Chapter 2

Lossless Source Coding

We begin this chapter by describing the general source coding scenario in Section 2.1. Section 2.2 introduces the most rudimentary kind of source codes, called fixed-length to fixed-length block. Section 2.3 introduces "lossless" source coding, which is the focus of this chapter. ("Lossy" source coding will be the focus of Chapters 11 and 12.) The subsequent sections of the chapter investigate the limits to the performance of several different kinds of lossless source codes.

2.1 Introduction to Source Coding

Source coding is the process of representing data with binary symbols in a compact and accurate way. The scenario, illustrated in Figure 2.1, is the following. A *source* generates an infinite sequence of symbols $\widetilde{U} = (U_1, U_2, \ldots)$; this is the data we wish to represent. A *source encoder* produces an infinite binary *representation* $\widetilde{Z} = (Z_1, Z_2, \ldots)$ intended for transmission or storage. A *source decoder* creates a *reproduction* $\widetilde{\widehat{U}} = (\widehat{U}_1, \widehat{U}_2, \ldots)$ of $\widetilde{U}$ from $\widetilde{Z}$ and presents it to the *user*. Together the encoder and decoder constitute a *source code.*

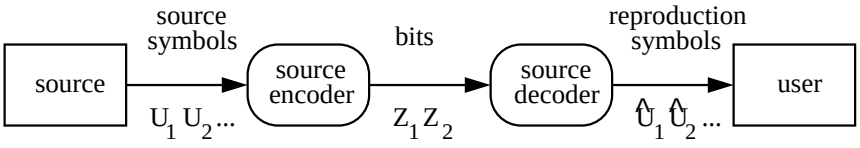


Figure 2.1: The source coding scenario.

The source symbols come from a set A_U called the *source alphabet*, and successive source outputs are modelled as random variables with this alphabet. In other words, the source is modelled as a random process, denoted $\{U_k\}$ or simply $\widetilde{U}$. Until otherwise stated, we will assume that $\widetilde{U}$ is stationary and memoryless; i.e., the U_k's are independent and identically distributed (IID).

We will adopt the conventions of Appendix A for characterizing the probability distributions of random variables. Accordingly, let $p_U(u)$ characterize the probability distribution of the U_k's. It is a probability mass function (pmf), when the U_k's are discrete, and a probability density function (pdf), when they are continuous.

The reproduction sequence $\widehat{\widetilde{U}}$ also consists of symbols from the source alphabet. The k^{th} reproduction symbol $\widehat{U}_k$ is considered to be a reproduction of the k^{th} source symbol U_k.

There are two principal aspects to the performance of a source code: *compactness* and accuracy, or *fidelity*. On the one hand, a good source code produces a compact binary representation, i.e. one with few bits, for such a representation requires minimal resources for its transmission or storage. On the other hand, for obvious reasons, a good source code produces a high fidelity reproduction, i.e. each decoder output $\widehat{U}_k$ is similar to the source symbol U_k for which it is a reproduction. Thus, when assessing source codes, there are two measures of performance: *rate*, which measures compactness, and *distortion*, which measures fidelity — actually the lack of fidelity. These are more carefully defined below.

There are actually two measures of rate, both defined in terms of the *code length function* $L_k(U_1, \ldots, U_k)$, which denotes the number of bits produced by the encoder after it receives U_k and before it receives U_{k+1} and which may depend on the previously received symbols $U_1, \ldots, U_{k-1}$. The *empirical average rate* of the code when encoding source sequence $\widetilde{U}$ is

$$\langle R \rangle \stackrel{\Delta}{=} \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} L_k(U_1, \ldots, U_k). \tag{2-1}$$

When, as is usual in this book, we have a random process model for the source data, we can also compute the *statistical average rate*

$$\overline{R} \stackrel{\Delta}{=} \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} EL_k(U_1, \ldots, U_k), \tag{2-2}$$

where E denotes expected value.

There are also two measures of distortion — empirical and statistical. Both are defined in terms of a user specified *distortion measure* d, which is a function such that $d(u, \widehat{u})$ indicates the lack of fidelity, i.e. distortion, in $\widehat{u}$ when used as a reproduction of the source symbol u. Specifically, d is a non-negative, real-valued function that maps $A_U \times A_U$ into $[0, \infty)$. Small distortion indicates good fidelity and large distortion indicates poor. The *empirical average distortion* of the code when encoding source sequence $\widetilde{U}$ is

$$\langle D \rangle \stackrel{\Delta}{=} \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} d(U_k, \widehat{U}_k). \tag{2-3}$$

And when we have a random process model for the source data, the *statistical average distortion* is

$$\overline{D} \stackrel{\Delta}{=} \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} Ed(U_k, \widehat{U}_k). \tag{2-4}$$

The two most common examples of distortion measures are Hamming distortion

$$d(u, \widehat{u}) = \begin{cases} 0, & u = \widehat{u} \\ 1, & u \neq \widehat{u} \end{cases} \tag{2-5}$$

in which case $Ed(U_k, \widehat{U}_k) = \Pr(U_k \neq \widehat{U}_k)$ so that average distortion become average error probability, and squared difference

$$d(u, \widehat{u}) = (u - \widehat{u})^2, \tag{2-6}$$

in which case average distortion is called mean squared error.

It is important to notice that the empirical average performances measures (rate and distortion) often depend on the source sequence being encoded, i.e. they can be different for different source sequences. Similarly, the statistical average performance measures often depend on the random process model for the source; i.e. they can be different for different

models. In this book we are concerned mostly with statistical average performance and the terms average rate, average distortion, rate, and distortion will mean statistical averages, unless otherwise stated. However, it is important to understand that empirical average performance is what somone using the source code would actually measure, whereas the statistical average performance is what one usually computes when designing a source code. The value in computing the latter, is that it is ordinarily a good predictor of the former. In any case, a good code is one with small average rate and distortion — empirical and/or statistical. Finally, we comment that we will typically omit the word "average" and simply say "rate" and "distortion".

It should come as no surprise that there is a conflict between compactness and fidelity. That is, it is hard to make one of them small without making the other large. In other words, there is a tradeoff between rate and distortion. Quantifying this tradeoff is one of the principal goals of our study of source coding.

Remarks

(1) We choose to focus on binary representations, as opposed to ternary or M-ary representations (for some integer M) because of their widespread appearance in transmission and storage systems. It would be equally possible to work with M-ary representations, and it is easy to convert what we learn about binary representations to M-ary representations. The decision to label the two symbols "0" and "1" is entirely arbitrary, and the only justification we offer is that it is the most widely adopted convention.

(2) In any practical system, it is always possible that some of the representation bits may be modified by noise or other phenomena before presentation to the decoder. Although this could have a significant affect on the fidelity of the code, we have not included the possibility of such "transmission errors" in our source coding scenario, because we wish to focus on the fundamental limitations of the source coding process in and of itself. However, there is one place later in this chapter where we briefly discuss the effects of errors on one type of source code, and in Chapter 10 we will see that in situations where transmission errors are prevalent we may follow the source code with a *channel code* that protects the binary representation from such transmission errors.

(3) Another important measure of the goodness of a source code is its complexity or its cost of implementation. While we shall not introduce formal measures of such, we urge the reader to consider what might be involved in implementing the various codes presented in this book. For example, how many arithmetic operations are required per source symbol for encoding and decoding? And how many symbols must be saved in auxiliary storage? From time to time we shall comment on such matters.

(4) Sometimes sources emit their symbols at regular intervals of time, for example, S_U symbols per second. While this is not always the case, it can clarify the sequential nature of the source coding process to add such an assumption to the source coding scenario. With this in mind we note that when a source with *symbol rate* S_U is encoded with a code with rate R bits per symbol, the encoder produces $S_Z = S_U R$ bits per second, which we call the *code symbol rate*. Now we see that the term "rate" could mean one of three things S_Z, S_U or R, so we need to be sure to add the appropriate modifier.

(5) There are situations where the reproduction alphabet is different than the source alphabet, for example, when color images are to be displayed on a monitor that displays only sixteen shades of gray. The theory of source coding can be extended straightforwardly to this case. However, for simplicity we have assumed that the source and reproduction alphabets are the same.

(6) There are some situations where the limits included in the definitions of rate and distortion (2-1)-(2-4) might not exist. In such cases, the conservative thing is to replace the "limit" with a "limit supremum", or "lim sup" as it is usually abbreviated.

The lim sup of a sequence $x_1, x_2, \ldots$ is the smallest number x such that for any $\epsilon > 0$ there is an integer N_o such that $x_n \leq x + \epsilon$ for all $n \geq N_o$. For example, the sequence $0, 1/2, 0, 2/3, 0, 3/4, 0, 7/8, 0, \ldots$ has no limit, but its lim sup is 1. Though we will not prove such, it turns out that for the codes, sources and distortion measures considered in this book, the limits in the definitions of statistical average rate and distortion do indeed exist. Thus, we will not need to use lim sup's.

(7) The distortions of codes defined in (2-3) and (2-4) are called *per-letter* because they average a distortion defined individually for successive symbols. We point out here that some types of infidelity cannot be adequately measured by a per-letter type distortion, no matter how the distortion measure d is chosen. For example, a per-letter average distortion cannot measure the degree to which a reproduction preserves the edges in an image or the short-term power spectra in a speech recording. Although such infidelities may indeed be quite important, information theory is primarily oriented towards per-letter distortions.

2.2 Fixed-Length to Fixed-Length Block Source Codes

Fixed-length to fixed-length block (FFB) codes are the most rudimentary source codes. We will focus on them through Section 2.6, and again in Chapters 11 and 12. An FFB code is characterized by a positive integer K called the *source length*, another positive integer L called the *code length*, a *codebook* C containing binary sequences of length L called *codewords*, a function f_e called an *encoding rule* that assigns codewords to source sequences of length K, and a function f_d called a *decoding rule*, that assigns source sequences of length K to codewords.

The code operates in the following "block fashion". See Figure 2.2. The encoder waits until K symbols have arrived from the source, forming a *block* $\underline{U}_1 = (U_1, \ldots, U_K)$. It then applies the encoding rule and produces the codeword $f_e(\underline{U}_1)$, which becomes the first L representation bits, $\underline{Z}_1 = (Z_1, \ldots, Z_L)$. These bits are transmitted or stored one by one. The encoder then waits for the next block of source symbols, $\underline{U}_2 = (U_{K+1}, \ldots, U_{2K})$, applies the encoding rule and produces the next L representation bits $\underline{Z}_2 = (Z_{L+1}, \ldots, Z_{2L}) = f_e(\underline{U}_2)$, transmits them one by one, and so on. The meaning of "in block fashion" should now be evident. The decoder operates in a similar manner. It waits for the first L representation bits $\underline{Z}_1$ applies the decoding rule f_d, produces the first K reproduction symbols $\hat{\underline{U}}_1 = (U_1, \ldots, U_K) = f_d(\underline{Z}_1)$ and presents them to the user one by one. It then waits for the next L bits $\underline{Z}_2$ decodes them producing $\hat{\underline{U}}_2 = (\hat{U}_{K+1}, \ldots, \hat{U}_{2K})$ and so on.

We will frequently refer to the rules f_e and f_d as if they *are* the encoder and decoder, respectively, instead of merely mappings that characterize their input-output law. When the source and reproduction alphabets are finite one may use tables to describe the encoding and decoding rules. For example, see Figures 2.3-2.5. One may visualize these rules with *point diagrams* such as that in Figure 2.6 for the example of Figure 2.3.

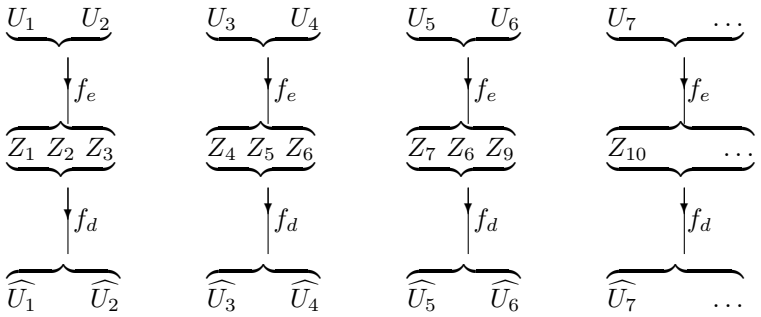


Figure 2.2: (a) The "block operation" of an FFB code with $K = 2,\ L = 3$

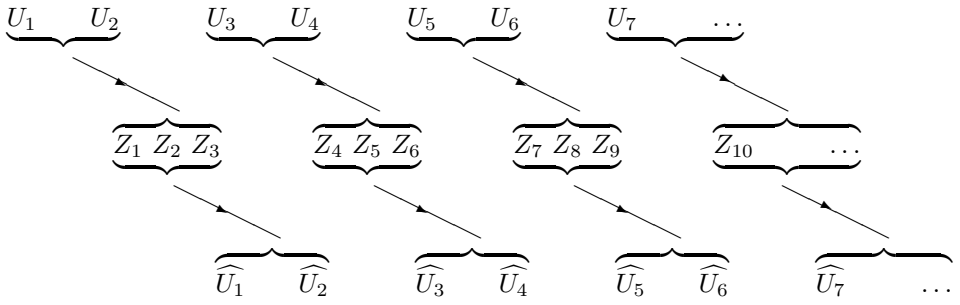


Figure 2.2: (b) The same code with time delays shown.

Encoding Rule f_e

U_1	U_2	Z_1	Z_2	Z_3	Z_4
a	a	0	0	0	0
a	b	0	0	0	1
a	c	0	0	1	0
b	a	0	0	1	1
b	b	0	1	0	0
b	c	0	1	0	1
c	a	0	1	1	0
c	b	0	1	1	1
c	c	1	0	0	0

Decoding Rule f_d

Z_1	Z_2	Z_3	Z_4	$\widehat{U}_1$	$\widehat{U}_2$
0	0	0	0	a	a
0	0	0	1	a	b
0	0	1	0	a	c
0	0	1	1	b	a
0	1	0	0	b	b
0	1	0	1	b	c
0	1	1	0	c	a
0	1	1	1	c	c
1	0	0	0	c	c

Figure 2.3: An FFB code with $K = 2,\ L = 4$.

Encoding Rule f_e

U_1	Z_1
a	0
b	0
c	1
d	1

Decoding Rule f_d

Z_1	$\widehat{U}_1$
0	a
1	c

Figure 2.4: An FFB code with $K = 1,\ L = 1$.

				z_7	0	0	0	0	1	1	1	1
				z_6	0	0	1	1	0	0	1	1
				z_5	0	1	0	1	0	1	0	1
z_1	z_2	z_3	z_4									
0	0	0	0		NUL	DLE	SP	0	@	P	’	p
0	0	0	1		BS	CAN	(	8	H	X	h	x
0	0	1	0		BOT	DC4	$	4	D	T	d	t
0	0	1	1		FF	FS	,	<	L	/	l	$\int$
0	1	0	0		STX	DC2	”	2	B	R	b	r
0	0	0	1		LF	SUB	*	:	J	Z	j	z
0	1	1	0		ACK	SYN	&	6	F	V	f	v
0	1	1	1		SO	RS	.	>	N	^	n	~
1	0	0	0		SOH	DC1	!	1	A	Q	a	q
1	0	0	1		HT	EM	)	9	I	Y	i	y
1	0	1	0		ENQ	NAK	%	5	E	U	e	u
1	0	1	1		CR	GS	-	=	M	[	m	}
1	1	0	0		EXT	DC3	#	3	C	S	c	s
1	1	0	1		VT	ESC	+	;	K	]	k	{
1	1	1	0		BEL	ETB	’	7	G	W	g	w
1	1	1	1		SI	US	/	?	O	-	o	DEL

Figure 2.5: The decoding table of the ASCII Code, which is an FFB code with $K = 1,\ L = 7$ for an alphabet with 128 symbols.

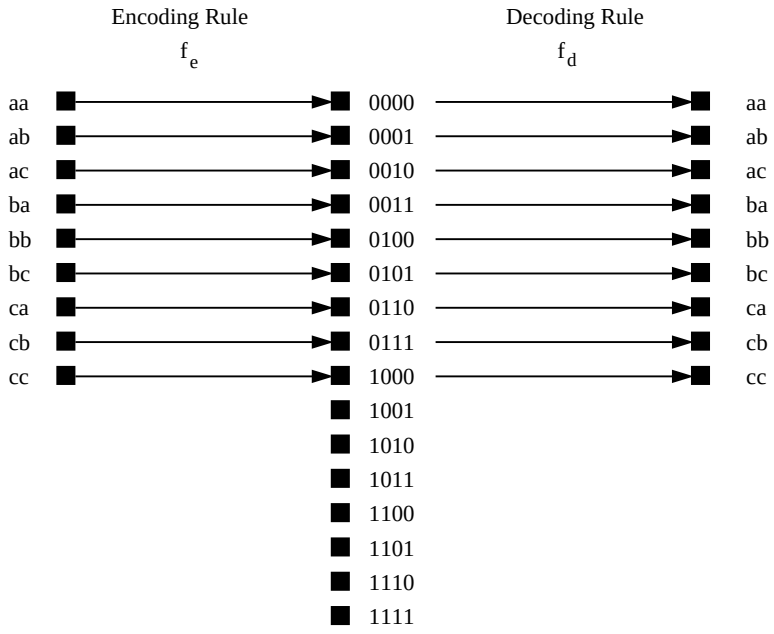


Figure 2.6: A point diagram for visualizing the encoding and decoding rules of Figure 2.3

In summary, an FFB code is characterized by a quintuple of parameters (K, L, C, f_e, f_d), where K is the source length, L is the code length, C is the codebook, f_e is the encoding rule, and f_d is the decoding rule. To keep notation short, we will often refer to a code simply by naming its codebook C. And when we wish to refer its rate and distortion, we will often write $\overline{R}(C)$ and $\overline{R}(C)$, respectively, to specify the code whose performance is under consideration.

We now discus how the formulas for performance of an FFB source code, i.e. the rate $\overline{R}$ and the distortion $\overline{D}$, simplify in the case of an FFB code. Recall that the rate of a source

code, as defined by (2-2), is the average number of representation bits per source symbol. For any FFB code with source length K and code length L, the code length function is

$$L_k(u_1, \ldots, u_k) = \begin{cases} L, & k = \text{multiple of } K \\ 0, & \text{otherwise} \end{cases} , \tag{2-7}$$

from which it is easy to see that the rate (statistical average) of an FFB code is

$$\overline{R} = \frac{L}{K} , \tag{2-8}$$

regardless of the source model. Because the rate never changes (indeed their empirical average rate is L/K as well), FFB codes are sometimes called *fixed-rate codes*.

Exercise 2.1 *Prove that empirical average rate $\langle R \rangle = L/K$, as well, for any source sequence.* □

For an FFB Code, the distortion (statistical average) defined by (2-4) simplifies to

$$\overline{D} = \frac{1}{K} \sum_{k=1}^{K} E\, d(U_k, \widehat{U}_k) . \tag{2-9}$$

Exercise 2.2 *(a) Prove (2-9). (Hint: Consider the periodicity of $E\, d(U_k, \widehat{U}_k)$.) (b) Reprove (2-9) without assuming the U_k's are IID. Instead, assume only that the U_k's form a stationary discrete-time random process. (c) Does empirical distortion also reduce to an average of K terms?* □

We conclude this section by commenting on the implementation and complexity of FFB codes. One way to implement their encoding and decoding rules is simply to store and use encoding tables, such as those shown in Figures 2.3-2.5. The principal thing to notice is that the amount of storage required for a table is proportional to the number of its rows, which is Q^K for FFB encoding or decoding. This means that the storage required for table look-up encoding and decoding increases exponentially with source length K, and indicates that complexity should be viewed as growing exponentially with source length. Thus, FFB codes can be expensive to use, unless K is kept small.

2.3 Introduction to Lossless Source Coding

Lossless source coding (also called *noiseless source coding*) is the special case of source coding in which the user demands "essentially" no distortion and asks for as small a rate as possible.

To quantify "essentially no distortion", it is customary to adopt the *Hamming distortion measure*:

$$d_H(u, \hat{u}) \triangleq \begin{cases} 0, & u = \hat{u} \\ 1, & u \neq \hat{u} \end{cases} . \tag{2-10}$$

In this case the average distortion between the k^{th} source symbol U_k and its reproduction $\hat{U}_k$ becomes the probability that they differ; i.e.,

$$E\, d_H(U_k, \widehat{U}_k) = \Pr(U_k \neq \widehat{U}_k) , \tag{2-11}$$

and distortion of the code reduces to *per-letter error probability*

$$\overline{D} = \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} \Pr(U_k \neq \widehat{U}_K) \triangleq P_{LE}. \tag{2-12}$$

For a block code with source length K, the result of Exercise 2.2 implies that this further reduces to

$$\overline{D} = P_{LE} = \frac{1}{K}\sum_{k=1}^{K} \Pr(U_k \neq \widehat{U}_k). \tag{2-13}$$

Consequently, the demand for "essentially no distortion" translates to a demand for $P_{LE} \cong 0$.

In this chapter, the main question we answer is:

Question 2.1 *What is the smallest rate of codes with $P_{LE} \cong 0$?*

As a start, in Section 2.4 we study FFB codes with P_{LE} exactly zero; these will be called *perfectly lossless*. Next in Sections 2.5 and 2.6, we will see that significantly smaller rates are attainable with FFB codes if P_{LE} is permitted to be a little large than zero. Such codes will be called *almost lossless*. Finally, in Section 2.7, we will investigate codes with variable-length codewords that are perfectly lossless, yet have the smaller rates just mentioned. In Chapters 11 and 12 we shall consider source coding at rates below those attainable by lossless source coding. Such codes introduce non-negligible amounts of distortion.

Remark

(1) Lossless coding with finite rates is impossible unless the source is discrete-valued. This is easy to establish for FFB codes (see the exercise below) and holds equally well for all other kinds of codes, including the variable-length codes considered later in this chapter. To simplify discussion, unless otherwise stated, we will assume that the source has a finite alphabet. Occasionally, however, we shall indicate how the results for finite alphabets extend to countably infinite alphabets.

Exercise 2.3 *Show that $P_{LE} = 1$ for any FFB code applied to any continous-valued source. Hint: Such codes can only have a finite number of codewords.* □

2.4 Perfectly Lossless FFB Source Codes

In this section we find the least rate of perfectly lossless fixed-length to fixed-length block codes. This is the "obvious" case and treating it explicitly will permit us to see clearly the gains of more serious source coding techniques to be presented later. Specifically, we will find

$$R^*_{PL}(K) \triangleq \min\left\{\overline{R}(C) : \begin{array}{l} C \text{ is a perfectly lossless FFB code with} \\ \text{source length } K \end{array}\right\}, \tag{2-14}$$

which is the least rate of any perfectly lossless FFB code with source length K, and

$$\begin{aligned} R^*_{PL} &\triangleq \inf\left\{\overline{R}(C) : \begin{array}{l} C \text{ is a perfectly lossless FFB code (with} \\ \text{any source length)} \end{array}\right\} \\ &= \inf\left\{R^*_{PL}(K) : K = 1, 2, \dots\right\}, \end{aligned} \tag{2-15}$$

which is the least rate of any perfectly lossless FFB code of any blocklength.[2]

As indicated in Remark (1) of the previous section, we will assume here and throughout the rest of this chapter that the source alphabet A_U is finite, specifically, having the Q symbols $\{a_1, a_2, \dots, a_Q\}$, each with nonzero probability.

[2]We write "inf" instead of "min" because there need not actually be a smallest rate r at which there is a perfectly lossless FFB code. But there will always be a smallest number r such that there exist perfectly lossless codes with rates arbitrarily close to r, and this number is called the *infimum* and denoted *inf*. For example, $\min\{x \in (0, 1]\}$ does not exist, but $\inf\{x \in (0, 1]\}$ equals 0.

In order for an FFB code with source length K to be perfectly lossless, its encoding rule must assign a distinct binary codeword of length L to each of the Q^K source sequences of length K. Since only 2^L binary sequences are available to be used as codewords, L and K must be chosen so that $2^L \geq Q^K$, or equivalently, so that $L \geq \lceil K \log_2 Q \rceil$, where $\lceil c \rceil$ denotes the smallest integer no smaller than c. It follows that the rate of any perfectly lossless FFB code with source length K and code length L is no smaller than $\lceil K \log_2 Q \rceil / K$.

Moreover, there exists a perfectly lossless FFB code with source length K and code length $L = \lceil K \log_2 Q \rceil$ and rate $\lceil K \log_2 Q \rceil / K$, because for this choice of L, $2^L \geq Q^K$. Thus it is possible to assign a distinct codeword to each source sequence.

We conclude that the least rate of perfectly lossless FFB codes with source length K is

$$R^*_{PL}(K) = \frac{\lceil K \log_2 Q \rceil}{K} \quad . \tag{2-16}$$

Since $K \log_2 Q \leq \lceil K \log_2 Q \rceil < K \log_2 Q + 1$, we obtain the following upper and lower bounds to $R^*_{PL}(K)$

$$\log_2 Q \leq R^*_{PL}(K) \leq \log_2 Q + \frac{1}{K} \quad . \tag{2-17}$$

On the one hand, the lower bound indicates that $R^*_{PL}(K)$ is never smaller than $\log_2 Q$. On the other hand, the upper bound indicates that $R^*_{PL}(K)$ becomes arbitrarily close to $\log_2 Q$ when K is large. Therefore, R^*_{PL}, which is the least rate of perfectly lossless FFB codes with any source length and which equals the infimum of $R^*_{PL}(K)$ over all positive integers K, must equal $\log_2 Q$. We summarize with the following theorem, which is the first of many "coding theorems" to appear in this book.

Theorem 2.1 (Coding Theorem for Perfectly Lossless FFB Codes) *For any source with a Q symbol alphabet, the least rate of any perfectly lossless FFB code with source length K is*

$$R^*_{PL}(K) = \frac{\lceil K \log_2 Q \rceil}{K} \quad , \tag{2-18}$$

and the least rate of any perfectly lossless FFB code with any source length is

$$R^*_{PL} = \log_2 Q \quad . \tag{2-19}$$

Each conclusion of this theorem may be decomposed into a *positive* and a *negative* statement. The positive statement corresponding to (2-18) is that there exists a perfectly lossless FFB code with source length K and rate as small as $\lceil K \log_2 Q \rceil / K$; the negative statement is that no perfectly lossless FFB code with source length K has rate less than $\lceil K \log_2 Q \rceil / K$. The positive statement corresponding to (2-19) is that there exist perfectly lossless FFB codes with rates arbitrarily close to $\log_2 Q$. The negative statement is that no perfectly lossless FFB codes have rate less than $\log_2 Q$. We will see in future sections and chapters that all coding theorems have positive and negative statements — the positive specifying that a certain degree of good performance is possible, the negative specifying that no better performance is possible.

Notice that according to the upper bound to $R^*_{PL}(K)$ in (2-17), as K increases, $R^*_{PL}(K)$ approaches $\log_2 Q$ at least as rapidly as $1/K$. However, as the following exercise shows, the approach is not always monotonic, and the upper bound can be loose or tight.

Exercise 2.4 *Assuming $Q = 3$, find R^*_{PL} and $R^*_{PL}(K)$ for $K = 1$ to 6. Does $R^*_{PL}(K)$ decrease monotonically with K? How tight is the upper bound provided by (2-17)?* □

Exercise 2.5 *For what values of Q will there be perfectly lossless FFB codes with rate exactly equal to $\log_2 Q$?* □

Example 2.1 *When English text is to be encoded, the alphabet A_U certainly contains the 26 letters $\{a, b, \ldots, z\}$. But it must also contain the symbol "space", as this too must be*

encoded. In this case, $R^*_{PL} = \log_2 27 = 4.75$ bits/character. If, in addition, we wish to distinguish capital and lower case letters, then $R^*_{PL} = \log_2 53 = 5.72$. The ASCII code shown in Figure 2.5 uses 7 bits to represent 128 different symbols, including the lower and upper case letters, space, the ten numerals 0, 1, 2, ..., 9, the standard punctuation symbols, common symbols such as %, & and a variety of computer control characters. □

Exercise 2.6 *Show that that if there exists a countably infinite number of source symbols with nonzero probability, then there can be no perfectly lossless FFB codes.* □

2.5 Almost Lossless FFB Source Codes

We now consider the possibility of designing FFB codes with rate less than $\log_2 Q$. Because of Theorem 2.1, such codes cannot be perfectly lossless, but it turns out they can have arbitrarily small error probability. In this section we will sketch the principal ideas; careful statements and proofs will be left to the next section and chapter. The main goal is to find

$$R^*_{AL} \triangleq \inf \left\{ r : \begin{array}{l} \text{for any } \delta > 0, \text{ there is an FFB code with} \\ P_{LE} \le \delta \text{ and } \overline{R} \le r \end{array} \right\}, \tag{2-20}$$

which is the precise way of defining the smallest rate at which arbitrarily small error probability is achievable[3].

We begin by examining what contributes to error probability. Given an FFB code with source length K, code length L, codebook C, encoding rule f_e and decoding rule f_d, the per-letter error probability is (by the result of Exercise 2.2)

$$P_{LE} = \frac{1}{K} \sum_{k=1}^{K} \Pr(U_k \neq \widehat{U}_k), \tag{2-21}$$

where $(\widehat{U}_1, \ldots, \widehat{U}_K) = f_d(f_e(U_1, \ldots, U_K))$. Unfortunately, it is usually rather difficult to compute P_{LE} or to make theoretical developments in terms of it. Instead, it is easier to work with the *block error probability*

$$P_{BE} \triangleq \Pr(U^K \neq \widehat{U}^K) = \Pr(U_1 \neq \widehat{U}_1 \text{ or } U_2 \neq \widehat{U}_2 \text{ or } \ldots \text{ or } U_K \neq \widehat{U}_K), \tag{2-22}$$

which is closely related to P_{LE} via

$$\frac{1}{K} P_{BE} \le P_{LE} \le P_{BE}. \tag{2-23}$$

Exercise 2.7 *Prove the above inequalities.* □

The upper bound $P_{LE} \le P_{BE}$ is especially important. For if you design a system to have small P_{BE}, the user will be comfortable knowing that P_{LE}, the real concern, is no larger. From now on we shall use P_{BE} in all further discussions of lossless block coding.

Given some FFB code, let G denote the set of *correctly encoded source sequences*; i.e. the set of source sequences of length K that are encoded and decoded without error. Formally,

$$G = \{u^K : f_d\left(f_e(u^K)\right) = u^K\}. \tag{2-24}$$

See Figure 2.7. We will show that the performance of the code is expressly related to properties of G. First, the error probability is related to the probability of G via

$$P_{BE} = \Pr(U^K \notin G) = 1 - \Pr(U^K \in G). \tag{2-25}$$

[3]It can be shown that the value of R^*_{AL} does not change if "$\overline{R} \le r + \delta$" replaces "$\overline{R} \le r$" in its definition.

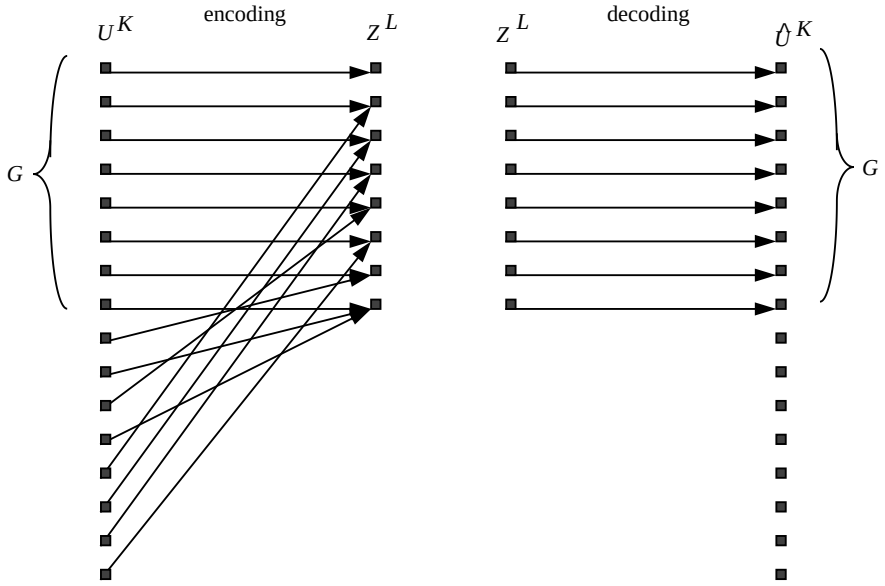


Figure 2.7: The set G of correctly encoded sequences. Each square represents one sequence.

Second, the rate of the code is related to the size of G by the fact that there must be a distinct codeword in the codebook for every correctly encoded sequence in G (otherwise they would not be correctly encoded and decoded). Since codewords are binary sequences of length L and since there are only 2^L such binary sequences, it must be that

$$|G| \leq 2^L \tag{2-26}$$

or, equivalently, that $L \geq \log_2 |G|$, where $|G|$ denotes the number of sequences in G. Consequently, the rate of the code is bounded by

$$\overline{R} = \frac{L}{K} \geq \frac{\log_2 |G|}{K}. \tag{2-27}$$

Thus we see that if one has a good code (low rate and $P_{BE} \cong 0$), then the set G of correctly encoded sequences is a "small" set with probability close to one.

Conversely, if one can find a "small" set of source sequences $\widetilde{G}$ with probability close to one, then one can use it as the basis for designing a good almost lossless FFB code (low rate and $P_{BE} \cong 0$), by choosing the encoder and decoder so that $\widetilde{G}$ becomes the correctly encoded set. Specifically, make f_e assign a distinct binary codeword of length $L = \lceil \log_2 |\widetilde{G}| \rceil$ to every sequence in $\widetilde{G}$, make f_e assign an already chosen codeword to every source sequence not in $\widetilde{G}$, and make f_d map each codeword into the source sequence from $\widetilde{G}$ that generates it. Accordingly, one obtains a code with rate $R = \lceil \log_2 |\widetilde{G}| \rceil / K$ and error probability $P_{BE} = 1 - \Pr(\widetilde{G}) \cong 0$.

From the above discussion we conclude that the key question in almost lossless FFB coding is:

Question 2.2 *How small is the smallest set of source sequences of length K with probability nearly one?*

This question can be studied apart from source coding; it is just a matter of how $p(u^K)$ distributes probability over source sequences of length K. Does it spread probability fairly uniformly, or does it mostly concentrate probability on a relatively small set, which could then be used as the basis for an almost lossless FFB code? If it concentrates probability on a set $\widetilde{G}$ whose size is significantly smaller than Q^K (the total number of source sequences of

length K), then there is an almost lossless FFB code with rate $\lceil \log_2 |\widetilde{G}| \rceil / K$, which is less than $\log_2 Q$, the least rate of perfectly lossless FFB codes.

We will show that when K is large, Question 2.2 may be answered with the law of large numbers, for example, the weak law of large numbers. A brief discussion of this law is given in Section A.7.2 of Appendix A, and a thorough discussion is given in Chapter 3. Here, we will merely state what we need and sketch the idea for its use.

Recall that our source is an IID random process $\{U_k\}$ with finite alphabet $A_U = \{a_1, \ldots, a_Q\}$ and probability mass function $p(u)$. Let p_q be a shorthand notation for $p(a_q)$. The weak law of large numbers (WLLN) shows that when K is large, the fraction of times that a symbol a_q occurs in the K random variables $U_1, \ldots, U_K$ is, with high probability, approximately equal to p_q, for every symbol a_q in the alphabet. To make this concrete, let $n_q(U^K)$ denote the number of times that a_q appears in U^K. Then the WLLN shows that for any positive number ϵ (that we ordinarily choose to be small)

$$\Pr\left(\frac{n_q(U^K)}{K} \doteq p_q \pm \epsilon,\ q = 1, \ldots, Q\right) \longrightarrow 1 \text{ as } K \longrightarrow \infty, \tag{2-28}$$

where $a \doteq b \pm \epsilon$ is shorthand for $|a - b| \leq \epsilon$ or, equivalently, $b - \epsilon \leq a \leq b + \epsilon$. In other words, when K is large, it is very likely that each symbol in the alphabet occurs in U^K with a frequency close to its probability.

Let us fix some small positive number ϵ. Like any event involving the random vector U^K, the event $\{n_q(U^K)/K \doteq p_q \pm \epsilon, \text{ for } q = 1, \ldots, Q\}$ can be expressed in the form $\{U^K \in T_K\}$, where T_K is some set of outcomes of U^K. Specifically,

$$T_K \triangleq \left\{u^K : \frac{n_q(u^K)}{K} \doteq p_q \pm \epsilon, \text{ for } q = 1, \ldots, Q\right\}. \tag{2-29}$$

Since every sequence in T_K has the property that each symbol a_q occurs with a frequency close to its probability and since this constitutes "typical" behavior, we will from now on call such sequences *typical*. In this terminology, the weak law of large numbers says that when K is large, the outcome of the random vector U^K will, with high probability, be typical; i.e., it will be one of the typical sequences in T_K. Equivalently,

$$\Pr(U^K \in T_K) \cong 1\ . \tag{2-30}$$

The careful reader will have noticed that our definition of what constitutes typical sequences, namely the set T_K, depends on the small number ϵ that we arbitrarily fixed. We will see that the conclusions we draw soon depend very little on precisely what value of ϵ is chosen.

Bearing in mind that we wish to find the smallest set with large probability and that T_K is, at least, a set with large probability, let us count how many sequences it contains. The feasibility of doing so derives from the key fact that all sequences in T_K have approximately the same probability. To demonstrate this, recall that the IID nature of the source implies that the probability of any sequence is a product of the probabilities of its components:

$$p(u^K) = p(u_1)p(u_2)\cdots p(u_K)\ . \tag{2-31}$$

Each term in this product is either p_1 or p_2 or ... or p_Q; specifically, $p(u_i) = p_q$ if $u_i = a_q$. Since $n_q(u^K)$ is the number of times a_q appears in u^K, the product may be rewritten in the form

$$p(u^K) = p_1^{n_1(u^K)} p_2^{n_2(u^K)} \cdots p_Q^{n_Q(u^K)}\ . \tag{2-32}$$

Now if u^K is typical (i.e., a member of T_K), then $n_q(u^K) \cong Kp_q$ (assuming ϵ is chosen to be small) and, consequently,

$$p(u^K) \cong p_1^{Kp_1} p_2^{Kp_2} \ldots p_Q^{Kp_Q} = \widetilde{p}^K\ , \tag{2-33}$$

where

$$\widetilde{p} \triangleq p_1^{p_1} p_2^{p_2} \ldots p_Q^{p_Q}\ , \tag{2-34}$$

which shows that each sequence in T_K has, approximately, the same probability.

Let us now return to the counting of T_K. Since each sequence in T_K has probability approximately equal to $\widetilde{p}^K$, and since T_K has probability approximately equal to one, the number of sequences in T_K must be, approximately, $1/\widetilde{p}^K$. Thus we have determined the size of T_K.

Having found its size, we now argue that T_K is, essentially, the smallest set with probability close to one. This is because the approximately $1/\widetilde{p}^K$ (typical) sequences in T_K, each having probability approximately equal to $\widetilde{p}^K$, account for essentially all of the probability in the distribution of U^K. It follows that the probability of any other set is, approximately, $\widetilde{p}^K$ times the number of typical sequences that it contains. Consequently, the only way to form a set with probability close to one is to include essentially all of the sequences of T_K (the set might also contain other sequences with very small probability). We conclude that T_K, is essentially, as small as any set with probability close to one.

We now have the complete answer to Question 2.2. When K is large, the smallest set of length K source sequences with probability close to one contains approximately $\widetilde{p}^{-K}$ sequences. Moreover, the probability distribution of U^K assigns nearly equal probability to each sequence in this set. This is often called the *asymptotic equipartition property* (AEP), because it says that asymptotically for large K the probability distribution is, essentially, equally divided among a certain set of sequences. The reader is cautioned that so far we have given only a rough statement of this result and a sketch of its derivation. Careful statements and proofs are the subject of Chapter 3, where it is formally stated and proved in the *Shannon-McMillan Theorem*.

Returning to source coding, it follows that when K is large, a perfectly lossless FFB code designed so that T_K is the set of correctly encoded sequences will have

$$P_{BE} = 1 - \Pr(U^K \in T_K) \cong 0 \tag{2-35}$$

and rate

$$\overline{R} = \frac{\lceil \log_2 |T_K| \rceil}{K} \cong \frac{\log_2 \widetilde{p}^{-K}}{K} = -\log_2 \widetilde{p} \; . \tag{2-36}$$

Since T_K is, essentially, the smallest set of length K sequences with probability close to one, $-\log_2 \widetilde{p}$ is the least rate attainable with almost lossless FFB codes. A careful statement and proof of this fact is given in the next section, where it is called the *Coding Theorem for Almost Lossless FFB codes*. Among other things it is shown there that our approximate method of counting has not lead us astray.

It is now evident that $-\log_2 \widetilde{p}$ is a very important quantity. Accordingly, it is worthwhile to find a direct expression for it:

$$\begin{aligned} -\log_2 \widetilde{p} &= -\log_2 p_1^{p_1} p_2^{p_2} \ldots p_Q p_Q \\ &= -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_Q \log_2 p_Q \; . \end{aligned} \tag{2-37}$$

Shannon decided to call this quantity the *entropy* of the source and to use the symbol H to represent it, because it has the same functional form as thermodynamical entropy.

Let us now summarize what we have shown as follows:

1. For any small number ϵ and for all large K, the set T_K of typical sequences has the properties that

$$\Pr(T_K) \cong 1 \; , \tag{2-38}$$

$$p(u^K) \cong 2^{-KH} \text{ for all } u^K \in T_K \; , \tag{2-39}$$

$$|T_K| \cong 2^{KH} \; . \tag{2-40}$$

   (These statements are made precise in the Shannon-McMillan theorem of the next subsection.)

2.

$$R^*_{AL} = H \triangleq -\sum_{q=1}^{Q} p_q \log_2 p_q \ . \tag{2-41}$$

(This is made precise in the Coding Theorem of the next subsection)

Thus the entropy H, which is a simple function of the symbol probabilities, determines size of the set of typical sequences, the probability of individual typical sequences, and the smallest possible rate of almost lossless source codes.

Although entropy will be thoroughly explored in Chapter 4, we would be remiss not to have a little discussion of it here. First, it is the sum of terms of the form $-p_q \log_2 p_q$, which are never negative, because $p_q \leq 1$. (See Figure 2.8(a) for a plot of $-p \log_2 p$.) Hence, entropy can never be negative. (One could also reach this conclusion from (2-39) or (2-40).) Second, there is the question of how to interpret $p_q \log_2 p_q$ if p_q is zero. From Figure 2.8(a) we see that $-p \log_2 p \longrightarrow 0$ as $p \longrightarrow 0$. Hence, we define $-0 \log_2 0$ to be 0. Lastly,

$$0 \leq H \leq \log_2 Q, \tag{2-42}$$

with $H = 0$ if and only if $p_q = 1$ for some q, i.e., if and only if there is no uncertainty about the outcome of U, and with $H = \log_2 Q$ if and only if $p_q = 1/Q$ for all outcomes, i.e., if and only if there is the maximum possible uncertainty about which outcome will occur. (See Exercises 2.8 and 2.9.) This suggests that H can be viewed as a measure of the amount of randomness or uncertainty in the outcome of U. In any event, we see that when the outcomes of U are not equiprobable, then $H < \log_2 Q$, and consequently, almost lossless FFB codes can outperform perfectly lossless FFB codes. As entropy places limits on the rate of codes, we take its units to be those of rate, namely, bits per source symbol.

Example 2.2 *The entropy of a binary probability distribution* $\{p, 1-p\}$*, as a function of* p*, is*

$$H = -p \log_2 p - (1-p) \log_2 (1-p) \ , \tag{2-43}$$

which is plotted in Figure 2.8. Notice that H *is a convex* $\cap$ *function of* p *(see Appendix A) that is symmetric about* $p = 1/2$ *and that increases steeply as* p *departs from either* 0 *or* 1*, reaching a peak of* 1 *at* $p = 1/2$*. For instance, if* $p = .1$*, then* $H = .47$*. This means that the least rate of almost lossless FFB codes is .47 bits per symbol. In comparison the least rate of perfectly lossless FFB codes is* $\log_2 2 = 1$ *bit per source symbol.* □

Example 2.3 *Estimates of the probabilities of the 26 letters and "space" in the English alphabet are shown in Figure 2.9. The corresponding entropy is 4.08 bits per source symbol. In comparison, it would take* $\log_2 27 = 4.75$ *bits per source symbol to encode English text with a perfectly lossless FFB code.* □

Although we know that almost lossless FFB codes can have rate as small as H, we have had no indication of how large their source lengths K need to be. To get a feeling for this, Figure 2.10 plots error probability vs. rate for the best possible FFB codes with various source lengths, and for the binary source of the previous example with $p = .1$ and $H = .47$. The figure shows that very large source lengths are needed in order that the rate be close to entropy and the error probability be very small. For example, source length 200 is needed to obtain, approximately, error probability 10^{-5} and rate .7, which is 50% larger than $H = .47$. In truth, this is somewhat disappointing, because it indicates that very large (and consequently expensive) FFB codes are needed to achieve the excellent performance predicted by this theory. Fortunately, there is an alternate approach, to be discussed in Section 2.7, that yields perfectly lossless codes at rates arbitrarily close to H with far less complexity.

Exercise 2.8 *Show that* $H = 0$ *if and only if* $p_q = 1$ *for some* q*.* □

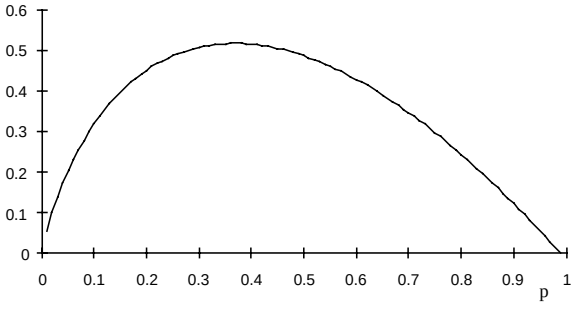


Figure 2.8: (a) $-p \log_2 p$

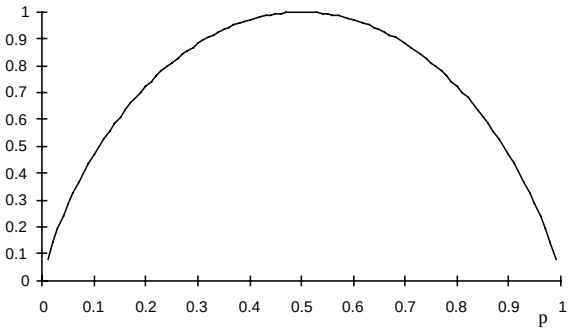


Figure 2.8: (b) Entropy of a binary variable: $H = -p \log_2 p - (1-p) \log_2 (1-p)$

Exercise 2.9 *(a) Show that $H \leq \log_2 Q$. (b) Show that $H = \log_2 Q$ if and only if the a_q's are equiprobable. (Hint: Use the relation $\ln u \leq u - 1$ with equality if and only if $u = 1$ in the sum $\sum_{q=1}^{Q} p_q \log_2 \frac{1/Q}{p_q}$.)* □

Exercise 2.10 *Find $q > 0$ such that a ternary random variable with $p(1) = p(2) = q$ and $p(3) = 1 - 2q$ has $H = 1$, the same as for a binary equiprobable random variable.* □

Exercise 2.11 *For positive integers K and n, $1 \leq n \leq K$, let $G_{K,n}$ denote the set of all binary sequences of length K with n or fewer ones. Find expressions for the block error probability and rate of an FFB code having $G_{K,n}$ as its set of correctly encoded sequences. These probablities and rate are what are plotted in Figure 2.10.* □

Exercise 2.12 *(From Gallager) An IID binary source has $p_0 = .995$ and $p_1 = .005$. An almost lossless FFB code is to be designed with source length $K = 100$ such that the set of correctly encoded sequences contains all sequences with 3 or fewer 1's.*
(a) Find the minimum possible rate of such a code.
(b) Find the block error probability P_{BE}.
(c) Use the Chebychev inequality (A-136) to find an upper bound to P_{BE} and compare the result to that of part (b). (If there is some leeway in how the inequality can be applied, apply it so as to get the smallest upper bound.) □

Symbol	Probability	Symbol	Probability
A	.0642	O	.0632
B	.0127	P	.0152
C	.0218	Q	.0008
D	.0317	R	.0484
E	.1031	S	.0514
F	.0208	T	.0796
G	.0152	U	.0228
H	.0467	V	.0083
I	.0575	W	.0175
J	.0008	X	.0013
K	.0049	Y	.0164
L	.0321	Z	.0005
M	.0198	Space	.1859
N	.0574		

Figure 2.9: Frequencies of English letters: $H = -\sum_{j=1}^{27} p_j \log_2 p_j = 4.08$ bits.

2.6 The Coding Theorem for Almost Lossless FFB Source Codes

In the previous section we learned from the asymptotic equipartition property that for large K the smallest set of length K source sequences with probability close to one contains approximately 2^{KH} sequences, where H is the entropy of the source. This fact was then used to argue that H is the least rate of any FFB code with small error probability. This important result about almost lossless coding is made precise in Theorem 2.2, whose statement and proof are the topic of this section.

In order to state the theorem, let us define $P^*_{BE}(r, K)$ to be the smallest block error probability of any FFB source code with source length K and rate less than or equal to r. That is,

$$P^*_{BE}(r,K) \triangleq \inf\left\{ P_{BE}(C) : \begin{array}{l} C \text{ is an FFB code with source length } K \\ \text{and rate } \overline{R} \le r \end{array} \right\} . \tag{2-44}$$

Theorem 2.2 (Coding Theorem for Almost Lossless FFB Source Codes) *Let $\widetilde{U}$ be an IID source with entropy H.*

(a) Positive statement: For any $r > H$,

$$P^*_{BE}(r,K) \longrightarrow 0 \text{ as } K \longrightarrow \infty \tag{2-45}$$

which implies

$$R^*_{AL} \le H \tag{2-46}$$

(b) Negative statement (converse): For any $r < H$

$$P^*_{BE}(r,K) \longrightarrow 1 \text{ as } K \longrightarrow \infty \tag{2-47}$$

In effect, the positive statement says that for large values of K, there are almost lossless FFB codes with source length K whose rate is arbitrarily close to H, and whose block error probability is arbitrarily small. It does not, however, tell us whether there are almost lossless codes with even smaller rates. This is the role of the negative statement (or converse), which says that codes with rate less than H and large source length have large block error probability.

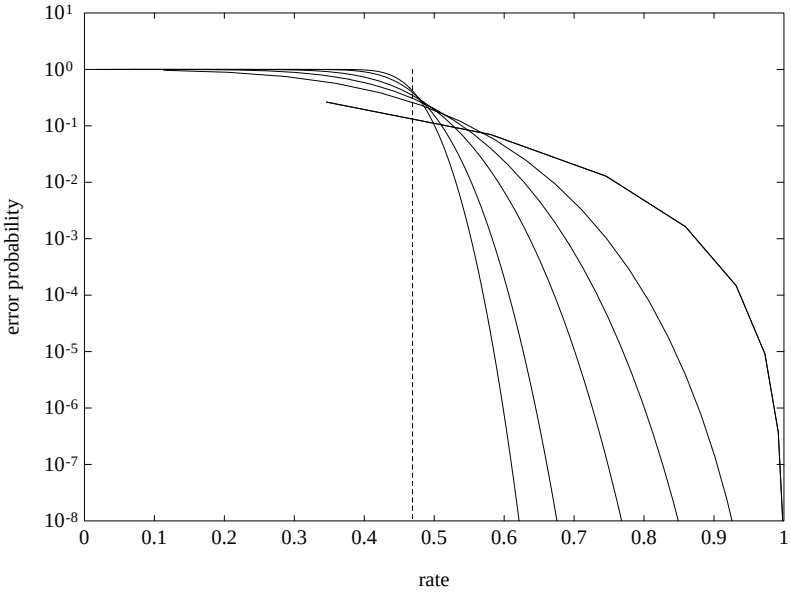


Figure 2.10: Block error probability vs. rate for the best FFB codes for a binary IID source with Pr(1)=.1. From right to left, the plotted curves correspond to source lengths $K = 10, 50, 100, 200, 500, 1000$. The dashed line indicates the entropy, $H = .47$ bits/symbol.

This theorem does not entirely answer the question of what is the least rate of almost lossless block codes, i.e. it does not completely specify R^*_{AL}, because the converse leaves open the possibility that for small source lengths, there may be almost lossless codes with rate less than H. It also leaves open the possibility that codes with rate less than H (with large or small source lengths) might have small per-letter error probability. (Recall that the latter can be less than block error probability.) A complete answer to the question must be postponed to Chapter 5, where it is shown that all codes with rate less than H (with large or small source length) have per-letter error probability bounded from below by a monotonic function of rate that is strictly greater than 0. (See Theorems 5.8.1 and 5.8.2.)

As previously indicated, the proof of this theorem makes use of the asymptotic equipartition property, which was sketched in the previous section and will be the principal topic of Chapter 3. For convenience, the version we need (from Chapter 3) is carefully stated below.

Theorem 2.3 (The Shannon-McMillan Theorem) *Let U be an IID source with entropy H.*

(a) Positive statement: For any $\epsilon > 0$ and positive integer K, there exists a set T^K_ϵ containing source sequences of length K and [4]

$$\text{(i)} \quad \Pr(U^K \in T^K_\epsilon) \longrightarrow 1 \text{ as } K \longrightarrow \infty, \tag{2-48}$$

$$\text{(ii)} \quad p(u^K) \doteq 2^{-K(H \pm \epsilon)}, \text{ for all } u^K \in T^K_\epsilon, \tag{2-49}$$

$$\text{(iii)} \quad |T^K_\epsilon| \doteq \Pr\left(U^K \in T^K_\epsilon\right) 2^{K(H \pm \epsilon)} \tag{2-50}$$

(b) Negative statement (converse): For any $\epsilon > 0$, there is a positively valued sequence $a_{\epsilon,K}$ that converges to zero as $K \longrightarrow \infty$ such that for any positive integer K and any set S

[4] The notation $b \doteq f(a \pm \epsilon)$ means

$$\min_{-\epsilon \le \delta \le \epsilon} f(a+\delta) \le b \le \max_{-\epsilon \le \delta \le \epsilon} f(a+\delta)$$

containing source sequences of length K,

$$|S| \geq \left(\Pr(U^K \in S) - a_{\epsilon,K}\right) 2^{K(H-\epsilon)}. \tag{2-51}$$

Proof of Theorem 2.2

(a) Positive statement

Let us fix a number $r > H$. To show, as we must, that $P^*_{BE}(r,K) \longrightarrow 0$ as $K \longrightarrow \infty$, we will construct a sequence of FFB codes with increasing source lengths such that the code with source length K has block error probability, denoted $P_{BE,K}$, going to zero as $K \longrightarrow \infty$ and rate, denoted $\overline{R}_K$, that is less than or equal to r for all sufficiently large K. Since these codes have $\overline{R}_K \leq r$ for all sufficiently large K, it must be that $P^*_{BE}(r,K) \leq P_{BE,K}$ for all sufficiently large K. And since $P_{BE,K}$ tends to zero as $K \longrightarrow \infty$, so must $P^*_{BE}(r,K)$ tend to zero, which will complete the proof.

To show the existence of a suitable sequence of FFB codes, let us apply the Positive Statement of the Shannon-McMillan Theorem with $\epsilon = (r-H)/2$. It shows that for every positive integer K there is a set T^K_ϵ of source sequences of length K such that (2-48)-(2-50) hold.

As in the previous section, for any K let us design an FFB code with source length K so that T^K_ϵ becomes the set of correctly encoded source sequences. That is, we make the encoder f_e assign a distinct binary codeword of length $L = \lceil \log_2 |T^K_\epsilon| \rceil$ to each sequence in T^K_ϵ, make f_e assign an already chosen codeword to each source sequence not in T^K_ϵ, and make f_d map each codeword into the source sequence from T^K_ϵ that generates it. The encoding rule is pictured in Figure 2.11. In this way, for all K we obtain a code with block error probability

$$P_{BE,K} = 1 - \Pr(U^K \in T^K_\epsilon)\ , \tag{2-52}$$

which goes to zero as $K \longrightarrow \infty$ by (2-48). The rate of this code is

$$\begin{aligned}
\overline{R}_K &= \frac{L}{K} = \frac{\lceil \log_2 |T^K_\epsilon| \rceil}{K} < \frac{\log_2 |T^K_\epsilon| + 1}{K} \\
&\leq \frac{K(H+\epsilon)+1}{K} = H + \epsilon + \frac{1}{K} \\
&\leq H + 2\epsilon \quad \text{for all sufficiently large } K \\
&= r\ ,
\end{aligned} \tag{2-53}$$

where the second inequality used (2-50) and the fact that $\Pr(U^K \in T^K_\epsilon) \leq 1$, and where the last equality used the fact that $\epsilon = (r-H)/2$. This shows what we set out to prove and, therefore, completes the proof of the positive statement.

Now recall the definition of R^*_{AL} in (2-20). To demonstrate that $R^*_{AL} \leq H$, we will show that every number r greater than H is a member of the set whose inf is R^*_{AL}. Specifically, we will show that if $r > H$, then for any $\delta > 0$, there is an FFB code with $\overline{R} \leq r$ and $P_{LE} \leq \delta$. However, this last statement follows directly from what we have already shown: we have found a sequence of FFB codes whose rates $\overline{R}$ become less than or equal to r and whose block error probabilities P_{BE} tend to zero as their source lengths K grow to infinity, and since $P_{LE} \leq P_{BE}$, they also have P_{BE} tending to zero. Thus, for any $\delta > 0$, when K is sufficiently large, $\overline{R} \leq r$ and $P_{LE} \leq \delta$. Therefore, every number r greater than H is a member of the set defining R^*_{AL}, and so $R^*_{AL} \leq H$.

(b) Negative statement

Let us fix a number $r < H$. To show, as we must, that $P^*_{BE}(r,K) \longrightarrow 1$ as $K \longrightarrow \infty$, we will find a lower bound to the block error probability of every FFB code with source length K and rate r or less that tends to one as $K \longrightarrow \infty$.

Let us apply the Negative Statement of the Shannon-McMillan Theorem with $\epsilon = (H-r)/2$. It shows there exists a positively valued sequence $a_{\epsilon,K}$ that converges to zero as

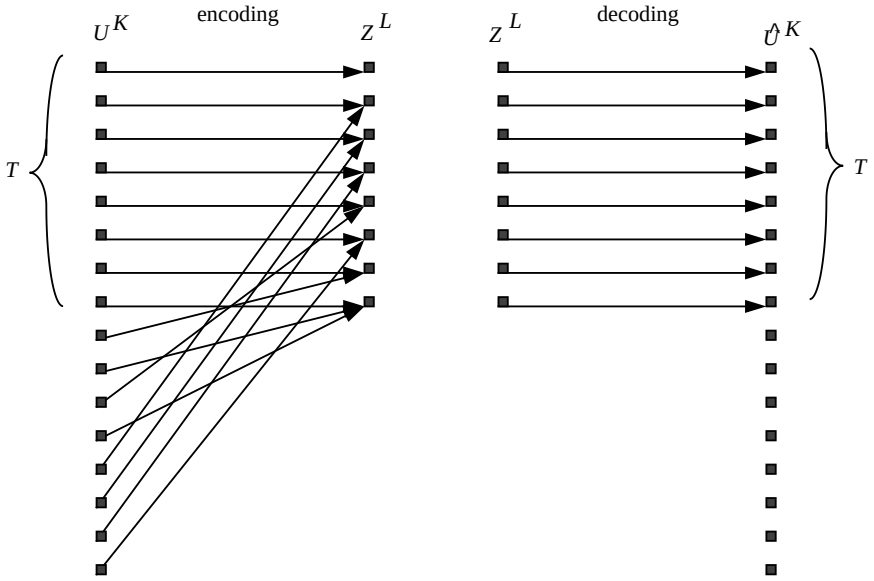


Figure 2.11: A code with T as the set of correctly encoded sequences. Each square represents one sequence.

$K \longrightarrow \infty$ such that for any positive integer K and any set S containing source sequences of length K,

$$|S| \geq \left(\Pr(U^K \in S) - a_{\epsilon,K}\right) 2^{K(H-\epsilon)} \ . \tag{2-54}$$

Equivalently,

$$\Pr(U^K \in S) \leq |S| 2^{-K(H-\epsilon)} + a_{\epsilon,K} \tag{2-55}$$

Now consider an arbitrary FFB code with source length K, code length L, rate $R = L/K \leq r$, encoding rule f_e and decoding rule f_d. Let G denote the correctly encoded set of source sequences; i.e., $G = \{u^K : f_d(f_e(u^K)) = u^K\}$. Then as argued in the previous section, the code's block error probability is

$$P_{BE} = 1 - \Pr(U^K \in G), \tag{2-56}$$

and the number of sequences in G can be no larger 2^L, the number of distinct binary sequences of length L. Hence,

$$|G| \leq 2^L = 2^{KR} \leq 2^{Kr} \ . \tag{2-57}$$

Substituting G for S in (2-55) and using the above bound on $|G|$ gives

$$\begin{aligned} \Pr(U^K \in G) &\leq 2^{Kr} 2^{-K(H-\epsilon)} + a_{\epsilon,K} \\ &= 2^{-K(H-r-\epsilon)} + a_{\epsilon,K} \\ &= 2^{-K\epsilon} + a_{\epsilon,K} \ , \end{aligned} \tag{2-58}$$

where the last equality used $\epsilon = (H-r)/2$. Finally, using the above yields

$$\begin{aligned} P_{BE} &= 1 - \Pr(U^K \in G) \\ &\geq 1 - 2^{-K\epsilon} - a_{\epsilon,K} \ . \end{aligned} \tag{2-59}$$

Notice that the right hand side of the above converges to one as K goes to ∞. Thus, as we set out to do, we have found a lower bound to the block error probability of every FFB code with source length K and rate r or less, which converges to one. This completes the proof of the negative statement and the entire theorem. □

Remarks

(1) Notice that the approximations for $p(u^K)$ and $|T_\epsilon^K|$ given in the Shannon-McMillan theorem are really quite loose because $2^{K\epsilon}$ grows to infinity as K increases. However, since the rate of the code based on T_ϵ^K is the logarithm of $|T_\epsilon^K|$ divided by K, these loose bounds were sufficient to prove the important result contained in the coding theorem.

(2) A simpler and in some respects stronger negative statement, called the *per-letter converse to the lossless source coding theorem,* will be given in Chapter 5.

(3) Although the results of this section show that almost lossless FFB codes can reduce the rate to, approximately, H (which in some cases is a big reduction), unfortunately the source lengths required to achieve this reduction are not small. For example, they may be on the order of 50 to 100. Since an FFB code needs to store the 2^{Kr} correctly encoded sequences, we see that this method is too complex for practical implementation, when for example $K = 50$ and $r = 1$.

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

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

Example 2.4 *Consider the encoding table shown below for an IID source $\widetilde{U}$ with alphabet $A_U = \{a, b, c\}$, probabilities $p_a = 1/2$, $p_b = 1/4$, $p_c = 1/4$, and entropy $H = 1.5$.*

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

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

In general, a perfectly lossless FVB code is characterized by its source length K, its codebook $C = \{ \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_{Q^K} \}$, where the i^{th} codeword $\underline{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iL_i})$ is a binary sequence with length denoted L_i, its encoding rule f_e assigning codewords in C to source sequences of length K, and its decoding rule f_d assigning source sequences of length K to codewords. As with an FFB code, the encoder operates in "block fashion".

It applies f_e to the first block, $\underline{U}_1 = (U_1, \ldots, U_K)$, produces a binary sequence denoted $\underline{Z}_1 = f_e(U_1, \ldots, U_K)$, then applies f_e to the next block, $\underline{U}_2 = (U_{K+1}, \ldots, U_{2K})$, produces the binary sequence $\underline{Z}_2 = f_e(U_{K+1}, \ldots, U_{2K})$, and so on.

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

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

It is helpful to visualize the decoding with the aid of a binary tree. For example, see Figure 2.12. Upward branches of the tree are identified with 0's and downward branches with 1's. Each codeword, $\underline{v} = (v_1, \ldots, v_L)$, indicates a path through the tree from left to right, with the i^{th} component v_i indicating whether the ith branch of the path is up or down. As a result, each codeword is associated with the node reached at the end of its path. Once this tree is specified, one may envision the decoding process as using the bits $Z_1, Z_2, \ldots$ to generate a path through the tree. When a node associated with some codeword is reached, one has found $Z_1, \ldots, Z_{J_1}$ and applies the decoding rule f_d. One then returns to the root node of the tree and uses the remaining bits $Z_{J_1+1}, Z_{J_1+2}, Z_{J_1+3}, \ldots$ to generate a path through the tree to the next codeword, and so on.

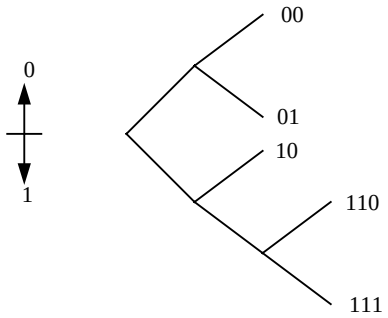


Figure 2.12: Tree diagram of the prefix code $C = \{00, 01, 10, 110, 111\}$.

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

$$\overline{R} = \frac{\overline{L}}{K} = \frac{1}{K} \sum_{u^K} p(u^K) L(u^K) \ , \tag{2-60}$$

where $L(u^K)$ denotes the length of the codeword $f_e(u^K)$ assigned to u^K, $p(u^K)$, as usual, denotes the probability of the source sequence u^K, and the sum is over all possible u^K's.

The principal goal of this section is to find

$$R^*_{VL}(K) \stackrel{\Delta}{=} \min \left\{ \overline{R}(C) : \begin{array}{l} C \text{ is a perfectly lossless FVB prefix code} \\ \text{with source length } K \end{array} \right\} , \tag{2-61}$$

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

$$R^*_{VL} \stackrel{\Delta}{=} \inf \left\{ \overline{R}(C) : \begin{array}{l} C \text{ is a perfectly lossless FVB prefix code} \\ \text{(with any source length)} \end{array} \right\} \tag{2-62}$$

$$= \inf \{ R^*_{VL}(K) : K = 1, 2, \dots \} , \tag{2-63}$$

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

Question 2.3 *How does one design an FVB prefix code?*

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

Codes with source length $K = 1$

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

$$L_q \cong -\log_2 p_q, \tag{2-64}$$

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

$$\overline{L} = \sum_{q=1}^{Q} p_q L_q \cong -\sum_{q=1}^{Q} p_q \log_2 p_q = H. \tag{2-65}$$

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

Question 2.4 *Does there actually exist a prefix code with lengths $L_q \cong -\log_2 p_q$?*

Question 2.5 *Could there be prefix codes with even smaller rates?*

Both of these questions may be answered using the following.

Theorem 2.4 (The Kraft inequality theorem) *There exists a binary prefix code with lengths $\{L_1, L_2, \dots, L_Q\}$ if and only if*

$$\sum_{q=1}^{Q} 2^{-L_q} \leq 1. \tag{2-66}$$

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

Proof

Let us first show that if $\{\underline{v}_1, \dots, \underline{v}_Q\}$ is a prefix code with lengths $\{L_1, \dots, L_Q\}$, then $\sum_{q=1}^{Q} 2^{-L_q} \leq 1$. Let L_{max} denote the length of the longest codeword.

We proceed by counting the number of binary sequences of length L_{max} that are prefixed by one codeword or another, and by comparing this number to $2^{L_{max}}$, the total number of binary sequences of length L_{max}. Specifically, the q-th codeword $\underline{v}_q$ is a prefix of $2^{L_{max}-L_q}$ binary sequences of length L_{max}. Since the code has the prefix-free property, no sequence of length L_{max} is prefixed by more than one codeword. Hence, the total number of sequences prefixed by some codeword is $\sum_{q=1}^{Q} 2^{L_{max}-L_q}$ and since this can be no larger than $2^{L_{max}}$, we have (after multiplying by $2^{-L_{max}}$)

$$\sum_{q=1}^{Q} 2^{-L_q} \leq 1 \ , \tag{2-67}$$

which is the Kraft inequality.

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

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

$$\sum_{q=1}^{n} 2^{L_{n+1}-L_q} = 2^{L_{n+1}} \sum_{q=1}^{n} 2^{-L_q} < 2^{L_{n+1}} \sum_{q=1}^{Q} 2^{-L_q} \leq 2^{L_{n+1}}; \tag{2-68}$$

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

Let us now use the Kraft Inequality Theorem to answer Question 2.4 by showing there are prefix codes with lengths $L_q \cong -\log_2 p_q$. Since $-\log_2 p_q$ need not be an integer, let us choose

$$L_q = \lceil -\log_2 p_q \rceil \ , \ q = 1, \ldots, Q \ . \tag{2-69}$$

To see that there is indeed a prefix code with these lengths, we need only check that they satisfy the Kraft inequality (2-66). Using the fact that

$$\lceil -\log_2 p_q \rceil \geq -\log_2 p_q, \tag{2-70}$$

we find

$$\sum_{q=1}^{Q} 2^{-L_q} \ = \ \sum_{q=1}^{Q} 2^{-\lceil -\log_2 p_q \rceil} \ \leq \ \sum_{q=1}^{Q} 2^{\log_2 p_q} \ = \ \sum_{q=1}^{Q} p_q = 1 \ , \tag{2-71}$$

which demonstrates that the Kraft inequality holds. Therefore, there does indeed exist a prefix code with lengths $L_q = \lceil -\log_2 p_q \rceil$, and this answers Question 2.4. One may find such a code simply by following the brute force procedure described in the second half of the proof of the Kraft inequality theorem. That is, choose $\underline{v}_1$ to be any binary sequence of length L_1, choose $\underline{v}_2$ be any binary sequence of length L_2 not prefixed by $\underline{v}$, and so on. The resulting codes are called *Shannon-Fano codes*.

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

$$\lceil -\log_2 p_q \rceil < -\log_2 p_q + 1, \tag{2-72}$$

we find

$$\begin{aligned} \overline{L} &= \sum_{q=1}^{Q} p_q L_q = \sum_{q=1}^{Q} p_q \lceil -\log_2 p_q \rceil < -\sum_{q=1}^{Q} p_q \log_2 p_q + \sum_{q=1}^{Q} p_q \\ &= H + 1 \ . \end{aligned} \tag{2-73}$$

Similarly, using the inequality

$$\lceil -\log_2 p_q \rceil \geq -\log_2 p_q \ , \tag{2-74}$$

we find

$$\begin{aligned} \overline{L} &= \sum_{q=1}^{Q} p_q L_q = \sum_{q=1}^{Q} p_q \lceil -\log_2 p_q \rceil \geq \sum_{q=1}^{Q} -p_q \log_2 p_q \\ &= H \ . \end{aligned} \tag{2-75}$$

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

$$H \leq \overline{L} < H + 1 \ . \tag{2-76}$$

We now answer Question 2.5 by showing that no prefix code with source length 1 can have average length smaller than H. To do this we make use of the elementary inequality

$$\ln x \leq x - 1 \ , \tag{2-77}$$

($\ln x$ denotes the natural logarithm of x), which is illustrated in Figure 2.13 and which is the basis of many important inequalities in information theory. Let $\{L_1, \ldots, L_Q\}$ be the lengths of any prefix code whatsoever. To show that $\overline{L}$ must be larger than H, consider their difference. We find

$$\begin{aligned} \overline{L} - H &= \sum_{q=1}^{Q} p_q L_q + \sum_{q=1}^{Q} p_q \log_2 p_q \\ &= -\sum_{q=1}^{Q} p_q \log_2 \left(\frac{2^{-L_q}}{p_q} \right) = -\sum_{q=1}^{Q} p_q \ln \left(\frac{2^{-L_q}}{p_q} \right) \frac{1}{\ln 2} \\ &\geq -\sum_{q=1}^{Q} p_q \left(\frac{2^{-L_q}}{p_q} - 1 \right) \frac{1}{\ln 2} = -\sum_{q=1}^{Q} \left(2^{-L_q} - p_q \right) \frac{1}{\ln 2} \\ &\geq -(1-1)\frac{1}{\ln 2} = 0 \ , \end{aligned} \tag{2-78}$$

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

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

Lemma 2.5 *Consider a finite probability distribution $\{p_1, p_2, \ldots, p_Q\}$.*

(a) There exists a prefix code (in particular a Shannon-Fano code) with lengths $\{L_1, L_2, \ldots, L_Q\}$ such that

$$\overline{L} < H + 1 \ . \tag{2-79}$$

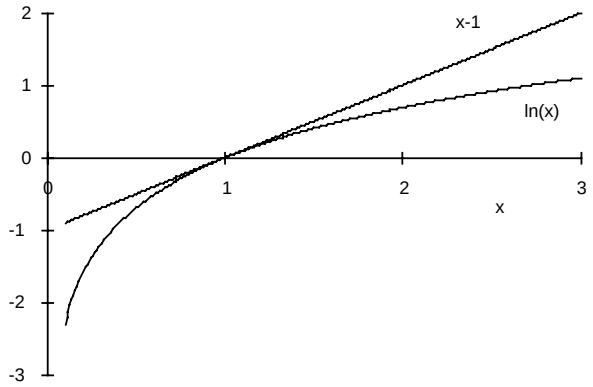


Figure 2.13: $\ln(x)$ and $x - 1$.

(b) For any prefix code whatsoever,

$$\overline{L} \geq H \ . \tag{2-80}$$

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

$$H \leq \overline{L}^* < H + 1 \ . \tag{2-81}$$

The following exercise helps to clarify the limitations of Shannon-Fano codes.

Exercise 2.13 *(a) Find a finite probability distribution for which Shannon-Fano codes have average length at least .9 bits larger than H. This shows that the average length of a Shannon-Fano code can be close to $H + 1$. (Hint: One need only consider a binary source.)*
(b) Find a finite probability distribution for which Shannon-Fano codes do not have the smallest possible average length; i.e. their average length is greater than $\overline{L}^*$. (Hint: One need only consider a binary source.)
(c) Find a finite probability distribution for which all prefix codes have average length at least .9 bits larger than H. This shows that $\overline{L}^*$ can be close to $H + 1$. (Hint: One need only consider a binary source.)
(d) Show that a Shannon-Fano code for a finite probability distribution has average length equal to the entropy if and only if all of the probabilities are powers of two.
(e) Show that $\overline{L}^* = H$ for a finite probability distribution if and only if all probabilities in the set are powers of two.
(f) Find a finite probability distribution $\{p_1, \ldots, p_Q\}$ for which there does not exist a prefix code with lengths $L_q = \lfloor -\log_2 p_q \rfloor$, $q = 1, \ldots, Q$. This explains why in developing the Shannon-Fano codes, we conservatively rounded up rather than down.
(g) Show that if a finite probability distribution $\{p_1, \ldots, p_Q\}$ contains at least one probability that is not a power of two, then there does not exist a prefix code with lengths $L_q = \lfloor -\log_2 p_q \rfloor$, $q = 1, \ldots, Q$. (This is a strengthening of the (f).) □

Exercise 2.14 *For each of the following lists, find whether or not there exists a prefix code whose codeword lengths are those in the list. (a) $\{1, 2, 3, 4\}$, (b) $\{1, 3, 4, 5, 5, 5, 5, 5, 5\}$, (c) $\{2, 2, 3, 3, 4, 4, 4, 4, 5, 5\}$.* □

Codes with larger source lengths

Let us now turn our attention to prefix codes with source lengths K greater than or equal to 1. Since the rate of such a code is proportional to its average length, it has minimal rate if

and only if it has minimal average length. So we need only apply what we have just learned, except that here we need a codeword for each source sequence of length K (i.e., Q^K codewords, one for each $u^K \in A_U^K$), and the relevant probabilities are $\{p(u^K) : u^K \in A_U^K\}$. We conclude that the codeword for u^K should have length approximately equal to $-\log_2 p(u^K)$. Specifically, there exists a prefix code with lengths $L(u^K) = \lceil -\log_2 p(u^K) \rceil$; this code has

$$H^K \le \overline{L} < H^K + 1 \; ; \tag{2-82}$$

and every prefix code with source length K has

$$\overline{L} \ge H^K \; , \tag{2-83}$$

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

$$H^K \stackrel{\Delta}{=} - \sum_{u^K \in A_U^K} p(u^K) \log_2 p(u^K) \; . \tag{2-84}$$

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

$$\begin{aligned} H^K &= -\sum_{u^K \in A_U^K} p(u^K) \log_2 \prod_{k=1}^K p(u_k) &&= -\sum_{u^K \in A_U^K} p(u^K) \sum_{k=1}^K \log_2 p(u_k) \\ &= -\sum_{k=1}^K \sum_{u^K \in A_U^K} p(u^K) \log_2 p(u_k) &&= -\sum_{k=1}^K \sum_{u_k \in A_U} p(u_k) \log_2 p(u_k) \\ &= KH \; . \end{aligned} \tag{2-85}$$

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

$$KH \le L_K^* < KH + 1 \; . \tag{2-86}$$

As a consequence, the least rate, $R_{VL,K}^* = L_K^*/K$, is between H and $H + 1/K$. In effect, larger source lengths enable us to reduce the 1 in equation (2-79) to $1/K$, which is especially important when H is small. In addition, we easily see that $R_{VL}^* \stackrel{\Delta}{=} \inf \{R_{VL,K}^* : K = 1, 2, \ldots\} = H$. We summarize in the following.

Theorem 2.6 (Coding Theorem for FVB Prefix Codes) *Let U be an IID source with finite alphabet and entropy H.*

(a) Positive statements:

$$R_{VL}^*(K) < H + \frac{1}{K} \; , \text{ for every positive integer } K \; ; \tag{2-87}$$

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

$$R_{VL}^* \le H \; ; \tag{2-88}$$

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

(b) Converse Statement:

$$R_{VL}^*(K) \ge R_{VL}^* \ge H \; , \text{ for every positive integer } K \; ; \tag{2-89}$$

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

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

$$H \le R_{VL}^*(K) < H + \frac{1}{K} \; , \tag{2-90}$$

and

$$R_{VL}^* = H \; . \tag{2-91}$$

Exercise 2.15 *(a) Show that $R^*_{VL}(K) \geq R^*_{VL}(MK)$ for any positive integers M and K. (Hint: Consider a code with source length MK whose codebook consists of all possible concatenations of M codewords from the codebook of an optimal code with source length K.) (b) Find an example of a source for which $R^*_{VL}(K+1) < R^*_{VL}(K)$ for some K. (c) (Difficult) Find another example for which $R^*_{VL}(K+1) > R^*_{VL}(K)$.* □

Huffman's code design algorithm

Our final task is to answer Question 2.3, namely: How does one design prefix codes with the least possible average length and rate? One of the points of Exercise 2.13 was to show that the Shannon-Fano code does not always give the least average length. Optimal codes, i.e those with smallest rate, are found by Huffman's algorithm, which we will now describe. The resulting codes are often called *Huffman codes*.

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

The key observation, to be proved later, is:

Lemma 2.7 *If $C_{Q-1} = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-1}\}$ is an optimum codebook for P_{Q-1}, then $C_Q = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-2}, \underline{v}'_{Q-1}0, \underline{v}'_{Q-1}1\}$ is an optimum codebook for P_Q.*

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

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

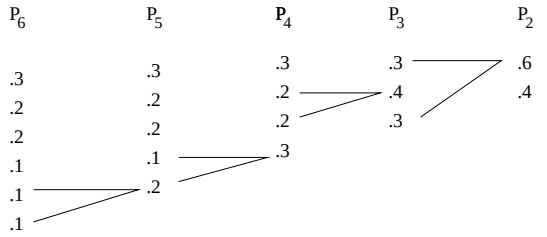
We now work our way backwards. An optimum codebook for the set P_2 is, obviously, $C_2 = \{0, 1\}$. An optimum codebook C_3 for P_3 (with three probabilities) is obtained by appending both 0 and 1 to the codeword in C_2 associated with the probability in P_2 that is the sum of the two smallest probabilities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

The process of reducing a probability distribution and then expanding the codebooks is illustrated in Figure 2.14. Notice that at various stages there are three or more smallest probabilities, from which we arbitrarily choose to combine two. Consequently, the Huffman algorithm may be used to generate a number of optimum codebooks, even having different collections of lengths (see Exercise 2.16). Of course they all have the same average length, for otherwise they would not all be optimum. It remains only to prove the key observation.

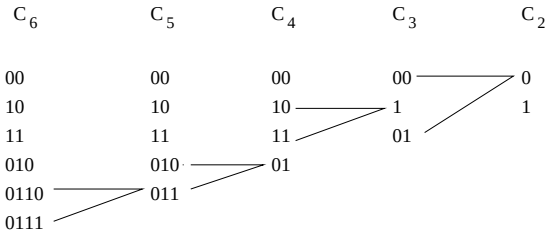
Proof of Lemma 2.7

Let $C_{Q-1} = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-1}\}$ be an optimum codebook for P_{Q-1}, and let $C_Q = \{\underline{v}'_1, \ldots, \underline{v}'_{Q-2}, \underline{v}'_{Q-1}0, \underline{v}'_{Q-1}1\}$ be the codebook for P_Q created by extending C_{Q-1}. The average length of C_Q is related to that of C_{Q-1} via

$$\overline{L}_Q = \sum_{q=1}^{Q} p_q L_q = \sum_{q=1}^{Q-2} p_q L'_q + p_{Q-1}(L'_{Q-1} + 1) + p_Q(L'_{Q-1} + 1)$$



(a) Reducing the sets of probabilities.



(b) Expanding the set of codewords.

Figure 2.14: Huffman design procedure.

$$
\begin{aligned}
&= \sum_{q=1}^{Q-1} p'_q L'_q + (p_{Q-1} + p_Q) \\
&= \overline{L}_{Q-1} + (p_{Q-1} + p_Q) \ . \qquad \text{(2-92)}
\end{aligned}
$$

We will now use proof by contradiction. Suppose C_Q were not optimum for P_Q. Then an optimum code $C_Q^* = \{\underline{v}_1^*, \ldots, \underline{v}_Q^*\}$ for P_Q will have average length $\overline{L}_Q^* < \overline{L}_Q$. Moreover, Exercise 2.22, below, shows that C_Q^* can be chosen so that the codewords associated with p_Q and p_{Q-1} are siblings in the sense of having the same length and differing only in the last bit. From C_Q^* we may in turn create a code $C_{Q-1}^* = \{\underline{v}_1^*, \ldots, \underline{v}_{Q-2}^*, \underline{v}'_{Q-1}\}$ for P_{Q-1}, where $\underline{v}'_{Q-1}$ is obtained by stripping the last bit from $\underline{v}_{Q-1}^*$ (or for that matter, from its sibling $\underline{v}_Q^*$). Notice that C_Q^* is, in fact, the direct extension of C_{Q-1}^*. Therefore using (2-92), the average length of C_{Q-1}^* is

$$
\begin{aligned}
\overline{L}_{Q-1}^* &= \overline{L}_Q^* - p_{Q-1} - p_Q \ < \ \overline{L}_Q - p_{Q-1} - p_Q \\
&= \overline{L}_{Q-1} \ , \qquad \text{(2-93)}
\end{aligned}
$$

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

Example 2.5 *An optimal code (source length $K = 1$) for the probabilities of English letters given in Figure 2.9 is shown in Figure 2.15. Its rate is 4.12 bits per symbol which compares to the entropy of 4.08.* □

Exercise 2.16 *An IID source U has alphabet $A_U = \{a, b, c, d, e\}$ and probabilities $\{.4, .2, .2, .1, .1\}$.*
(a) Find two prefix codes with source length 1 whose average lengths are minimum and that whose collections of lengths are different.
(b) For each code compute the average and variance of its lengths.

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

Figure 2.15: Huffman code for English

(c) Can you think of a reason why a code with smaller variance would be useful? (Hint: See the discussion below on buffering.) (d) Find the smallest source length K for which there exists a prefix code with rate $R \leq H + .1$. □

Exercise 2.17 *A binary IID source U has $p_0 = .9$ and $p_1 = .1$.*
(a) Find the smallest possible rate of any FVB lossless source code?
(b) Find a fixed-to-variable length block prefix code with rate .55 or less. Make it as simple and good as possible. Compute the rate of your code. □

Exercise 2.18 *A binary IID source has $p(0) = .995$ and $p(1) = .005$.*
(a) Find an FVB noiseless source code with rate no larger than .4 . (It should be as simple as possible.)
(b) Compare the performance and complexity of this code with the FFB code of Problem 2.12.
(c) Is it possible to find an FVB noiseless source code with rate less .1? If so, without applying Huffman's algorithm, what can be said about how large its source length would have to be? That is, find upper and lower bounds to the minimum possible blocklength. The tighter the bounds, the better. □

Exercise 2.19 *An IID source U has $L_2^* = 4$ and $L_3^* = 4.8$. What can be said about its entropy H? In other words, find upper lower and bounds to H. The tighter the bounds, the better.* □

Exercise 2.20 *Show that $R_{VL}^*(K) \leq R_{PL}^*(K)$ for every K. Does equality always hold? Does it hold sometimes?* □

Exercise 2.21 *An IID source U has $Q = 3$ equiprobable symbols.*
(a) For $K = 1, 2, 3$, find $R_{VL}^*(K)$ and compare to the upper and lower bounds in (2-90).
(b) For $K = 1, 2, 3$, find $R_{PL}^*(K)$ and compare to the upper and lower bounds in (2-17).
(c) For $K = 1, 2, 3$, compare $R_{VL}^*(K)$ and $R_{PL}^*(K)$ □

Exercise 2.22 *Show there exists an optimum codebook C_Q for the probability distribution $P_Q = \{p_1, \ldots, p_Q\}$, $p_1 \geq p_2 \geq \cdots \geq p_Q$, such that the codewords assigned to p_{Q-1} and p_Q are siblings in the sense of having the same length and differing only in the last bit. Hint: First show that the longest codeword in any optimum codebook has another codeword as a sibling.* □

Exercise 2.23 *(From McEliece, Problem 10.22) Consider the game of "twenty questions" in which you are required to determine the outcome of one roll of a pair of dice by asking*

questions that can be answered "yes" or "no". The outcome to be guessed is one of the integers $2, 3, 4, \ldots, 12$. *A question takes the form "Is* $D \in S$*?" where* D *is the outcome of the dice and* S *is a subset of the integers* $\{2, 3, \ldots, 12\}$. *The choice of a question, i.e the choice of* S*, may depend on the answers to the previous questions, and the number of questions until the outcome is determined need not be the same for all outcomes. Find a questioning strategy that, on the average, requires the fewest number of questions.*

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

Remark

Benefits of larger source lengths

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

Notice that although Theorem 2.6 finds R^*_{VL} exactly, it gives only bounds to $R^*_{VL}(K)$. To find the latter exactly, one must apply the Huffman algorithm to find an optimum code with source length K for the probability distribution $p_{U^K}(u^k)$. By definition, the rate of this code is $R^*_{VL}(K)$.

Complements

Synchronization and transmission errors

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

Let us first consider the situation in which a prefix code is used to encode an infinite sequence of source symbols, but for some reason, the first few bits of the binary representation become lost. Clearly, the decoder is not likely to be able to determine any of the source symbols whose codewords have missing bits. But it may also happen that subsequent source symbols are incorrectly decoded; that is, the errors caused by this loss may propagate. For example, suppose the codebook $\{01, 001, 101, 110\}$ is used for the alphabet $\{a, b, c, d\}$, suppose the codeword 110 for d is transmitted repeatedly; and suppose the first bit is lost, so the decoder is given only 10110110110110110 .... Instead of finding the codeword 110 repeated infinitely many times and decoding into $ddd\ldots$, the decoder finds 101 repeated infinitely and decodes into $ccc\ldots$ Basically, the loss of the initial bit caused the encoder to lose track of where the codewords began. We call this a loss of synchronization. Its effect on this code is disastrous. In contrast, a loss of synchronization has very little effect on the codebook $\{1, 01, 001, 0001\}$ because the end of each codeword is so easily recognized.

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

Buffering

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

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

Underflow is the reverse problem. Suppose over some period of time the source produces a sequence of very likely symbols, so that the encoder produces bits at a rate below that which the channel needs. At some point, the channel will find the buffer empty. Although there is nothing to transmit, the channel will nevertheless produce a bit (probably at random) at its output, which the decoder will interpret as a real bit; i.e., it gets inserted into Z. This will cause at least one source symbol error and possibly more, if synchronization is lost. To prevent this sort of thing, whenever the buffer empties, one should immediately put in a specially designated codeword, called a *flag*, that indicates to the decoder that there was really nothing to send. The code, augmented by the flag, must be a prefix code, and this means that one or more of the codewords will be longer than they would otherwise need to be. Thus the rate of the code will be slightly larger. The flag will also add delay to the system, for once it is entered into the buffer, it must be transmitted in entirety, even if the encoder has already placed something in the buffer.

Separable codes

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

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

Exercise 2.24 *Verify that prefix codes are always separable.* □

Exercise 2.25 *Show that any codebook with the suffix-free property that no codeword is the suffix of another is separable.* □

A result known as McMillan's Theorem shows that, just as with prefix codes, the lengths of any separable code satisfy the Kraft inequality. Accordingly, there must also be a prefix code with exactly the same lengths and rate. This is why one can restrict attention to prefix codes with no loss in potential performance.

Separable codes are also called *uniquely decodable*. However, Exercise 2.26, below, suggests this is not such a good name, for although it is always possible to uniquely decode finite sequences of codewords from a separable code it is not always possible to uniquely decode infinite sequences. All the more reason to prefer prefix codes.

Exercise 2.26 *(a) Show that the codebook* $\{00, 001, 1010, 0101\}$ *is separable. (Hint: see Exercise 2.25.) (b) Show that the infinite binary sequence* $001010101010101\ldots$ *can be decoded in two very different ways.* □

Infinite alphabet sources

It can be shown that Kraft inequality also holds for countably infinite collections of code lengths. Thus, although we restricted attention in this section to finite alphabet sources, in fact, the coding theorem for FVB prefix codes (Theorem 2.6) holds as stated for sources with countably infinite alphabets. Shannon-Fano coding works fine, as well. On the other hand, Huffman's code design algorithm depended greatly on the finite alphabet assumption and cannot be applied when the alphabet is countably infinite.