Second Edition

Coding and Information Theory

RICHARD W. HAMMING

Naval Postgraduate School

PRENTICE-HALL / Englewood Cliffs NJ 07632

If you estimate the time to locate an error in miswriting a name against the time to add one or more characters to all the names while writing the original program, you will see that using longer-than-minimum names is probably a very wise idea—but it is unlikely that this computation will convince many people to do so! Minimal-length names are the source of much needless confusion and waste of time, yours and the machine's.

The distance function in the Hamming codes is based completely on white noise. Sections 2.7 and 2.9 are included to emphasize that the proper distance function to use depends in some cases on the *psychological distance* between the names as well as more uniformly random keystroke errors.

3.10 Summary

We have given the fundamental nature of error detection and error correction for white noise, namely the minimum distance between message points that must be observed. We have given methods for constructing codes for:

Single-error detecting	min. dist. = 2
Single-error correction	min. dist. = 3
Single-error correction + double-error detection	min. dist. = 4

Their design is easy, and they are practical to construct in software or in hardware chips. They can compensate for weak spots in a system, or can be used throughout an entire system, to get reliable performance out of unreliable parts. One need not put up with poor equipment performance, but the price is both in storage (or time of transmission) and equipment (or time) to encode and possible correct. You do not get something for nothing! The codes also make valuable contributions to maintenance since they pinpoint the error, and the repairmen will not try fixing the wrong things (meaning "fix" what is working right and ignore what is causing the error!). A more widespread use of the idea of distance between messages was sketched in Section 3.9.

The use of such codes, and more highly developed codes, is rapidly spreading as we get faster and smaller integrated-circuit components. Increasingly, in VLSI (very large system integration) chips the code is part of the hardware.

Chapter 4

Variable-Length Codes: Huffman Codes

4.1 Introduction

We now turn to the topic of source encoding. The codes we have looked at so far have all used a fixed length, and they are called *block codes* from the fact that the messages are of fixed block lengths in the stream of symbols being sent. The Morse code mentioned in Chapter 1 is an example of a *variable-length* code, and we now examine variable-length codes in more detail. The advantage of a code in which the message symbols are of variable length is that sometimes the code is more *efficient* in the sense that to represent the same information we can use fewer digits on the average. To accomplish this, we need to know something about the statistics of the messages being sent. If every symbol is as likely as every other one, then the block codes are about as efficient as any code can be (see Section 4.6). But if some symbols are more probable than others, then we can take advantage of this to make the most frequent symbols correspond to the shorter encodings, and the less frequent symbols correspond to the longer encodings. This is exactly what the Morse code does. The letter E in the English language occurs most frequently, and corresponds to the encoded symbol "dot."

However, variable-length codes bring with them a fundamental problem: At the receiving end, how do you recognize each symbol of the code? In a binary system, for example, how do you recognize the end of one code word and the beginning of the next? If the probabilities of the frequencies of occurrence of the individual symbols are sufficiently different, then variable-length encoding can be significantly more efficient than block encoding.

In this chapter we take advantage only of the frequencies of occurrence of the individual symbols being sent, and neglect any larger structure the messages may have. An example of a slightly larger scale structure is the fact that in English the letter Q is usually followed by the letter U; in Chapter 5 we take advantage of such correlations in the message being sent.

Exercise

4.1-1 List advantages and disadvantages of variable-length codes.

4.2 Unique Decoding

We need to make clear when we are talking about the symbols to be sent and when we are talking about the symbols used by the signaling system. We will refer to the *source symbols* when we refer to the symbols (such as the letters of the English alphabet) that are to be sent, and to the *code's alphabet* when we refer to the symbols used in sending (such as 0 and 1 in the binary system). In general, we will assume that the source alphabet being sent has q symbols, $s_1, s_2 \ldots, s_q$, and that the code's alphabet has r symbols (r for the radix of the system).

The first property that we need is *unique decodability*—the received message must have a single, unique possible interpretation. Consider a code in which the source alphabet S has four symbols, and they are to be encoded in binary as follows:

$$s_1 = 0$$

$$s_2 = 01$$

$$s_3 = 11$$

$$s_4 = 00$$

The particular received message 0011 could be one of these two:

$$0011 = \begin{cases} s_4, s_3 \\ s_1, s_1, s_3 \end{cases}$$

Thus the code is not uniquely decodable. Although this property of unique decodability is not always absolutely essential, it is usually highly desirable.

To get our thoughts clear, we make a formal definition:

Definition. The nth extension of a code is simply all possible concatenations of n symbols of the original source code.

This is also called the nth *direct product* of the code. There are q^n symbols in the nth extension. The definition is necessary because the messages I send look to you, the receiver, as concatenations of the encoded symbols of the source alphabet, and you need to decide which sequence of source symbols I sent. For unique decodability, no two encoded concatenations can be the same, even for different extensions. Clearly, only if every distinct sequence of source symbols has a corresponding encoded sequence that is unique can we have a uniquely decodable signaling system. This is a necessary and sufficient condition, but it is hardly usable in this form.

Exercises

4.2-1 Is the code 0, 01, 001, 0010, 0011 uniquely decodable?

4.2-2 Is the code 0, 01, 011, 111 uniquely decodable?

4.3 Instantaneous Codes

Consider the following code:

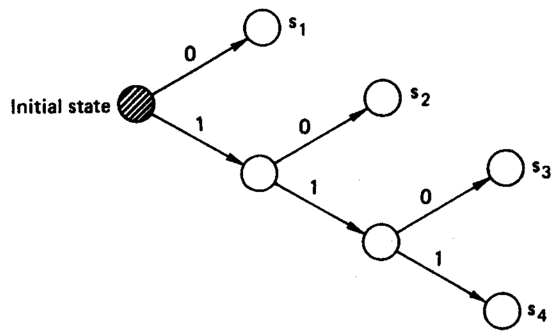
$$s_1 = 0$$

$$s_2 = 10$$

$$s_3 = 110$$

$$s_4 = 111$$

Now consider how you as the receiver would decode messages sent in this code. You would set up what is equivalent to a *finite automaton*, or if you prefer, a *decision tree*. Starting in the initial state (Figure 4.3-1), the first binary digit received will cause a branch, either to a terminal state s_1 if the digit is 0, or else to a second decision point if it is a 1. For the next binary digit this second branch would go to the terminal state s_2 if a 0 is received, and to a third decision point if it is a 1. The third would go to the terminal state s_3 if the third digit is a 0, and to the terminal symbol s_4 if it is a 1. Each terminal state would,



$$\begin{cases} s_1 = 0 \\ s_2 = 10 \\ s_3 = 110 \\ s_4 = 111 \end{cases}$$

Figure 4-3.1 Decoding tree

of course, emit its symbol and then return control to the initial state. Note that each bit of the received stream is *examined only once*, and that the terminal states of this tree are the four source symbols s_1, s_2, s_3, and s_4.

In this example the decoding is *instantaneous* since when a complete symbol is received, you, the receiver, immediately know this, and do not have to look further before deciding what message symbol you received. No encoded symbol of this code is a prefix of any other symbol. This particular type of code is what is called a *comma code* since the binary digit 0 indicates the end of a symbol, *plus* the fact that in this case no symbol is longer than three digits.

From this you see the basic equivalence of the existence of the decoding tree and the instantaneous decodability; each implies the other. Notice again that using the decoding tree means that each received symbol is looked at only once in the decoding process.

The following code is uniquely decodable but is not instantaneous because you do not know when one symbol is over without looking further:

$$s_1 = 0$$
$$s_2 = 01$$
$$s_3 = 011$$
$$s_4 = 111$$

(It is the previous code with the bits reversed.) We can see the trouble—some code words are *prefixes* of other words; that is, they are the same

as the beginning part of some other symbol. Now consider the string

$$0111\ldots\underbrace{111}_{s_4}\underbrace{1}_{s_4}$$

It *can only* be decoded by first going to the end and then identifying runs of three 1's as each being an s_4 until the first symbol is reached. Only then can it be identified as s_1, s_2, or s_3. Thus you cannot decide whether you have the word that corresponds to the prefix, or if you must wait until more is sent to complete the word. The simplest way to decode messages in this particular code is always to start at the back end of the received message! This puts a severe burden on the storage and also causes a time delay.

In view of the existence of the decoding tree, *it is clearly both necessary and sufficient that an instantaneous code have no code word s_i which is a prefix of another code word, s_j.* If we had to deal with a noninstantaneous code, then the attempted decoding tree would have a structure which instead of returning to the start would, when it realized (finally) that it had received a complete word some time ago, emit the proper code word and then go to an appropriate place in the tree, not necessarily the start. As in the example above, the tree could require potentially infinite storage.

4.4 Construction of Instantaneous Codes

It is clear that of all uniquely decodable codes, the instantaneous codes are preferable to ones that are not, and since it will turn out (Section 4.7) that they cost us nothing extra, it is worth concentrating on them. Let us, therefore, explore the problem of constructing them.

Given that we are to construct a code with five symbols s_i in the source code S, and that the code alphabet is binary, we can assign

$$s_1 = 0$$
$$s_2 = 10$$
$$s_3 = 110$$
$$s_4 = 1110$$
$$s_5 = 1111$$

to get an instantaneous (comma) code (Figure 4.4-1).

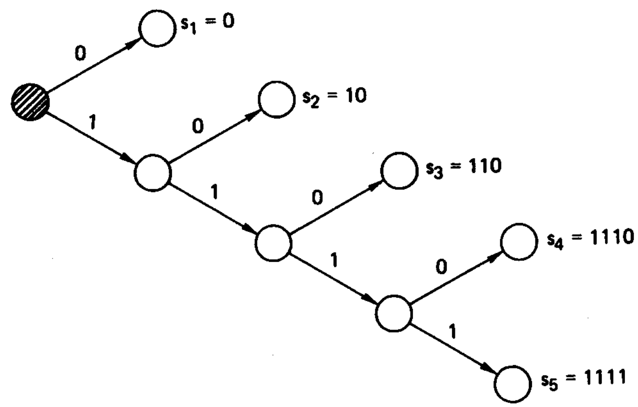


Figure 4-4.1 Decoding tree

In this construction the use of the 0 for the first symbol reduced the number of possibilities available later. Instead of this, let us use two digits for the first two symbols, $s_1 = 00$ and $s_2 = 01$. We can now use $s_3 = 10$. With two symbols yet to go, we cannot use $s_4 = 11$; instead, we must try $s_4 = 110$, which leaves us with $s_5 = 111$. We get the code

$$s_1 = 00$$
$$s_2 = 01$$
$$s_3 = 10$$
$$s_4 = 110$$
$$s_5 = 111$$

This code is clearly instantaneous since no symbol is a prefix of any other symbol, and the decoding tree is easily constructed (Figure 4.4-2).

Which of these two codes is better (more efficient)? This depends on the frequency of occurrence of the symbols s_i. By "better" we mean, of course, that the sent messages *on the average* will be shorter, more efficient. We will investigate this more closely in Section 4.8. Evidently, "efficient" *must depend on the probability of the various symbols being used in the messages being sent.*

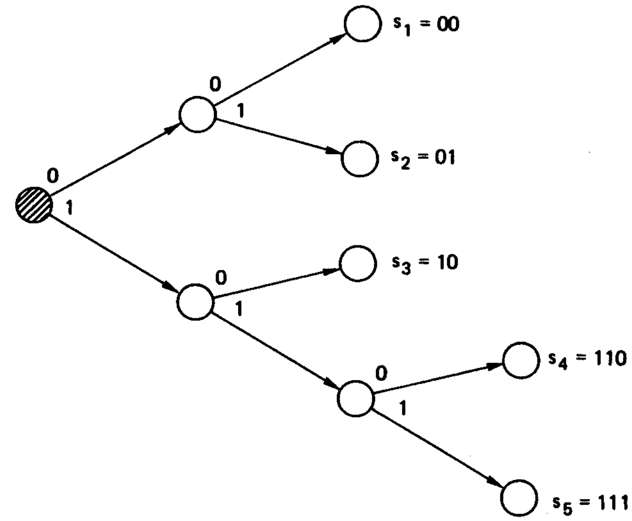


Figure 4-4.2 Decoding tree

Exercises

4.4-1 Devise a similar example using six words.

4.4-2 Find probabilities that favor one or the other of the two codes in this section. Ans. $p_3 + p_4 + p_5$ vs. p_1

4.5 The Kraft Inequality

The Kraft inequality, which we now examine, gives conditions on the existence of instantaneous codes; it tells when the *lengths* of the code words permit forming an instantaneous code, but it does not discuss the code itself.

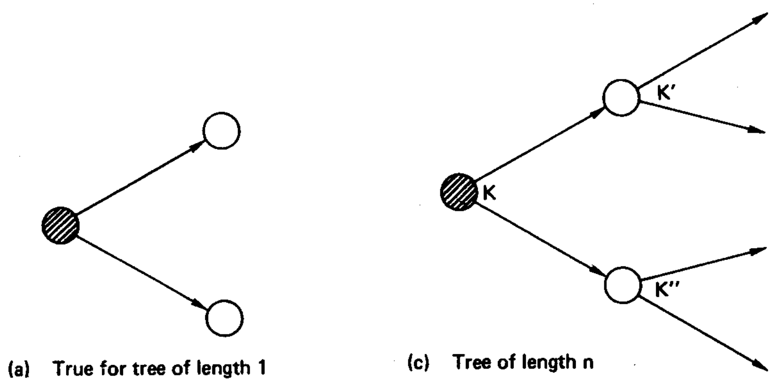
Theorem. A necessary and sufficient condition for the existence of an instantaneous code S of q symbols s_i ($i = 1, \ldots, q$) with encoded words of lengths $l_1 \le l_2 \le l_3 \le \ldots \le l_q$ is

$$\sum_{i=1}^{q} \frac{1}{r^{l_i}} \le 1 \qquad (4.5\text{-}1)$$

where r is the radix (number of symbols) of the alphabet of the encoded symbols.

This inequality is asserting that we cannot have too many short encoded symbols. Most of the l_i must be reasonably large.

It is easy to prove the Kraft inequality from the decoding tree, whose existence follows from the instantaneous decodability. We proceed by induction. For simplicity consider first the binary case (see Figure 4.5-1). For a tree whose maximum length is 1, we have either



(a) True for tree of length 1

(b) Assume true for length n - 1

(c) Tree of length n

$\frac{1}{2}K' + \frac{1}{2}K'' = K$

Figure 4-5.1 Proof of Kraft inequality

one or two branches of length 1. Thus we have either (for one symbol)

$$\tfrac{1}{2} \le 1$$

or (for two symbols)

$$\tfrac{1}{2} + \tfrac{1}{2} \le 1$$

We next assume that the Kraft inequality is true for all trees of length less than n. Now given a tree of maximum length n, the first node leads to a pair of subtrees of length at most $n - 1$, for which we have the inequalities $K' \le 1$ and $K'' \le 1$, where K' and K'' are the values of their respective sums. Each length l_i in a subtree is increased by 1 when the subtree is joined to the main tree, so an extra factor of $\frac{1}{2}$ appears. We have, therefore,

$$\tfrac{1}{2}K' + \tfrac{1}{2}K'' \le 1$$

For radix r instead of binary, we have at most r branches at each node—at most r subtrees each with an extra factor of $1/r$ when joined to the main tree. Again the theorem is true.

When can an inequality occur? A moment's inspection shows that if every terminal node of the tree is a code word, then $K = 1$. It is only when some terminal nodes are not used that the inequality occurs. But if any terminal node is not used *for a binary code alphabet*, the preceding decision is wasted and that corresponding digit can be removed from every symbol that passes through this node in its decoding. Thus if the inequality holds, the code is inefficient, and how to correct for this is immediately evident *for binary trees*. Thus $K = 1$ for binary trees with all the terminals used. It is only for radix $r > 2$ that it is reasonable to have unused terminals and hence a K less than 1. Since the theorem gives a condition on the lengths only, the main use is in questions of the existence of a code with a given set of lengths.

Note again that the theorem refers to the existence of such a code, and does not refer to a particular code. A particular code may obey the Kraft inequality and still not be instantaneous, but there will exist codes that have the l_i and are instantaneous.

In the next two examples we will assume that only the code word lengths are given since this is what matters in the theorem, not the actual code words. If the lengths when encoded in binary symbols are 1, 3, 3, 3, the Kraft sum will be $\frac{1}{2} + \frac{3}{8} = \frac{7}{8}$, and an instantaneous code with those lengths is possible. One of the words of length 3 could be shortened to 2 bits. But if the lengths were 1, 2, 2, 3, the sum would be $\frac{1}{2} + 2(\frac{1}{4}) + \frac{1}{8} = \frac{9}{8}$, and such an instantaneous code could not exist.

Let us apply the theorem to yet another example. Suppose that we pick the radix $r = 3$ and want words of lengths 1, 2, 2, 2, 2, 2, 3, 3, 3, 3. The Kraft inequality gives on the left side $\frac{1}{3} + 5(\frac{1}{9}) + 4(\frac{1}{27}) = \frac{28}{27}$, so we cannot hope to find an *instantaneous code* with these constraints. If we drop the last code word of length 3, the sum would be exactly 1 and we can find such a code. To find an instantaneous code with these lengths, we proceed systematically: $s_1 = 0, s_2 = 10, s_3 = 11, s_4 = 12, s_5 = 20, s_6 = 21, s_7 = 220, s_8 = 221, s_9 = 222$. In this construction we have followed the decoding tree (Figure 4.5-2), and have systematically increased the ternary (base 3) number equivalents of the codes so that the argument is easily followed. The reader should realize that the three symbols 0, 1, and 2 are arbitrary and are not numbers. Thus any interchanges of the three symbols at any stage would leave the code essentially the same; it merely interchanges the order of the branches at a node. Indeed, one could interchange the order at one node without doing so at another and still have an equivalent code.

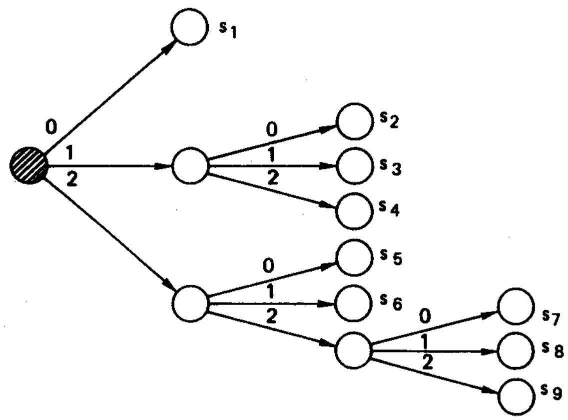


Figure 4-5.2 Decoding tree

Exercises

4.5-1 Does the infinite-length comma code $l_1 = 1, l_2 = 2, \ldots, l_k = k, \ldots$ satisfy the Kraft inequality? (Use $r = 2$.)

4.5-2 Generalize Exercise 4.5-1 to radix r.

4.5-3 Carry out the computation of K for a comma code.

4.6 Shortened Block Codes

Returning for the moment to the earlier block codes, if we had exactly 2^m code words in a binary system (r^m in a radix r system), we could use m digits to represent each symbol. But suppose that we do not have an exact power of the radix but still want the maximum length to be as short as possible. To see what can happen, consider the case of five symbols. Of the eight binary symbols

000
001
010
011
100
101
110
111

we can drop any three. But if we drop 001, 011, and 101, we can shorten three branches of the decoding tree and still have instantaneous decodability. We will have

$$s_1 = 00$$
$$s_2 = 01$$
$$s_3 = 10$$
$$s_4 = 110$$
$$s_5 = 111$$

(See Figure 4.6-1.) Instead of this choice, we can drop 001, 010, and 011 and shorten only one branch of the tree to the code.

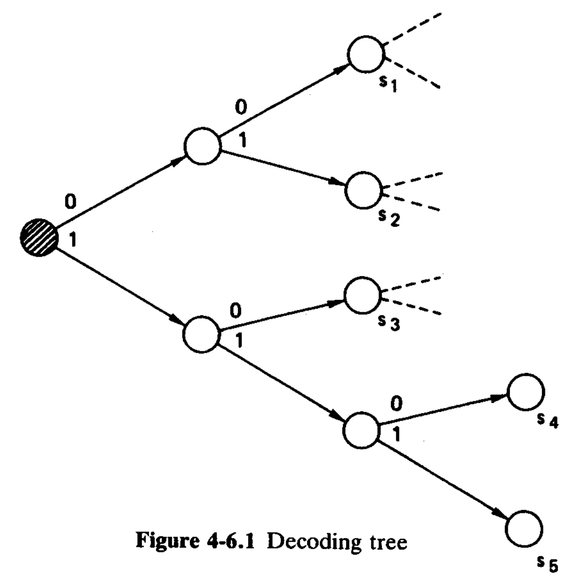


Figure 4-6.1 Decoding tree

$$s_1 = 0$$
$$s_2 = 100$$
$$s_3 = 101$$
$$s_4 = 110$$
$$s_5 = 111$$

(See Figure 4.6-2.) In both cases there now are no unused terminals and therefore $K = 1$. We will call these *shortened block codes;* they are essentially block codes with small modifications.

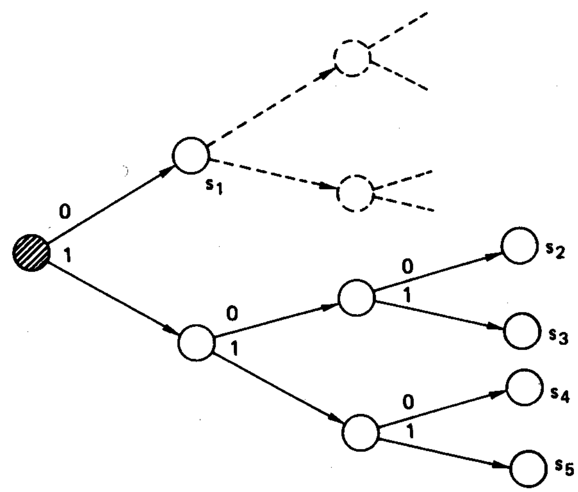


Figure 4.6-2

Exercises

4.6-1 Discuss the case of two symbols in a ternary base.

4.6-2 Discuss the case of five symbols in the ternary base.

4.7 The McMillan Inequality

The Kraft inequality applies to instantaneous codes, which are a special case of *uniquely decodable codes.* McMillan showed that the same inequality applies to uniquely decodable codes. The underlying idea of the proof of the necessity is that very high powers of a number greater than 1 grow rapidly. If we can tightly bound this growth, then we know that the number is not greater than 1. The proof of the sufficiency follows from the fact that we can do it for instantaneous codes which are special cases of uniquely decodable codes.

The necessity part of the proof begins by taking the nth power of the Kraft expression

$$\left[\sum_{i=1}^{q} \frac{1}{r^{l_i}}\right]^n \equiv K^n$$

When we expand the left-hand term we find that we have a sum of many terms to various powers: the exponents running from n, the lowest possible power, to nl, the highest, where l is the length of the longest symbol. Thus we have the expression

$$K^n = \sum_{k=n}^{nl} \frac{N_k}{r^k}$$

where N_k is the number of code symbols (of radix r) of length k. Since the code is uniquely decodable, N_k cannot be greater than r^k, which is the number of distinct sequences of length k in our code alphabet of radix r. Therefore, we have the bound

$$K^n \leq \sum_{k=n}^{nl} \frac{r^k}{r^k} = nl - n + 1 < nl$$

(The $+1$ comes from the fact that both end terms in the sum are counted.) This is the inequality we need, since for any $x > 1$ a sufficiently large n makes the number $x^n > nl$. But n can be chosen as large as we please, and it follows therefore that the number K (the Kraft sum) must be ≤ 1.

From this we see that there is very little to gain from avoiding instantaneously decodable codes and using the more general uniquely decodable codes; both have to satisfy the same Kraft inequality on the lengths of the encoded symbols.

4.8 Huffman Codes

Now for the first time we will make use of the *probabilities* of the various symbols being sent. As in the Morse code, we want the most frequent symbols to have the shortest encodings. If the probability of the ith symbol is p_i and its length is l_i, then the average length of the code is

$$L_{av} = \sum_{i=1}^{q} p_i l_i$$

With no loss in generality the p_i may be taken in decreasing order. If the lengths l_i are not in the opposite order, that is, we do not have both

$$p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_q$$

and

$$l_1 \le l_2 \le l_3 \le \ldots \le l_q$$

then the code is not *efficient* in the sense that we could have a shorter average length by reassigning the code representations of the symbols $s_1, s_2, s_3, \ldots, s_q$. To prove this assertion, suppose that for $m < n$ we have both conditions (for some m and n)

$$p_m > p_n \quad \text{and} \quad l_m > l_n$$

In computing the average length, we have, among others, the two terms

$$\text{Old:} \quad p_m l_m + p_n l_n$$

By interchanging the encoded symbols for s_m and s_n we get the corresponding terms

$$\text{New:} \quad p_m l_n + p_n l_m$$

Subtracting the *Old* from the *New* we have the change due to this reassignment:

$$\text{New} - \text{Old:} \quad p_m(l_n - l_m) + p_n(l_m - l_n) = (p_m - p_n)(l_n - l_m) < 0$$

From the foregoing assumptions this is a negative number; we will decrease the average code length if we interchange the encoded symbols for s_m and s_n. We therefore assume that the two running inequalities both hold.

We begin our examination of Huffman coding with encoding into the binary alphabet. In Section 4.11 we look at the base r alphabet. We will use "source symbol" for the input s_i and "code alphabet" for the alphabet into which we are encoding.

The first thing to prove is that the two least frequent source symbols of an efficient code have the same encoded lengths. Suppose that the maximum coded symbol has length l. If there is only one of such length, then since the code is instantaneous, any shorter coded symbol of length $l - 1$ (or shorter) is not a prefix of the maximum-length coded symbol. Therefore, the last part of the longest symbol could be dropped with no loss of information in decoding. Thus the two longest symbols must have the same length, and because of the running inequalities, they must be the two least probable.

The proof of the encoding properties, as well as the method of encoding, is a reduction at each stage to a shorter code. We simply

combine the two least probable symbols of the source alphabet into a single symbol whose probability is equal to the sum of the two corresponding probabilities. Thus we have to encode a source alphabet of one less symbol. Repeating this step by step, we get down to the problem of encoding just two symbols of a source alphabet, which is easy—merely use 0 and 1. Now in going backward one of these two symbols is to be split into two symbols, and this can be done by appending a second digit 0 for one of them and 1 for the other. In the next stage of going back, one of these three symbols is to be split into two symbols in the same way. And so it goes. For one special case given below, Figure 4.8-1 shows the reduction process, and Figure

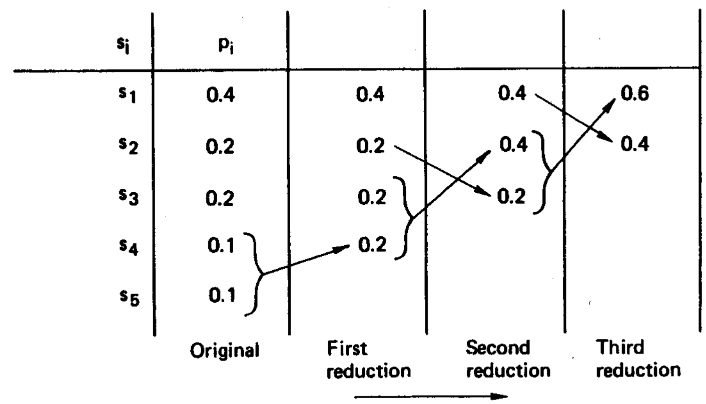


Figure 4-8.1 Reduction process

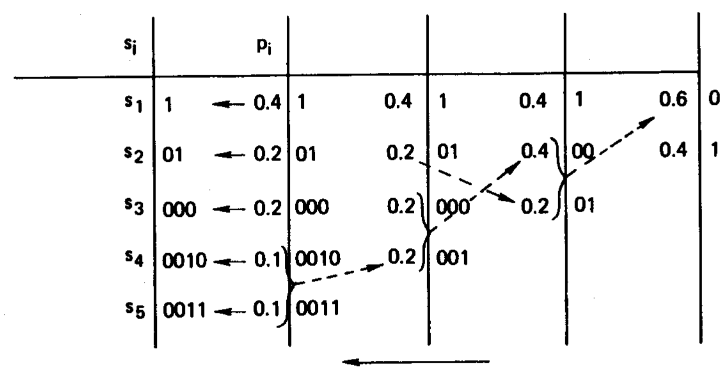


Figure 4-8.2 Splitting process

4.8-2 shows the corresponding splitting (expansion) process. The general case should be obvious from this.

How do we know that this process generates an efficient code? Suppose that there were a shorter code with code length L' with

$$L' < L$$

Let us compare the two decoding trees. In an efficient binary code, all the terminals are occupied and there are no "dead branches." (A dead branch would enable us to shorten the code by deleting the corresponding binary digit in all the terminals that pass through this useless decision point.)

If there are only two symbols of maximum length in a tree, they must have their last decision node in common, and they must be the two least probable symbols. Before we reduce a tree, the two symbols contribute

$$l_q(p_q + p_{q-1})$$

and after the reduction they contribute

$$(l_q - 1)(p_q + p_{q-1})$$

so that the code length is reduced by

$$p_q + p_{q-1}$$

If there are more than two symbols of the maximum length, we can use the following proposition: Symbols having the same length may be interchanged without changing the average code length. Using this, we can bring the two least probable symbols so that they share their final decision node. Thus after the reduction, we have shortened the code by the amount

$$p_q + p_{q-1}$$

Therefore, in either code we shorten the code and decrease the average code length *by the same amount.*

We apply this to both the decoding trees we are comparing. Since both are decreased by the same amount, the amount of inequality between their lengths is preserved. Repeated application of this will reduce both trees to two symbols. In the Huffman code the length is 1; for the other it must be less than 1, which is impossible. Therefore, for binary codes the Huffman code is the shortest possible code.

The encoding process is not unique in several respects. First, the assignment of the 0 or 1 symbols to the two source symbols at each splitting stage is arbitrary, but this produces only trivial differences. Second, when two probabilities are equal, it appears to be a matter of indifference which we put above the other in the table, although the resulting codes can have different lengths of words. However, in both cases the *average* length of the encoding of messages in these codes will be the same.

As an example of two different Huffman encodings of the same source, let

$$p_1 = 0.4$$
$$p_2 = 0.2$$
$$p_3 = 0.2$$
$$p_4 = 0.1$$
$$p_5 = 0.1$$

If at each stage we put the merged states as low as possible, then in Figure 4.8-1 we get lengths (1, 2, 3, 4, 4) and the average length is

$$L = 0.4(1) + 0.2(2) + 0.2(3) + 0.1(4) + 0.1(4) = 2.2$$

On the other hand, if we push the merged states up as high as possible (Figure 4.8-3), we will get lengths (2, 2, 2, 3, 3) and the average length is

$$L = 0.4(2) + 0.2(2) + 0.2(2) + 0.1(3) + 0.1(3) = 2.2$$

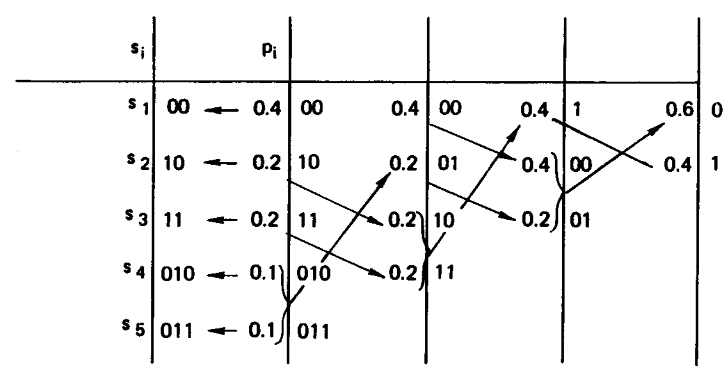


Figure 4-8.3 Alternative coding

Both codes have the same efficiency (average length) but not the same set of lengths of the symbols.

Which one of these two codes should we choose? A very reasonable choice is the one whose average length would vary least over the ensemble of messages. We therefore compute the variances in the two cases:

$$\begin{aligned}\mathrm{Var(I)} &= 0.4(1 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(3 - 2.2)^2 \\ &\quad + 0.1(4 - 2.2)^2 + 0.1(4 - 2.2)^2 = 1.36 \\ \mathrm{Var(II)} &= 0.4(2 - 2.2)^2 + 0.2(2 - 2.2)^2 + 0.2(2 - 2.2)^2 \\ &\quad + 0.1(3 - 2.2)^2 + 0.1(3 - 2.2)^2 = 0.16\end{aligned}$$

Thus the second code has significantly less variability in use on finite-length messages, and is therefore probably the preferable code. It is true that always moving a combined state as high as possible will give a minimum variance code, although we do not prove it here. By moving a merged symbol higher than it should be, we can greatly reduce the variance for a small increase in the average length.

Exercises

4.8-1 Give the Huffman code for $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{3}$, and $p_3 = \frac{1}{6}$.

4.8-2 Give the Huffman code for $p_1 = \frac{1}{3}$, $p_2 = \frac{1}{4}$, $p_3 = \frac{1}{5}$, $p_4 = \frac{1}{6}$, and $p_5 = \frac{1}{20}$.

4.8-3 Give the Huffman code for $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{4}$, $p_3 = \frac{1}{8}$, $p_4 = \frac{1}{16}$, and $p_5 = \frac{1}{16}$.

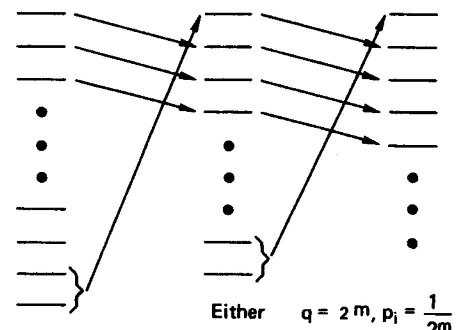
4.8-4 Give the Huffman code for $p_1 = \frac{27}{40}$, $p_2 = \frac{9}{40}$, $p_3 = \frac{3}{40}$, and $p_4 = \frac{1}{40}$.

4.8-5 Give the Huffman code for the probabilities $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{6}$, $\frac{1}{8}$, $\frac{1}{12}$, and $\frac{1}{24}$.

4.8-6 Show two distinct decoding trees for the case $\frac{1}{3}$, $\frac{1}{3}$, $\frac{1}{6}$, and $\frac{1}{6}$.

4.9 *Special Cases of Huffman Coding*

There are a number of interesting cases of Huffman coding to look at. First, if all the symbols are equally likely and if there are exactly $q = 2^m$ source symbols, then the (binary) Huffman code will be a block code



Either $q = 2^m$, $p_i = \frac{1}{2^m}$ $\rightarrow$ block code

or $p_{q-1} + p_q > p_1$ $\rightarrow$ shortened block code

Figure 4-9.1

with all symbols having the same length m (see Figure 4.9-1). If there are not exactly 2^m source symbols we will get a shortened block code (Section 4.6).

The second case is more interesting. Suppose that the two least probable symbols have their probabilities such that their sum is greater than the most probable symbol; that is, $p_{q-1} + p_q > p_1$. In the Huffman process (see Figure 4.9-1) the corresponding symbol word will go to the top. It then follows that the next two least probable symbols will combine and will go to the top, and so on. If the number of original symbols q is an exact power of 2, then the repeated combining process into the final code of two symbols will have each symbol go through *the same number* of splitting steps, and each symbol will acquire the same number of binary digits in the expansion process. Thus again we will have a block code. If q is not an exact power of 2, then, of course, there are small modifications to the foregoing process, and we get a shortened block code (Section 4.6).

It is only when the probabilities of the source symbols of the message are very different that we get a significant economy from the Huffman encoding process. This is what we would expect if we but thought a bit—only when there are large differences in the probabilities of occurrence does the variable length of the code symbols pay off.

For example, if the probabilities p_i have great variability,

$$p_j \geq \frac{2}{3} \sum_{k=j+1}^{q} p_k \tag{4.9-1}$$

for all j, then a comma code will emerge. In words, equation (4.9-1) says that each probability is at least as great as two-thirds of the sum of all the probabilities that follow.*

*Koh Wee Jin observed that (4.9-1) can be improved to $p_j \geq \sum_{k=j+2}^{q} p_k$

One way of avoiding resynchronization problems is to use a parallel channel. Another way, when using serial transmission, is to embed the message in a longer block with synchronization pulses at both ends. This, of course, wastes capacity, but it does catch spurious pulses.

With the current low cost of accurate crystal clocks it is common practice to use clock pulses either from a single clock, or by keeping several clocks in synchronization, so that both the sender and the receiver have the same timing pulses.

The topic is not of enough general interest to do more than point out the problem and the obvious solutions. Many other methods of code compression can be found in Ref. [DG].

Chapter 6

Entropy and Shannon's First Theorem

6.1 Introduction

Up to this point we have been concerned with coding theory; now we begin information theory. Coding theory answers the questions of (1) how to design codes for white noise (mainly), and (2) how to compress the message when the probabilities (structure) of the messages are known.

We now need a general method for measuring the structure of the source. To do this we introduce the concept of *entropy*. While entropy was used long ago in many physical situations, and the concept in information theory bears a strong resemblance to the classical definition, we shall study entropy on its own merits in this particular field, and not get involved in the various suggestive analogies. For us, entropy is simply a function of a probability distribution p_i. For some analogies, see Refs. [Ga], [Gu], and [ME].

Information theory combines noise protection and efficient use of the channel into a single theory. However, the simple model of channel noise (white noise) is sometimes unrealistic, and we will occasionally treat more general patterns of errors. This leads to the important concept of *channel capacity*, which is taken up in Chapter 8.

The result we are headed for is Shannon's main theorem in Chapter 10, which gives a relationship between the channel capacity C (to be defined in Chapter 8) and the maximum rate of signaling possible. We will prove the remarkable theorem that we can come arbitrarily close to the maximum rate of signaling and also achieve an arbitrarily low rate of error. In a sense this achieves both *Huffman-type* compression and *Hamming-type* noise protection in the same code. Unfortunately, the proof is not constructive, so that while information theory sets the bounds on what can be done, it does not tell us how to achieve them. Yet, as we said in Section 1.2, the theory is very useful.

In this chapter we prove a very special case of the main theorem, namely the *noiseless coding theorem*, where we ignore the problem of noise. When this theorem and its proof are understood, the proof of the main theorem will appear a bit clearer—once a lot of preliminary mathematical results are covered in Chapter 9.

6.2 Information

Suppose that we have the source alphabet of q symbols $s_1, s_2, \ldots, s_q$, each with its probability $p(s_1) = p_1, p(s_2) = p_2, \ldots, p(s_q) = p_q$. When we receive one of these symbols, how much information do we get? For example, if $p_1 = 1$ (and, of course, all the other $p_i = 0$), then there is no "surprise," no information, since you know what the message must be. On the other hand, if the probabilities are all very different, then when a symbol with a low probability arrives, you feel more surprised, get more information, than when a symbol with a higher probability arrives. Thus information is somewhat inversely related to the probability of occurrence.

We wish to construct this function $I(p)$, which measures the amount of information—surprise, uncertainty—in the occurrence of an event of probability p. We assume three things about $I(p)$.

1. $I(p) \geq 0$ (a real nonnegative measure).
2. $I(p_1 p_2) = I(p_1) + I(p_2)$ for independent events (additive).
3. $I(p)$ is a continuous function of p.

The second of these conditions is known as the *Cauchy functional equation* for the function $I(p)$, meaning that it serves to define $I(p)$. If p_1 and p_2 are both the same number p, not necessarily the same event, then

$$I(p^2) = I(p) + I(p) = 2I(p)$$

Now if $p_1 = p$ and $p_2 = p^2$, then we have

$$I(p^3) = I(p) + I(p^2) = 3I(p)$$

and in general we have

$$I(p^n) = nI(p)$$

We recognize that the standard law of exponents for positive integers applies to the function $I(p)$. Following this clue we adapt for our needs the usual exponent extension to fractional values. We set

$$p^n = y, \qquad p = y^{1/n}$$

and hence

$$I(y) = nI(y^{1/n})$$

or, after some further manipulation,

$$I(y^{m/n}) = \frac{m}{n} I(y)$$

Thus for the rational numbers the function of $I(p)$ obeys the same formula as the log function. Assumption 3 of continuity allows us to extend this to all numbers ($0 \leq p \leq 1$), rational or irrational. Thus we have

$$I(p) = k \log p$$

for some constant k and some base of the log system. From the first assumption it is natural to pick the constant k as -1, and we have, finally,

$$I(p) = -\log p = \log \frac{1}{p}$$

for some base of the log system.

Let us examine and illustrate this *additive* property 2 that we assumed, since it plays so central a role in deriving the measure of surprise. Consider the simultaneous and independent toss of a coin and roll of a die. We *feel* that each outcome (coin and die) has no influence on the outcome of the other. The coin has two equally likely outcomes and the die has 6. The product space of the two independent trials has 12 equally likely outcomes. If c_i is the outcome of the coin and d_j is the outcome of the die, then the derived formula of $I(p)$ is

$$I(c_i d_j) = I(c_i) + I(d_j)$$

$$\log \tfrac{1}{12} = \log \tfrac{1}{2} + \log \tfrac{1}{6}$$

which is, of course, true.

The words "uncertainty," "surprise," and "information" are related. Before the event (experiment, reception of a message symbol, etc.) there is the amount of uncertainty; when the event happens there is the amount of surprise; and after the event there is the gain in the amount of information. All these amounts are the same.

This is an engineering definition based on probabilities and is not a definition based on the meaning of the symbols to the human receiver. The confusion at this point has been very great for outsiders who glance at information theory; they fail to grasp that this is a highly technical definition that captures *only part* of the richness of the usual idea of information.

What base of the log system shall we use? It is simply a matter of convention since any set of logs is proportional to any other set. This follows directly from the fundamental relationship for logs:

$$\log_a x = \frac{\log_b x}{\log_b a} = (\log_a b) \log_b x$$

It is convenient to use the base 2 logs; the resulting unit of information is called a *bit* (binary digit). If we use base e, as we must whenever we get into the calculus, then the unit of information is called a *nat*. Finally, sometimes the base 10 is used and the unit is called a *Hartley*, after R. V. L. Hartley, who first proposed the use of the logarithmic measure of information.

We are using the word "bit" in two different ways, both as a digit in the number base 2 and as a unit of information. They are not the same, and we shall be careful to say "bit of information" when we mean that definition and there could be confusion.

It is easy to convert logs from one base to another using

$$\log_{10} 2 = 0.30103\ldots \qquad \log_2 10 = 3.32193\ldots$$

$$\log_{10} e = 0.43429\ldots \qquad \log_e 10 = 2.30259\ldots$$

$$\log_e 2 = 0.69315\ldots \qquad \log_2 e = 1.44270\ldots$$

We have just found one solution to the Cauchy functional equation, namely $I(p) = \log_2 (1/p)$. Is this solution unique, to within the base of the log system? Suppose that there were another solution to the derived functional equation

$$I(p^n) = nI(p)$$

Call it $g(p)$. Hence we have the equation

$$g(p^n) = ng(p)$$

Now take the difference of the two solutions and allow for the constant of proportionality C, which depends on the base

$$g(p^n) - C \log_2 \frac{1}{p^n} = n\left[g(p) - C \log_2 \frac{1}{p}\right]$$

Next, pick C as ($p_0 \neq 0, 1$)

$$C = \frac{g(p_0)}{\log_2(1/p_0)}$$

This choice makes the right-hand side of the equation above equal to zero at p_0. Now for any number z there is an n such that (assuming p_0 neither 0 nor 1)

$$z = p_0^n$$

Hence the left-hand side of the equation becomes

$$g(z) = C \log_2 \frac{1}{z}$$

and the solution is essentially unique. The continuity assumption covers the two values of p that were omitted, namely 0 and 1.

6.3 Entropy

If we get $I(s_i)$ units of information when we receive the symbol s_i, how much do we get on the average? The answer is that since p_i is the probability of getting the information $I(s_i)$, then on the average we get for each symbol s_i,

$$p_i I(s_i) = p_i \log_2 \frac{1}{p_i}$$

From this it follows that *on the average*, over the whole alphabet of symbols s_i, we will get

$$\sum_{i=1}^{q} p_i \log_2 \frac{1}{p_i}$$

Following custom, we label this important quantity (for radix r)

$$H_r(S) = \sum_{i=1}^{q} p_i \log_r \left(\frac{1}{p_i}\right) \tag{6.3-1}$$

and call it the *entropy* of the signaling system S having symbols s_i and probabilities p_i. Of course,

$$H_r(S) = H_2(S) \log_r 2$$

This is the entropy function for a distribution *when all that is considered are the probabilities* p_i of the symbols s_i. In Section 6.10 we will consider the entropy of a Markov process.

Corresponding to each distribution $P = (p_1, p_2, \ldots, p_q)$ of symbols s_i, there is a single number called the entropy and labeled $H(S)$. This is analogous to the usual idea of an average of a distribution—the average is a single number which, in some sense, summarizes the distribution. The entropy $H(S)$ is the weighted average of the logs of the reciprocals of the probabilities of the distribution. The entropy is a single measure of a distribution; it is the average information of the alphabet S.

As an alternative approach to the entropy function, consider that in a long message of N symbols (from the alphabet S) you expect Np_1 of the first symbol; Np_2 of the second, and so on. The probability P of a message of length N with these numbers of symbols is exactly

$$\begin{aligned} P &= p_1^{(Np_1)} p_2^{(Np_2)} \ldots p_q^{(Np_q)} \\ &= [p_1^{(p_1)} p_2^{(p_2)} \ldots p_q^{(p_q)}]^N \end{aligned}$$

Hence the information is

$$\log\left(\frac{1}{P}\right) = N \sum_{i=1}^{q} p_i \log\left(\frac{1}{p_i}\right)$$

and the information *per symbol* of the alphabet S is

$$H(S) = \sum \left(p_i \log \frac{1}{p_1}\right)$$

which is again the entropy (as it should be).

It is important to realize that a remark like "Consider the entropy of the source" can have no meaning unless a model of the source is included. Using random numbers as an example, you have formulas for generating *pseudorandom* numbers. Suppose that we had a table of such numbers. If you do not recognize that they are pseudorandom numbers, then you would probably compute the entropy based on the frequencies of occurrence of the individual numbers. Since pseudorandom-number generators do a good job of simulating random numbers, you would also find that each new number came as a complete surprise. But if you knew the structure of the formula used to generate the table, you would, after a few numbers, be able to predict perfectly the next number—there would be no surprise. Your estimate of the entropy of a source of symbols therefore depends on the model you adopt of the structure of the symbols.

The entropy function (6.3-1) involves only the distribution of the probabilities—it is a function of a probability distribution p_i and does not involve the s_i. For example, if the probabilities are 0.4, 0.3, 0.2, and 0.1, then from the table of Appendix B, column 3, we have

p	$p \log \frac{1}{p}$
0.4	0.52877
0.3	0.52109
0.2	0.46439
0.1	0.33219
Sum	1.84644

The entropy of this distribution is therefore 1.84644 (approximately). Although we should write the entropy function $H(S)$ as a function of

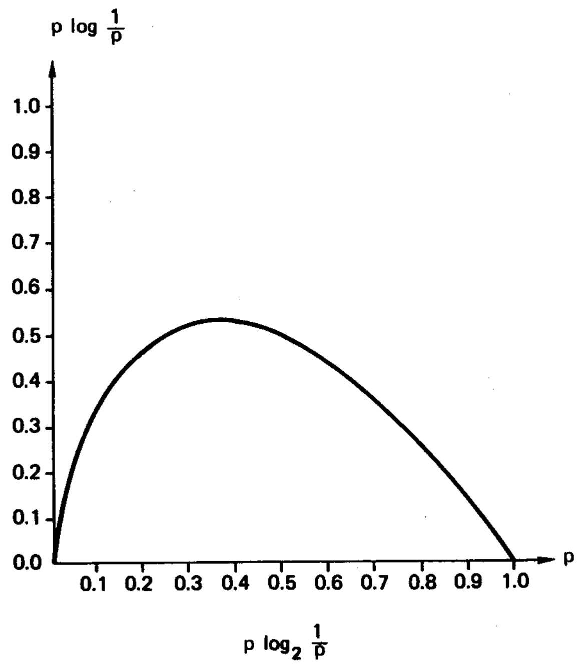


Figure 6-3.1

P, we will continue to refer to the alphabet S and write $H(S)$, although occasionally we will write $H(P)$, when there are only two events, with probabilities p and $(1 - p)$.

The function of $p \log_2(1/p)$ is graphed in Figure 6.3-1. From

$$\frac{d}{dp}\left(p \log_2 \frac{1}{p}\right) = \frac{d}{dp}\left(p \log_e \frac{1}{p}\right) \log_2 e = \log_2 \left(\frac{1}{p}\right) - \log_2 e$$

We see both the infinite slope at $p = 0$ and that the maximum occurs at $p = 1/e$, where the derivative is zero.

We also need the property (since it is equated to zero, we can ignore the factor $\log_e 2$)

$$\lim_{x \to 0} (x \log_e x) = 0$$

To prove this, we write it as

$$\lim_{x \to 0} \left(\frac{\log_e x}{1/x}\right)$$

and apply l'Hôpital's rule by differentiating both numerator and denominator separately:

$$\lim_{x \to 0} \left(\frac{1/x}{-1/x^2}\right) = \lim_{x \to 0} (-x) = 0$$

As another example of the entropy of a distribution, consider the source alphabet $S = \{s_1, s_2, s_3, s_4\}$, where $p_1 = \frac{1}{2}$, $p_2 = \frac{1}{4}$, $p_3 = \frac{1}{8}$, and $p_4 = \frac{1}{8}$. With $r = 2$ we have the entropy

$$\begin{aligned} H_2(S) &= \tfrac{1}{2} \log_2 2 + \tfrac{1}{4} \log_2 4 + \tfrac{1}{8} \log_2 8 + \tfrac{1}{8} \log_2 8 \\ &= (\tfrac{1}{2})1 + (\tfrac{1}{4})2 + (\tfrac{1}{8})3 + (\tfrac{1}{8})3 \\ &= 1\tfrac{3}{4} \text{ bits of information} \end{aligned}$$

As one application of the idea of entropy, consider the toss of a coin when both sides are considered to be equally likely.

$$I(s_i) = \log_2 (\tfrac{1}{2})^{-1} = \log_2 2 = 1$$

$$H_2(S) = (\tfrac{1}{2})I(s_1) + (\tfrac{1}{2})I(s_2) = 1$$

The distribution consisting of just two events is very common. If p is the probability of the first symbol (event), then the entropy function is

$$H_2(P) = p \log_2 \left(\frac{1}{p}\right) + (1 - p) \log_2 \left(\frac{1}{1 - p}\right)$$

and is tabulated in the last column of the table in Appendix B. The graph of this function is given in Figure 6.3-2. Note that at $p = 0$ and $p = 1$ it has a vertical tangent, since

$$\begin{aligned} \frac{d}{dp}&\left[p \log_2 \left(\frac{1}{p}\right) + (1 - p) \log_2 \left(\frac{1}{1 - p}\right)\right] \\ &= \left[\log_e \left(\frac{1}{p}\right) - 1 - \log_e \left(\frac{1}{1 - p}\right) + 1\right] \log_2 e \\ &= \log_2 \left(\frac{1}{p}\right) - \log_2 \left(\frac{1}{1 - p}\right) \end{aligned}$$

An application similar to the coin is the roll of a well-balanced die (singular of dice). We have

$$I(s_i) = \log_2 \left(\frac{1}{6}\right)^{-1} = \log_2 6$$

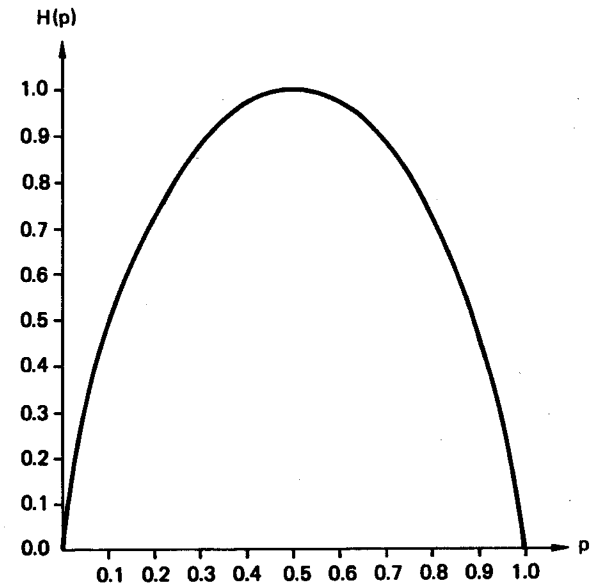


Figure 6-3.2 Entropy function for two probabilities

and the entropy is

$$H(S) = 6[\tfrac{1}{6}I(s_i)] = \log_2 6$$
$$= 2.5849\ldots \text{ bits of information}$$

As we see from this example, whenever all the probabilities are equal, the average over the alphabet is the same as the information for any one event.

A final example of the computation of the entropy of a distribution is the entropy of Zipf's law (Section 4.9), where the frequency of the kth item is proportional to $1/k$. Thus for N items, we have the probabilities

$$p_k = \frac{1/k}{S_N} \qquad (k = 1, 2, \ldots, N)$$

where

$$S_N = \sum_{k=1}^{N} \frac{1}{k}$$

The entropy for the N items is

$$\begin{aligned} H_2(N) &= -\sum_{k=1}^{N} p_k \log_2 p_k \\ &= -\sum_{1}^{N} \frac{1/k}{S_N}\left[\log_2\left(\frac{1}{k}\right) - \log_2 S_N\right] \\ &= -\frac{1}{S_N}\left[\sum_{1}^{N} \frac{1}{k}\left(\log_2 \frac{1}{k} - \log_2 S_N\right)\right] \\ &= \frac{1}{S_N}\left(\sum_{1}^{N} \frac{\log_2 k}{k} + \log_2 S_N \sum_{1}^{N} \frac{1}{k}\right) \\ &= \frac{T_N}{S_N} + \log_2 S_N \end{aligned}$$

where

$$T_N = \sum_{k=1}^{N} \frac{\log_2 k}{k}$$

Using the recursive definitions

$$S_1 = 1$$
$$T_1 = 0$$
$$S_N = S_{N-1} + \frac{1}{N}$$
$$T_N = T_{N-1} + \frac{\log_2 N}{N}$$

then

$$H_2(N) = \frac{T_N}{S_N} + \log_2 S_N$$

is easily computed.

We have Table 6.3-1 (corresponding to Table 4.9-1).

The entropy function of a distribution summarizes one aspect of a distribution much as the *average* in statistics summarizes a distribution. The entropy has properties of both the arithmetic mean (the average)

TABLE 6.3-1 *Entropy of Zipf's Law*

N	Entropy	Huffman $L_H(N)$
2	0.918	1.00
4	1.792	1.80
8	2.618	2.68
16	3.403	3.43
32	4.149	4.17
64	4.864	4.89
128	5.553	5.60
256	6.222	6.26
512	6.873	6.90
1024	7.511	7.54

and the geometric mean. We have

$$\begin{aligned} H(S) &= \sum p_i \log_2 \left(\frac{1}{p_i}\right) \\ &= \sum \log_2 \left(\frac{1}{p_i}\right)^{p_i} \\ &= \log_2 \prod_{i=1}^{q} \left(\frac{1}{p_i}\right)^{p_i} \end{aligned}$$

which is a weighted geometric mean.

Exercises

6.3-1 How many bits of information do we get from one draw of a card from a deck of 52 cards?

6.3-2 For $p_1 = \frac{1}{3}, p_2 = \frac{1}{4}, p_3 = \frac{1}{6}, p_4 = \frac{1}{9}, p_5 = \frac{1}{12}$, and $p_6 = \frac{1}{18}$, compute the entropy. Ans. $\frac{8}{9} + \frac{11}{12} \log 3 \simeq 2.34177$

6.4 *Mathematical Properties of the Entropy Function*

The entropy function measures the average amount of uncertainty, surprise, or information that we get from the outcome of some situation, say the reception of a message or the outcome of some experiment. It is therefore an important function of the probabilities of the individual events that can occur. Thus in designing an experiment we usually wish

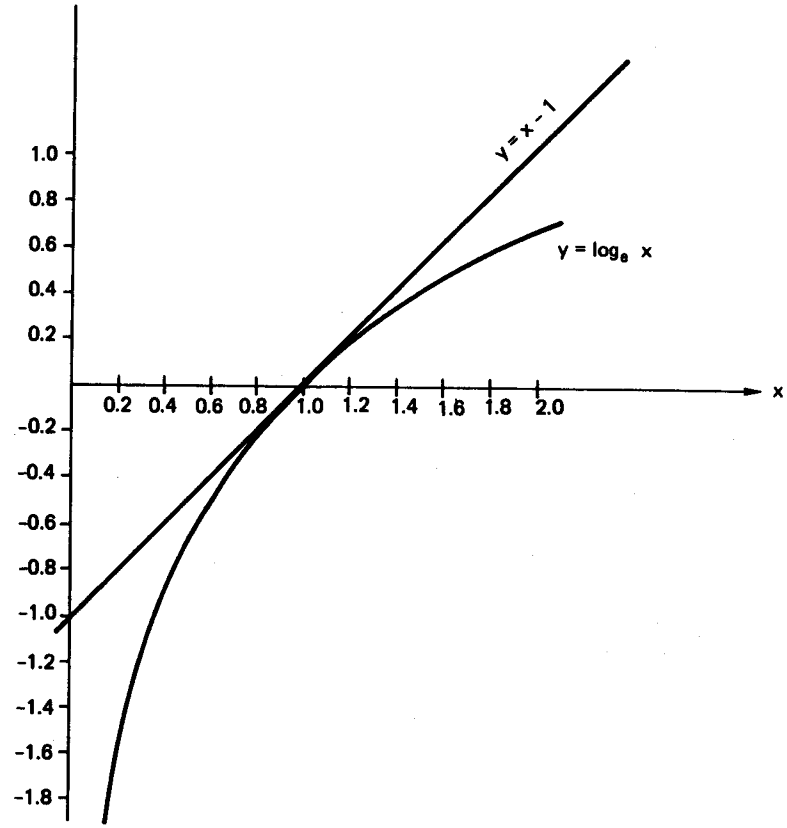


Figure 6-4.1 Bound on $\log_e x$ function

to maximize the amount of information we expect to get; we wish to maximize the entropy function. To do this we need to at least partially control the probabilities of the individual outcomes of the experiment; we need to "design the experiment" suitably. The maximum entropy is increasingly used as a criterion in many situations. Therefore, the entropy function is worth some study for its own sake, independent of the particular applications we will make of it.

The entropy function has a number of mathematical properties that are very useful, and we examine them before getting deeper into the theory. A first property of the $\log_e x$ function can be seen from Figure 6.4-1.

Fitting the tangent line at the point (1, 0), we find that the slope is

$$\left.\frac{d(\log_e x)}{dx}\right|_{x=1} = 1$$

so that the tangent line is

$$y - 0 = 1(x - 1)$$

or

$$y = x - 1$$

Thus we have for all x greater than zero the useful inequality

$$\log_e x \le x - 1 \tag{6.4-1}$$

or

$$\log_2 x \le (x - 1)\log_2 e$$

The equality holds *only* at the point $x = 1$.

The second result we need is the fundamental relationship between two probability distributions. Let the first probability distribution be x_i with of course $\Sigma\, x_i = 1$, and the second probability distribution be y_i with correspondingly $\Sigma\, y_i = 1$. Consider, now, the expression involving both distributions

$$\sum_{i=1}^{q} x_i \log_2\left(\frac{y_i}{x_i}\right) = \frac{1}{\log_e 2}\sum_{i=1}^{q} x_i \log_e\left(\frac{y_i}{x_i}\right)$$

Using the previous relation (6.4-1), we get

$$\begin{aligned}\frac{1}{\log_e 2}\sum_i x_i \log_e \frac{y_i}{x_i} &\le \frac{1}{\log_e 2}\sum_i x_i\left(\frac{y_i}{x_i} - 1\right)\\ &\le \frac{1}{\log_e 2}\sum_i (y_i - x_i)\\ &\le \frac{1}{\log_e 2}\left(\sum_i y_i - \sum_i x_i\right) = 0\end{aligned}$$

Converting back to logs base 2, we have the *fundamental Gibbs inequality*

$$\sum_{i=1}^{q} x_i \log_2\left(\frac{y_i}{x_i}\right) \le 0 \tag{6.4-2}$$

Note that the equality holds *only* when *all* the $x_i = y_i$.

It is natural to ask for the conditions on a probability distribution that lead to the maximum entropy (the minimum clearly occurs when one $p_i = 1$ and all the others $= 0$). We have

$$H_2(S) = \sum_i p_i \log_2 \frac{1}{p_i} \qquad \text{with} \qquad \sum_i p_i = 1$$

We begin by considering the quantity

$$\begin{aligned}H_2(S) - \log_2 q &= \sum_{i=1}^{q} p_i \log_2\left(\frac{1}{p_i}\right) - \log_2 q \sum_{i=1}^{q} p_i\\ &= \sum_{i=1}^{q} p_i \log_2\left(\frac{1}{qp_i}\right)\end{aligned}$$

Using the Gibbs inequality (6.4-2) with $y_i = 1/q$, we have

$$H_2(S) - \log_2 q \le 0$$

Therefore,

$$H_2(S) \le \log_2 q \tag{6.4-3}$$

For equality in (6.4-3) we must have all the $p_i = 1/q$.

For an alternative derivation of this important result, we use the Lagrange multipliers, and consider the function

$$f(p_1, p_2, \ldots, p_q) = \frac{1}{\log_e 2}\sum_i p_i \log_e\left(\frac{1}{p_i}\right) + \lambda\left(\sum_i p_i - 1\right)$$

$$\frac{\partial f}{\partial p_j} = \frac{1}{\log_e 2}\left[\log_e\left(\frac{1}{p_j}\right) - 1\right] + \lambda = 0 \qquad (j = 1, 2, \ldots, q)$$

Since everything in this equation is a constant, it follows that each p_j is the same constant. If all the p_j are equal, each has the value $1/q$, where, as usual, the distribution has q members. Thus the maximum entropy is

$$H_2(S) = \log_2 q$$

For any other distribution than the equally likely distribution, the entropy is less than $\log_2 q$.

Based on this result, we now ask: What book has the maximum information? Of course, the answer depends on the size of the book, the size of the type font, the variety of the type font, and so on. Supposing these to be fixed; then the answer is obviously, the book with uniformly random letters! Each letter will come as a complete surprise!

This shows how far the formal engineering definition of information is from the usual human meaning of information. Yes, the definition of information that Shannon made is proper for computing machines, for the telephone system, and for communication systems, but flies in the face of the normal human meaning of information. It was the failure to notice that the definition is appropriate for machines and for machine-like situations, but is not appropriate for what we normally mean by "information," that led to so many misapplications of information theory in the early days. Now that we understand the limitations of the definition we are better able to apply information theory usefully. The very limitation of the definition to machines is what makes the definition so useful in such situations—the definition eliminates much that is vague, uncertain, and confusing. By limiting the theory to a sharp definition, we can get sharp, useful results in many situations. Perhaps it would have been better had Shannon called it *communication theory*, but the title *information theory* is often appropriate.

A simple, familiar application of this principle—that the uniform distribution contains the most information—is given by the standard grading system of A, B, C, D, and F. Setting aside F as having special properties, namely that the student must take the course again, *if* we want to communicate the maximum amount of information with the grading system, then we should use all the other grades equally often. Of course, we may want especially to distinguish the very best, and give comparatively few A's. This is, however, giving a different meaning to information than we have defined. The common habit in graduate schools of giving only A's and B's is a plain waste of signaling capacity. The extreme of one probability being 1 and all the others therefore being 0 is the case of a constant signal. There is no information transmitted. "The constant complainer is soon ignored."

In dealing with a new situation it is often useful to look at the extremes that can arise. For entropy of a distribution $H_2(S)$ we get the least value, for one extreme, $p_1 = 1$, and all other $p_i = 0$—no surprise:

$$H_2(1, 0, 0, \ldots, 0) = 0$$

As we proved, the maximum occurs when all the probabilities are equal, namely

$$H_2\left(\frac{1}{q}, \frac{1}{q}, \ldots, \frac{1}{q}\right) = ln_2 q$$

6.5 Entropy and Coding

We now prove a fundamental relationship between the average code length L and the entropy $H(S)$. Given *any instantaneous code* it has some definite code word lengths l_i represented in some radix r. From the Kraft inequality (4.5-1), we have

$$K = \sum_{i=1}^{q} \left(\frac{1}{r^{l_i}}\right) \leq 1 \qquad (6.5\text{-}1)$$

We now define the numbers Q_i (pseudo probabilities):

$$Q_i = \frac{r^{-l_i}}{K} \qquad (6.5\text{-}2)$$

where, of course,

$$\sum_{i=1}^{q} Q_i = 1$$

The Q_i may be regarded as a probability distribution. Therefore, we can use the fundamental Gibbs inequality (6.4-2),

$$\sum_{i=1}^{q} p_i \log_2 \left(\frac{Q_i}{p_i}\right) \leq 0$$

Upon expanding the log term into a sum of logs, we notice that one term leads to the entropy function,

$$H_2(S) = \sum_{i=1}^{q} p_i \log_2 \left(\frac{1}{p_i}\right) \le \sum_{i=1}^{q} p_i \log_2 \left(\frac{1}{Q_i}\right)$$

Using (6.5-2) in the right-hand side, we obtain

$$\le \sum_{i=1}^{q} p_i(\log_2 K - \log_2 r^{-l_i})$$

$$\le \log_2 K + \sum_{i=1}^{q} p_i l_i \log_2 r$$

By the Kraft inequality $K \le 1$, so that $\log_2 K \le 0$. Dropping this term can only strengthen the inequality. We have, therefore,

$$H_2(S) \le \sum_{i=1}^{q} (p_i l_i) \log_2 r = L \log_2 r$$

or

$$H_r(S) \le \mathrm{L} \tag{6.5-3}$$

where L is the average code word length,

$$L = \sum_{i=1}^{q} p_i l_i \tag{6.5-4}$$

This is the fundamental result that we need; the entropy supplies a lower bound on the average code length L for any instantaneous decodable system. By the McMillan inequality of Section 4.7, it also applies to any uniquely decodable system.

For efficient binary codes $K = 1$ and we have $\log_2 K = 0$. Therefore, the inequality occurs (in the binary case) only when

$$p_i \ne Q_i = 2^{-l_i}$$

Table 6.3-1 shows how Huffman coding approaches the entropy for the Zipf distribution.

6.6 Shannon–Fano Coding

In Section 6.5 we assumed that the code word lengths l_i were given. Suppose, as is more likely, that the probabilities p_i are given. Huffman coding gives the lengths, but that method makes each length depend on the *whole* set of probabilities.

Shannon–Fano coding is less efficient than is Huffman coding, but has the advantage that you can go directly from the probability p_i to the code word length l_i. Given the source symbols $s_1, s_2, \ldots, s_q$ and their corresponding probabilities $p_1, p_2, \ldots, p_q$, then for each p_i there is an integer l_i such that

$$\log_r \left(\frac{1}{p_i}\right) \le l_i < \log_r \left(\frac{1}{p_i}\right) + 1 \tag{6.6-1}$$

since the two extreme values just span a unit length. Removing the logs, we get

$$\left(\frac{1}{p_i}\right) \le r^{l_i} < \left(\frac{r}{p_i}\right)$$

Take the reciprocal of each term. We obtain

$$p_i \ge \left(\frac{1}{r^{l_i}}\right) > \frac{p_i}{r}$$

Since $\Sigma\, p_i = 1$, when we sum this inequality, we get

$$1 \ge \sum_{i=1}^{q} \left(\frac{1}{r^{l_i}}\right) > \frac{1}{r} \tag{6.6-2}$$

which gives the Kraft inequality. Therefore, there is an instantaneous decodable code having these Shannon–Fano lengths.

To get the entropy of the distribution of p_i, we multiply the equation (6.6-1) by p_i and sum:

$$H_r(S) = \sum_{i=1}^{q} p_i \log_r \frac{1}{p_i} \le \sum_{i=1}^{q} p_i l_i < H_r(S) + 1$$

In terms of the average length L of the code (6.5-4), we have

$$H_r(S) \le \mathrm{L} < H_r(S) + 1 \tag{6.6-3}$$

Thus for Shannon–Fano coding we again have the entropy as a lower bound on the average length of the code. It is also part of the upper bound. It is not as easy to directly find an upper bound for Huffman coding (Chapter 4), but since Huffman coding is optimal, it is at least as good as Shannon–Fano.

How do we find the actual code symbols? We simply assign them in order. For example, from the probabilities

$$p_1 = p_2 = \tfrac{1}{4}, \qquad p_3 = p_4 = p_5 = p_6 = \tfrac{1}{8}$$

we get the Shannon–Fano lengths,

$$l_1 = l_2 = 2, \qquad l_3 = l_4 = l_5 = l_6 = 3$$

We then assign

$$\begin{array}{ll} s_1 = 00 & s_3 = 100 \\ s_2 = 01 & s_4 = 101 \\ & s_5 = 110 \\ & s_6 = 111 \end{array}$$

We are assured by the Kraft inequality, which we showed that Shannon–Fano coding obeys, that there are always enough symbols to assign for an instantaneous code. Our orderly assignment leads readily to the decoding tree and the prefix condition is met.

This shows that we can do fairly well even with Shannon–Fano coding, but it is interesting to see how the left-hand inequality arises. First, if each probability p_i were exactly equal to the reciprocal of a power of the radix r, then we would have equality for this assignment of code word lengths—the average word length would be exactly the entropy. To get the code words we simply assign $r$ary numbers in increasing sequence and of the required lengths.

6.7 *How Bad Is Shannon–Fano Coding?*

Since Huffman coding is optimal, and we have temporarily descended to the less-than-optimal (at times) Shannon–Fano coding, it is reasonable to ask "How bad is Shannon–Fano coding?" Consider the following examples.

For a source alphabet s_1, s_2 with probabilities

$$p_1 = 1 - \frac{1}{2^k}, \qquad p_2 = \frac{1}{2^k} \qquad (k \geq 2)$$

we get

$$\log_2\left(\frac{1}{p_1}\right) \leq \log_2 2 = 1$$

Thus $l_1 = 1$. But for l_2, we have

$$\log_2\left(\frac{1}{p_2}\right) = \log_2 2^k = k = l_2$$

Hence where the Huffman encoding gives both code words one binary digit, Shannon–Fano has s_1, a 1-bit word, and s_2, a k-bit word.

Before getting too worried about this inefficiency, let us compute the average word length. Clearly, Huffman has

$$L_H = 1$$

For Shannon–Fano, we have

$$L_{SF} = 1\left(1 - \frac{1}{2^k}\right) + k\left(\frac{1}{2^k}\right) = 1 + \left(\frac{k-1}{2^k}\right)$$

We have the table

k	$1 + \frac{k-1}{2^k}$
2	$1 + \frac{1}{4} = 1.25$
3	$1 + \frac{1}{4} = 1.25$
4	$1 + \frac{3}{16} = 1.1875$
5	$1 + \frac{1}{8} = 1.125$
6	$1 + \frac{5}{64} = 1.078125$
	etc.