $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$.

We expect that we need at least as much randomness in the fair bits as we produce in the output samples. Since entropy is a measure of randomness, and each fair bit has an entropy of 1 bit, we expect that the number of fair bits used will be at least equal to the entropy of the output. This is proved in the following theorem.

We will need a simple lemma about trees in the proof of the theorem. Let $\mathcal{Y}$ denote the set of leaves of a complete tree. Consider a distribution on the leaves, such that the probability of a leaf at depth k on the tree is 2^{-k}. Let Y be a random variable with this distribution. Then we have the following lemma:

Lemma 5.12.1: *For any complete tree, consider a probability distribution on the leaves, such that the probability of a leaf at depth k is* 2^{-k}. *Then the expected depth of the tree is equal to the entropy of this distribution.*

Proof: The expected depth of the tree

$$ET = \sum_{y \in \mathcal{Y}} k(y) 2^{-k(y)} \tag{5.106}$$

and the entropy of the distribution of Y is

$$H(Y) = -\sum_{y \in \mathcal{Y}} \frac{1}{2^{k(y)}} \log \frac{1}{2^{k(y)}} \tag{5.107}$$

$$= \sum_{y \in \mathcal{Y}} k(y) 2^{-k(y)}, \tag{5.108}$$

where $k(y)$ denotes the depth of leaf y. Thus

$$H(Y) = ET. \quad \square \tag{5.109}$$

Theorem 5.12.1: *For any algorithm generating X, the expected number of fair bits used is greater than the entropy H(X), i.e.,*

$$ET \geq H(X). \tag{5.110}$$

Proof: Any algorithm generating X from fair bits can be represented by a binary tree. Label all the leaves of this tree by distinct symbols $y \in \mathcal{Y} = \{1, 2, \ldots\}$. If the tree is infinite, the alphabet $\mathcal{Y}$ is also infinite.

Now consider the random variable Y defined on the leaves of the tree, such that for any leaf y at depth k, the probability that $Y = y$ is 2^{-k}. By Lemma 5.12.1, the expected depth of this tree is equal to the entropy of Y, i.e.,

$$ET = H(Y). \tag{5.111}$$

Now the random variable X is a function of Y (one or more leaves map

onto an output symbol), and hence by the result of Problem 5 in Chapter 2, we have

$$H(X) \leq H(Y) . \tag{5.112}$$

Thus for any algorithm generating the random variable X, we have

$$H(X) \leq ET . \quad \square \tag{5.113}$$

The same argument answers the question of optimality for a dyadic distribution.

Theorem 5.12.2: *Let the random variable X have a dyadic distribution. The optimal algorithm to generate X from fair coin flips requires an expected number of coin tosses precisely equal to the entropy, i.e.,*

$$ET = H(X) . \tag{5.114}$$

Proof: The previous theorem shows that we need at least $H(X)$ bits to generate X.

For the constructive part, we use the Huffman code tree for X as the tree to generate the random variable. For a dyadic distribution, the Huffman code is the same as the Shannon code and achieves the entropy bound. For any $x \in \mathcal{X}$, the depth of the leaf in the code tree corresponding to x is the length of the corresponding codeword, which is $\log \frac{1}{p(x)}$. Hence when this code tree is used to generate X, the leaf will have a probability $2^{-\log(1/p(x))} = p(x)$.

The expected number of coin flips is the expected depth of the tree, which is equal to the entropy (because the distribution is dyadic). Hence for a dyadic distribution, the optimal generating algorithm achieves

$$ET = H(X) . \quad \square \tag{5.115}$$

What if the distribution is not dyadic? In this case, we cannot use the same idea, since the code tree for the Huffman code will generate a dyadic distribution on the leaves, not the distribution with which we started. Since all the leaves of the tree have probabilities of the form 2^{-k}, it follows that we should split any probability p_i that is not of this form into atoms of this form. We can then allot these atoms to leaves on the tree.

To minimize the expected depth of the tree, we should use atoms with as large a probability as possible. So given a probability p_i, we find the largest atom of the form 2^{-k} that is less than p_i, and allot this atom to the tree. Then we calculate the remainder and find that largest atom that will fit in the remainder. Continuing this process, we can split all the probabilities into dyadic atoms. This process is equivalent to finding

the binary expansions of the probabilities. Let the binary expansion of the probability p_i be

$$p_i = \sum_{j \geq 1} p_i^{(j)}, \tag{5.116}$$

where $p_i^{(j)} = 2^{-j}$ or 0. Then the atoms of the expansion are the $\{p_i^{(j)} : i = 1, 2, \ldots, m, j \geq 1\}$.

Since $\Sigma_i p_i = 1$, the sum of the probabilities of these atoms is 1. We will allot an atom of probability 2^{-j} to a leaf at depth j on the tree. The depths of the atoms satisfy the Kraft inequality, and hence by Theorem 5.2.1, we can always construct such a tree with all the atoms at the right depths.

We illustrate this procedure with an example:

Example 5.12.2: Let X have the distribution

$$X = \begin{cases} a & \text{with probability } \frac{2}{3}, \\ b & \text{with probability } \frac{1}{3}. \end{cases} \tag{5.117}$$

We find the binary expansions of these probabilities:

$$\frac{2}{3} = 0.10101010\ldots_2 \tag{5.118}$$

$$\frac{1}{3} = 0.01010101\ldots_2 \tag{5.119}$$

Hence the atoms for the expansion are

$$\frac{2}{3} \rightarrow \left(\frac{1}{2}, \frac{1}{8}, \frac{1}{32}, \ldots\right) \tag{5.120}$$

$$\frac{1}{3} \rightarrow \left(\frac{1}{4}, \frac{1}{16}, \frac{1}{64}, \ldots\right) \tag{5.121}$$

These can be allotted to a tree as shown in Figure 5.9.

This procedure yields a tree that generates the random variable X. We have argued that this procedure is optimal (gives a tree of minimum

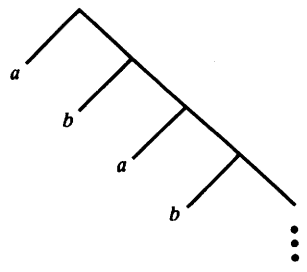


Figure 5.9. Tree to generate a $(\frac{2}{3}, \frac{1}{3})$ distribution.

expected depth), but we will not give a formal proof. Instead, we bound the expected depth of the tree generated by this procedure.

Theorem 5.12.3: *The expected number of fair bits required by the optimal algorithm to generate a random variable X lies between H(X) and H(X) + 2, i.e.,*

$$H(X) \leq ET < H(X) + 2 . \tag{5.122}$$

Proof: The lower bound on the expected number of coin tosses is proved in Theorem 5.12.1.

For the upper bound, we write down an explicit expression for the expected number of coin tosses required for the procedure described above. We split all the probabilities $(p_1, p_2, \ldots, p_m)$ into dyadic atoms, e.g.,

$$p_1 \rightarrow (p_1^{(1)}, p_1^{(2)}, \ldots) , \tag{5.123}$$

etc. Using these atoms (which form a dyadic distribution), we construct a tree with leaves corresponding to each of these atoms. The number of coin tosses required to generate each atom is its depth in the tree, and therefore the expected number of coin tosses is the expected depth of the tree, which is equal to the entropy of the dyadic distribution of the atoms. Hence

$$ET = H(Y) , \tag{5.124}$$

where Y has the distribution, $(p_1^{(1)}, p_1^{(2)}, \ldots, p_2^{(1)}, p_2^{(2)}, \ldots, p_m^{(1)}, p_m^{(2)}, \ldots)$. Now since X is a function of Y, we have

$$H(Y) = H(Y, X) = H(X) + H(Y|X) , \tag{5.125}$$

and our objective is to show that $H(Y|X) < 2$. We now give an algebraic proof of this result. Expanding the entropy of Y, we have

$$H(Y) = -\sum_{i=1}^{m} \sum_{j \geq 1} p_i^{(j)} \log p_i^{(j)} \tag{5.126}$$

$$= \sum_{i=1}^{m} \sum_{j : p_i^{(j)} > 0} j 2^{-j} , \tag{5.127}$$

since each of the atoms is either 0 or 2^{-k} for some k. Now consider the term in the expansion corresponding to each i, which we shall call T_i, i.e.,

$$T_i = \sum_{j : p_i^{(j)} > 0} j 2^{-j} . \tag{5.128}$$

We can find an n such that $2^{-(n-1)} > p_i \geq 2^{-n}$, or

$$n-1<-\log p_i \le n\,. \tag{5.129}$$

Then it follows that $p_i^{(j)}>0$ only if $j \ge n$, so that we can rewrite (5.128) as

$$T_i = \sum_{j:j\ge n,\ p_i^{(j)}>0} j2^{-j}\,. \tag{5.130}$$

We use the definition of the atom to write p_i as

$$p_i = \sum_{j:j\ge n,\ p_i^{(j)}>0} 2^{-j}\,. \tag{5.131}$$

In order to prove the upper bound, we first show that $T_i < -p_i \log p_i + 2p_i$. Consider the difference

$$T_i + p_i \log p_i - 2p_i \overset{(a)}{<} T_i - p_i(n-1) - 2p_i \tag{5.132}$$

$$= T_i - (n-1+2)p_i \tag{5.133}$$

$$= \sum_{j:j\ge n,\ p_i^{(j)}>0} j2^{-j} - (n+1) \sum_{j:j\ge n,\ p_i^{(j)}>0} 2^{-j} \tag{5.134}$$

$$= \sum_{j:j\ge n,\ p_i^{(j)}>0} (j-n-1)2^{-j} \tag{5.135}$$

$$= -2^{-n} + 0 + \sum_{j:j\ge n+2,\ p_i^{(j)}>0} (j-n-1)2^{-j} \tag{5.136}$$

$$\overset{(b)}{=} -2^{-n} + \sum_{k:k\ge 1,\ p_i^{(k+n+1)}>0} k2^{-(k+n+1)} \tag{5.137}$$

$$\overset{(c)}{\le} -2^{-n} + \sum_{k:k\ge 1} k2^{-(k+n+1)} \tag{5.138}$$

$$= -2^{-n} + 2^{-(n+1)}2 \tag{5.139}$$

$$= 0\,, \tag{5.140}$$

where (a) follows from (5.129), (b) from a change of variables for the summation and (c) from increasing the range of the summation. Hence we have shown that

$$T_i < -p_i \log p_i + 2p_i\,. \tag{5.141}$$

Since $ET = \Sigma_i T_i$, it follows immediately that

$$ET < -\sum_i p_i \log p_i + 2\sum_i p_i = H(X) + 2 \tag{5.142}$$

completing the proof of the theorem. □

SUMMARY OF CHAPTER 5

Kraft inequality: Instantaneous codes $\Leftrightarrow \sum D^{-l_i} \leq 1$

McMillan inequality: Uniquely decodable codes $\Leftrightarrow \sum D^{-l_i} \leq 1$

Entropy bound on data compression (*Lower bound*):

$$L \stackrel{\Delta}{=} \sum p_i l_i \geq H_D(X) . \tag{5.143}$$

Shannon code:

$$l_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil \tag{5.144}$$

$$L < H_D(X) + 1 . \tag{5.145}$$

Huffman code:

$$L^* = \min_{\sum D^{-l_i} \leq 1} \sum p_i l_i . \tag{5.146}$$

$$H_D(X) \leq L^* < H_D(X) + 1 . \tag{5.147}$$

Wrong code: $X \sim p(x)$, $l(x) = \lceil \log \frac{1}{q(x)} \rceil$, $L = \sum p(x) l(x)$:

$$H(p) + D(p \| q) \leq L < H(p) + D(p \| q) + 1 . \tag{5.148}$$

Stochastic processes:

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n} . \tag{5.149}$$

Stationary processes:

$$L_n \rightarrow H(\mathcal{X}) . \tag{5.150}$$

Competitive optimality: $l(x) = \lceil \log \frac{1}{p(x)} \rceil$ (Shannon code) versus any other code $l'(x)$:

$$\Pr(l(X) \geq l'(X) + c) \leq \frac{1}{2^{c-1}} . \tag{5.151}$$

Generation of random variables:

$$H(X) \leq ET < H(X) + 2 . \tag{5.152}$$

PROBLEMS FOR CHAPTER 5

1. *Uniquely decodable and instantaneous codes.* Let $L = \sum_{i=1}^{m} p_i l_i^{100}$ be the expected value of the 100th power of the word lengths associated with an encoding of the random variable X. Let $L_1 = \min L$ over all instantaneous codes; and let $L_2 = \min L$ over all uniquely decodable codes. What inequality relationship exists between L_1 and L_2?

2. *How many fingers has a Martian?* Let

$$S = \begin{pmatrix} S_1, \ldots, S_m \\ p_1, \ldots, p_m \end{pmatrix}.$$

The S_i's are encoded into strings from a D-symbol output alphabet in a uniquely decodable manner. If $m = 6$ and the codeword lengths are $(l_1, l_2, \ldots, l_6) = (1, 1, 2, 3, 2, 3)$, find a good lower bound on D. You may wish to explain the title of the problem.

3. *Slackness in the Kraft inequality.* An instantaneous code has word lengths $l_1, l_2, \ldots, l_m$ which satisfy the strict inequality

$$\sum_{i=1}^{m} D^{-l_i} < 1.$$

The code alphabet is $\mathcal{D} = \{0, 1, 2, \ldots, D-1\}$. Show that there exist arbitrarily long sequences of code symbols in $\mathcal{D}^*$ which cannot be decoded into sequences of codewords.

4. *Huffman coding.* Consider the random variable

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ 0.49 & 0.26 & 0.12 & 0.04 & 0.04 & 0.03 & 0.02 \end{pmatrix}$$

(a) Find a binary Huffman code for X.
(b) Find the expected codelength for this encoding.
(c) Find a ternary Huffman code for X.

5. *More Huffman codes.* Find the binary Huffman code for the source with probabilities (1/3, 1/5, 1/5, 2/15, 2/15). Argue that this code is also optimal for the source with probabilities (1/5, 1/5, 1/5, 1/5, 1/5).

6. *Bad codes.* Which of these codes cannot be Huffman codes for any probability assignment?
(a) $\{0, 10, 11\}$.
(b) $\{00, 01, 10, 110\}$.
(c) $\{01, 10\}$.

7. *Huffman 20 Questions.* Consider a set of n objects. Let $X_i = 1$ or 0 accordingly as the i-th object is good or defective. Let $X_1, X_2, \ldots, X_n$ be independent with $\Pr\{X_i = 1\} = p_i$; and $p_1 > p_2 > \ldots > p_n > 1/2$. We are asked to determine the set of all defective objects. Any yes-no question you can think of is admissible.

(a) Give a good lower bound on the minimum average number of questions required.

(b) If the longest sequence of questions is required by nature's answers to our questions, what (in words) is the last question we should ask? And what two sets are we distinguishing with this question? Assume a compact (minimum average length) sequence of questions.

(c) Give an upper bound (within 1 question) on the minimum average number of questions required.

8. *Simple optimum compression of a Markov source.* Consider the 3-state Markov process $U_1, U_2, \ldots$ having transition matrix

U_{n-1} \ U_n	S_1	S_2	S_3
S_1	1/2	1/4	1/4
S_2	1/4	1/2	1/4
S_3	0	1/2	1/2

Thus the probability that S_1 follows S_3 is equal to zero. Design 3 codes C_1, C_2, C_3 (one for each state S_1, S_2, S_3), each code mapping elements of the set of S_i's into sequences of 0's and 1's, such that this Markov process can be sent with maximal compression by the following scheme:

(a) Note the present symbol S_i.

(b) Select code C_i.

(c) Note the next symbol S_j and send the codeword in C_i corresponding to S_j.

(d) Repeat for the next symbol.

What is the average message length of the next symbol conditioned on the previous state $S = S_i$ using this coding scheme? What is the unconditional average number of bits per source symbol? Relate this to the entropy rate $H(\mathcal{U})$ of the Markov chain.

9. *Optimal code lengths that require one bit above entropy.* The source coding theorem shows that the optimal code for a random variable X has an expected length less than $H(X) + 1$. Give an example of a random variable for which the expected length of the optimal code is close to $H(X) + 1$, i.e., for any $\epsilon > 0$, construct a distribution for which the optimal code has $L > H(X) + 1 - \epsilon$.

10. *Ternary codes that achieve the entropy bound.* A random variable X takes on m values and has entropy $H(X)$. An instantaneous ternary code is found for this source, with average length

$$L = \frac{H(X)}{\log_2 3} = H_3(X). \tag{5.153}$$

(a) Show that each symbol of X has a probability of the form 3^{-i} for some i.

(b) Show that m is odd.

11. *Suffix condition.* Consider codes that satisfy the suffix condition, which says that no codeword is a suffix of any other codeword. Show that a suffix condition code is uniquely decodable, and show that the minimum average length over all codes satisfying the suffix condition is the same as the average length of the Huffman code for that random variable.

12. *Shannon codes and Huffman codes.* Consider a random variable X which takes on four values with probabilities $(\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{12})$.

(a) Construct a Huffman code for this random variable.

(b) Show that there exist two different sets of optimal lengths for the codewords, namely, show that codeword length assignments $(1, 2, 3, 3)$ and $(2, 2, 2, 2)$ are both optimal.

(c) Conclude that there are optimal codes with codeword lengths for some symbols that exceed the Shannon code length $\lceil \log \frac{1}{p(x)} \rceil$.

13. *Twenty questions.* Player A chooses some object in the universe, and player B attempts to identify the object with a series of yes-no questions. Suppose that player B is clever enough to use the code achieving the minimal expected length with respect to player A's distribution. We observe that player B requires an average of 38.5 questions to determine the object. Find a rough lower bound to the number of objects in the universe.

14. *Huffman code.* Find the (a) *binary* and (b) *ternary* Huffman codes for the random variable X with probabilities

$$p = \left(\frac{1}{21}, \frac{2}{21}, \frac{3}{21}, \frac{4}{21}, \frac{5}{21}, \frac{6}{21}\right).$$

(c) Calculate $L = \Sigma\, p_i l_i$ in each case.

15. *Classes of codes.* Consider the code $\{0, 01\}$

(a) Is it instantaneous?

(b) Is it uniquely decodable?

(c) Is it nonsingular?

16. *The game of Hi-Lo.*

(a) A computer generates a number X according to a known probability mass function $p(x), x \in \{1, 2, \ldots, 100\}$. The player asks a question, "Is $X = i$?" and is told "Yes", "You're too high," or "You're too low." He continues for a total of six questions. If he is right (i.e. he receives the answer "Yes") during this sequence, he receives a prize of value $v(X)$. How should the player proceed to maximize his expected winnings?

(b) The above doesn't have much to do with information theory. Consider the following variation: $X \sim p(x)$, prize $= v(x)$, $p(x)$

known, as before. But *arbitrary* Yes–No questions are asked sequentially until X is determined. ("Determined" doesn't mean that a "Yes" answer is received.) Questions cost one unit each. How should the player proceed? What is his expected return?

(c) Continuing (b), what if $v(x)$ is fixed, but $p(x)$ can be chosen by the computer (and then announced to the player)? The computer wishes to minimize the player's expected return. What should $p(x)$ be? What is the expected return to the player?

17. *Huffman codes with costs.* Words like Run! Help! and Fire! are short, not because they are frequently used, but perhaps because time is precious in the situations in which these words are required. Suppose that $X = i$ with probability p_i, $i = 1, 2, \ldots, m$. Let l_i be the number of binary symbols in the codeword associated with $X = i$, and let c_i denote the cost per letter of the codeword when $X = i$. Thus the average cost C of the description of X is $C = \sum_{i=1}^{m} p_i c_i l_i$.

(a) Minimize C over all $l_1, l_2, \ldots, l_m$ such that $\sum 2^{-l_i} \leq 1$. Ignore any implied integer constraints on l_i. Exhibit the minimizing $l_1^*, l_2^*, \ldots, l_m^*$ and the associated minimum value C^*.

(b) How would you use the Huffman code procedure to minimize C over all uniquely decodable codes? Let C_{Huffman} denote this minimum.

(c) Can you show that

$$C^* \leq C_{\text{Huffman}} \leq C^* + \sum_{i=1}^{m} p_i c_i ?$$

18. *Conditions for unique decodability.* Prove that a code C is uniquely decodable if (and only if) the extension

$$C^k(x_1, x_2, \ldots, x_k) = C(x_1)C(x_2)\cdots C(x_k)$$

is a one-to-one mapping from $\mathcal{X}^k$ to D^* for every $k \geq 1$. (The only if part is obvious.)

19. *Average length of an optimal code.* Prove that $L(p_1, \ldots, p_m)$, the average codeword length for an optimal D-ary prefix code for probabilities $\{p_1, \ldots, p_m\}$, is a continuous function of $p_1, \ldots, p_m$. This is true even though the optimal code changes discontinuously as the probabilities vary.

20. *Unused code sequences.* Let C be a variable length code that satisfies the Kraft inequality with equality but does *not* satisfy the prefix condition.

(a) Prove that some finite sequence of code alphabet symbols is not the prefix of any sequence of codewords.

(b) (Optional) Prove or disprove: C has infinite decoding delay.

21. *Optimal codes for uniform distributions.* Consider a random variable with m equiprobable outcomes. The entropy of this information source is obviously $\log_2 m$ bits.

(a) Describe the optimal instantaneous binary code for this source and compute the average codeword length L_m.

(b) For what values of m does the average codeword length L_m equal the entropy $H = \log_2 m$?

(c) We know that $L < H + 1$ for any probability distribution. The *redundancy* of a variable length code is defined to be $\rho = L - H$. For what value(s) of m, where $2^k \le m \le 2^{k+1}$, is the redundancy of the code maximized? What is the limiting value of this worst case redundancy as $m \to \infty$?

22. *Optimal codeword lengths.* Although the codeword lengths of an optimal variable length code are complicated functions of the message probabilities $\{p_1, p_2, \ldots, p_m\}$, it can be said that less probable symbols are encoded into longer codewords. Suppose that the message probabilities are given in decreasing order $p_1 > p_2 \ge \cdots \ge p_m$.

(a) Prove that for any binary Huffman code, if the most probable message symbol has probability $p_1 > 2/5$, then that symbol must be assigned a codeword of length 1.

(b) Prove that for any binary Huffman code, if the most probable message symbol has probability $p_1 < 1/3$, then that symbol must be assigned a codeword of length ≥ 2.

23. *Merges.* Companies with values $W_1, W_2, \ldots, W_m$ are merged as follows. The two least valuable companies are merged, thus forming a list of $m - 1$ companies. The *value of the merge* is the sum of the values of the two merged companies. This continues until one supercompany remains. Let V equal the sum of the values of the merges. Thus V represents the total reported dollar volume of the merges. For example, if $\mathbf{W} = (3, 3, 2, 2)$, the merges yield $(3, 3, 2, 2) \to (4, 3, 3) \to (6, 4) \to (10)$, and $V = 4 + 6 + 10 = 20$.

(a) Argue that V is the minimum volume achievable by sequences of pair-wise merges terminating in one supercompany. (*Hint*: Compare to Huffman coding.)

(b) Let $W = \Sigma W_i$, $\tilde{W}_i = W_i/W$, and show that the minimum merge volume V satisfies

$$WH(\tilde{\mathbf{W}}) \le V \le WH(\tilde{\mathbf{W}}) + W. \tag{5.154}$$

24. *The Sardinas-Patterson test for unique decodability.* A code is not uniquely decodable iff there exists a finite sequence of code symbols which can be resolved in two different ways into sequences of codewords. That is, a situation such as

A_1	A_2	A_3 ... A_m
B_1	B_2	B_3 ... B_n

must occur where each A_i and each B_i is a codeword. Note that B_1 must be a prefix of A_1 with some resulting "dangling suffix." Each dangling suffix must in turn be either a prefix of a codeword or have

another codeword as its prefix, resulting in another dangling suffix. Finally, the last dangling suffix in the sequence must also be a codeword. Thus one can set up a test for unique decodability (which is essentially the Sardinas-Patterson test [228]) in the following way: Construct a set S of all possible dangling suffixes. The code is uniquely decodable iff S contains no codeword.

(a) State the precise rules for building the set S.

(b) Suppose the codeword lengths are l_i, $i = 1, 2, \ldots, m$. Find a good upper bound on the number of elements in the set S.

(c) Determine which of the following codes is uniquely decodable:

i. $\{0, 10, 11\}$.

ii. $\{0, 01, 11\}$.

iii. $\{0, 01, 10\}$.

iv. $\{0, 01\}$.

v. $\{00, 01, 10, 11\}$.

vi. $\{110, 11, 10\}$.

vii. $\{110, 11, 100, 00, 10\}$.

(d) For each uniquely decodable code in part (c), construct, if possible, an infinite encoded sequence with a known starting point, such that it can be resolved into codewords in two different ways. (This illustrates that unique decodability does not imply finite decodability.) Prove that such a sequence cannot arise in a prefix code.

25. *Shannon code.* Consider the following method for generating a code for a random variable X which takes on m values $\{1, 2, \ldots, m\}$ with probabilities $p_1, p_2, \ldots, p_m$. Assume that the probabilities are ordered so that $p_1 \geq p_2 \geq \cdots \geq p_m$. Define

$$F_i = \sum_{k=1}^{i-1} p_i , \tag{5.155}$$

the sum of the probabilities of all symbols less than i. Then the codeword for i is the number $F_i \in [0, 1]$ rounded off to l_i bits, where $l_i = \lceil \log \frac{1}{p_i} \rceil$.

(a) Show that the code constructed by this process is prefix-free and the average length satisfies

$$H(X) \leq L < H(X) + 1 . \tag{5.156}$$

(b) Construct the code for the probability distribution (0.5, 0.25, 0.125, 0.125).

26. *Optimal codes for dyadic distributions.* For a Huffman code tree, define the probability of a node as the sum of the probabilities of all the leaves under that node. Let the random variable X be drawn from a dyadic distribution, i.e., $p(x) = 2^{-i}$, for some i, for all $x \in \mathcal{X}$. Now consider a binary Huffman code for this distribution.

(a) Argue that for any node in the tree, the probability of the left child is equal to the probability of the right child.

(b) Let $X_1, X_2, \ldots, X_n$ be drawn i.i.d. $\sim p(x)$. Using the Huffman code for $p(x)$, we map $X_1, X_2, \ldots, X_n$ to a sequence of bits $Y_1, Y_2, \ldots, Y_{k(X_1, X_2, \ldots, X_n)}$. (The length of this sequence will depend on the outcome $X_1, X_2, \ldots, X_n$.) Use part (a) to argue that the sequence $Y_1, Y_2, \ldots,$ forms a sequence of fair coin flips, i.e., that $\Pr\{Y_i = 0\} = \Pr\{Y_i = 1\} = \frac{1}{2}$, independent of $Y_1, Y_2, \ldots, Y_{i-1}$.
Thus the entropy rate of the coded sequence is 1 bit/symbol.

(c) Give a heuristic argument why the encoded sequence of bits for any code that achieves the entropy bound cannot be compressible and therefore should have an entropy rate of 1 bit per symbol.

HISTORICAL NOTES

The foundations for the material in this chapter can be found in Shannon's original paper [238], in which Shannon stated the source coding theorem and gave simple examples of codes. He described a simple code construction procedure (described in Problem 25), which he attributed to Fano. This method is now called the Shannon-Fano code construction procedure.

The Kraft inequality for uniquely decodable codes was first proved by McMillan [193]; the proof given here is due to Karush [149]. The Huffman coding procedure was first exhibited and proved to be optimal by Huffman [138].

In recent years, there has been considerable interest in designing source codes that are matched to particular applications such as magnetic recording. In these cases, the objective is to design codes so that the output sequences satisfy certain properties. Some of the results for this problem are described by Franaszek [116], Adler, Coppersmith and Hassner [2] and Marcus [184].

The arithmetic coding procedure has its roots in the Shannon-Fano code developed by Elias (unpublished), which was analyzed by Jelinek [146]. The procedure for the construction of a prefix-free code described in the text is due to Gilbert and Moore [121]. Arithmetic coding itself was developed by Rissanen [217] and Pasco [207]; it was generalized by Rissanen and Langdon [171]. See also the enumerative methods in Cover [61]. Tutorial introductions to arithmetic coding can be found in Langdon [170] and Witten, Neal and Cleary [275]. We will discuss universal source coding algorithms in Chapter 12, where we will describe the popular Lempel-Ziv algorithm.

Section 5.12 on the generation of discrete distributions from fair coin flips follows the work of Knuth and Yao [155].