5

Entropy and Shannon's Source Coding Theorem

Up to this point we have been concerned with coding theory. We have described codes and given algorithms of how to design them. And we have evaluated the performance of some *particular* codes. Now we begin with information theory, which will enable us to learn more about the fundamental properties of *general* codes without having actually to design them.

Basically, information theory is a part of physics and tries to describe what information is and how we can work with it. Like all theories in physics it is a model of the real world that is accepted as true as long as it predicts how nature behaves accurately enough.

In the following we will start by giving some suggestive examples to motivate the definitions that follow. However, note that these examples are not a justification for the definitions; they just try to shed some light on the reason why we will define these quantities in the way we do. The real justification of all definitions in information theory (or any other physical theory) is the fact that they turn out to be useful.

5.1 Motivation

We start by asking the question: what is information?

Let us consider some examples of sentences that contain some "information."

- The weather will be good tomorrow.
- The weather was bad last Sunday.
- The president of Taiwan will come to you tomorrow and will give you one million dollars.

The second statement seems not very interesting as you might already know

what the weather was like last Sunday. The last statement is much more exciting than the first two and therefore seems to contain much more information. But, on the other hand, do you actually believe it? Do you think it is likely that you will receive one million dollars tomorrow?

Let us consider some easier examples.

- You ask: "Is the temperature in Taiwan currently above 30 degrees?"
  This question has only two possible answers: "yes" or "no."
- You ask: "The president of Taiwan has spoken with a certain person from Hsinchu today. With whom?"
  Here, the question has about 400 000 possible answers (since Hsinchu has about 400 000 inhabitants).

Obviously the second answer provides you with a much bigger amount of information than the first one. We learn the following.

> The number of possible answers r should be linked to "information."

Here is another example.

- You observe a gambler throwing a fair dice. There are six possible outcomes $\{1,2,3,4,5,6\}$. You note the outcome and then tell it to a friend. By doing so you give your friend a certain amount of information.
- Next you observe the gambler throwing the dice *three times*. Again, you note the three outcomes and tell them to your friend. Obviously, the amount of information that you give to your friend this time is three times as much as the first time.

So we learn the following.

> "Information" should be additive in some sense.

Now we face a new problem: regarding the example of the gambler above we see that in the first case we have $r = 6$ possible answers, while in the second case we have $r = 6^3 = 216$ possible answers. Hence in the second experiment there are 36 times more possible outcomes than in the first experiment! But we would like to have only a three times larger amount of information. So how do we solve this?

Idea: use a logarithm. Then the exponent 3 will become a factor exactly as we wish: $\log_b 6^3 = 3 \cdot \log_b 6$.

Exactly these observations have been made by the researcher Ralph Hartley in 1928 in Bell Labs [Har28]. He gave the following definition.

Definition 5.1 We define the following measure of information:

$$\tilde{I}(U) \triangleq \log_b r, \tag{5.1}$$

where r is the number of all possible outcomes of a random message U.

Using this definition we can confirm that it has the wanted property of additivity:

$$\tilde{I}(U_1, U_2, \ldots, U_n) = \log_b r^n = n \cdot \log_b r = n \cdot \tilde{I}(U). \tag{5.2}$$

Hartley also correctly noted that the basis b of the logarithm is not really important for this measure. It only decides on the *unit of information*. So, similarly to the fact that 1 km is the same distance as 1000 m, b is only a change of units without actually changing the amount of information it describes.

For two important and one unimportant special cases of b it has been agreed to use the following names for these units:

$b = 2$ ($\log_2$):	bit,
$b = e$ (ln):	nat (natural logarithm),
$b = 10$ ($\log_{10}$):	Hartley.

Note that the unit Hartley has been chosen in honor of the first researcher who made the first (partially correct) attempt at defining information. However, as nobody in the world ever uses the basis $b = 10$ for measuring information, this honor is questionable.

The measure $\tilde{I}(U)$ is the right answer to many technical problems.

Example 5.2 A village has eight telephones. How long must the phone number be? Or, asked differently: how many bits of information do we need to send to the central office so that we are connected to a particular phone?

$$8 \text{ phones} \quad \Longrightarrow \quad \log_2 8 = 3 \text{ bits}. \tag{5.3}$$

We choose the following phone numbers:

$$\{000, 001, 010, 011, 100, 101, 110, 111\}. \tag{5.4}$$

$\Diamond$

In spite of its usefulness, Hartley's definition had no effect whatsoever in the world. That's life... On the other hand, it must be admitted that Hartley's definition has a fundamental flaw. To realize that something must be wrong, note that according to (5.1) the *smallest* nonzero amount of information is

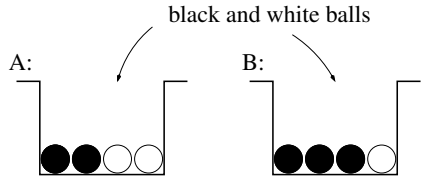

Figure 5.1 Two hats with four balls each.

$\log_2 2 = 1$ bit. This might sound like only a small amount of information, but actually 1 bit can be *a lot* of information! As an example, consider the 1-bit (yes or no) answer if a man asks a woman whether she wants to marry him. If you still do not believe that one bit is a huge amount of information, consider the following example.

Example 5.3 Currently there are 6 902 106 897 persons living on our planet (U.S. Census Bureau, 25 February 2011, 13:43 Taiwan time). How long must a binary telephone number U be if we want to be able to connect to every person?

According to Hartley we need $\tilde{I}(U) = \log_2(6902106897) \simeq 32.7$ bits. So with only 33 bits we can address every single person on this planet. Or, in other words, we only need 33 times 1 bit in order to distinguish every human being alive. ◊

We see that 1 bit is a lot of information and it cannot be that this is the smallest amount of (nonzero) information.

To understand more deeply what is wrong, consider the two hats shown in Figure 5.1. Each hat contains four balls, where the balls can be either white or black. Let us draw one ball at random and let U be the color of the ball. In hat A we have $r = 2$ colors: black and white, i.e. $\tilde{I}(U_A) = \log_2 2 = 1$ bit. In hat B we also have $r = 2$ colors and hence also $\tilde{I}(U_B) = 1$ bit. But obviously, we get less information if in hat B black shows up, since we somehow expect black to show up in the first place. Black is much more *likely*!

We realize the following.

> A proper measure of information needs to take into account the probabilities of the various possible events.

This was observed for the first time by Claude Elwood Shannon in 1948 in

his landmark paper "A mathematical theory of communication" [Sha48]. This paper has been like an explosion in the research community![1]

Before 1948, the engineering community was mainly interested in the behavior of a sinusoidal waveform that is passed through a communication system. Shannon, however, asked why we want to transmit a deterministic sinusoidal signal. The receiver already knows in advance that it will be a sinus, so it is much simpler to generate one at the receiver directly rather than to transmit it over a channel! In other words, Shannon had the fundamental insight that we need to consider *random* messages rather than deterministic messages whenever we deal with information.

Let us go back to the example of the hats in Figure 5.1 and have a closer look at hat B.

- There is one chance out of four possibilities that we draw a white ball.

  Since we would like to use Hartley's measure here, we recall that the quantity r inside the logarithm in (5.1) is "the number of all possible outcomes of a random message." Hence, from Hartley's point of view, we will see one realization out of r possible realizations. Translated to the case of the white ball, we see that we have one realization out of four possible realizations, i.e.

  $$\log_2 4 = 2 \text{ bits} \tag{5.5}$$

  of information.
- On the other hand, there are three chances out of four that we draw a black ball.

  Here we cannot use Hartley's measure directly. But it is possible to translate the problem into a form that makes it somehow accessible to Hartley: we need to "normalize" the statement into a form that gives us *one* realization out of r. This can be done if we divide everything by 3, the number of black balls: we have one chance out of $4/3$ possibilities (whatever this means), or, stated differently, we have one realization out of $4/3$ possible "realizations," i.e.

  $$\log_2 \frac{4}{3} = 0.415 \text{ bits} \tag{5.6}$$

  of information.

[1] Besides the amazing accomplishment of inventing information theory, at the age of 21 Shannon also "invented" the computer in his *Master* thesis [Sha37]! He proved that electrical circuits can be used to perform logical and mathematical operations, which was the foundation of digital computer and digital circuit theory. It is probably the most important Master thesis of the twentieth century! Incredible, isn't it?

So now we have two different values depending on what color we get. How shall we combine them to one value that represents the information? The most obvious choice is to average it, i.e. we weight the different information values according to their probabilities of occurrence:

$$\frac{1}{4}\cdot 2 \text{ bits} + \frac{3}{4}\cdot 0.415 \text{ bits} = 0.811 \text{ bits} \tag{5.7}$$

or

$$\frac{1}{4}\log_2 4 + \frac{3}{4}\log_2 \frac{4}{3} = 0.811 \text{ bits}. \tag{5.8}$$

We see the following.

Shannon's measure of information is an "average Hartley information":

$$\sum_{i=1}^{r} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{r} p_i \log_2 p_i, \tag{5.9}$$

where p_i denotes the probability of the ith possible outcome.

We end this introductory section by pointing out that the given three motivating ideas, i.e.

(1) the number of possible answers r should be linked to "information";
(2) "information" should be additive in some sense; and
(3) a proper measure of information needs to take into account the probabilities of the various possible events,

are not sufficient to exclusively specify (5.9). The interested reader can find in Appendix 5.8 some more information on why Shannon's measure should be defined like (5.9) and not differently.

5.2 Uncertainty or entropy

5.2.1 Definition

We now formally define the Shannon measure of "self-information of a source." Due to its relationship with a corresponding concept in different areas of physics, Shannon called his measure *entropy*. We will stick to this name as it is standard in the whole literature. However, note that *uncertainty* would be a far more precise description.

Definition 5.4 (Entropy) The *uncertainty* or *entropy* of a random message U that takes on r different values with probability p_i, $i = 1, \ldots, r$, is defined as

$$H(U) \triangleq -\sum_{i=1}^{r} p_i \log_b p_i. \tag{5.10}$$

Remark 5.5 What happens if $p_i = 0$? Remember that $\log_b 0 = -\infty$. However, also note that $p_i = 0$ means that the symbol i never shows up. It therefore should not contribute to the uncertainty. Luckily this is the case:

$$\lim_{t \to 0} t \log_b t = 0, \tag{5.11}$$

i.e. we do not need to worry about this case.

So we note the following.

> Whenever we sum over $p_i \log_b p_i$, we implicitly assume that we exclude all indices i with $p_i = 0$.

As in the case of the Hartley measure of information, b denotes the *unit* of uncertainty:

$$b = 2: \quad \text{bit}, \tag{5.12}$$

$$b = e: \quad \text{nat}, \tag{5.13}$$

$$b = 10: \quad \text{Hartley}. \tag{5.14}$$

If the base of the logarithm is not specified, then we can choose it freely. However, note that the units are very important. A statement "$H(U) = 0.43$" is completely meaningless: since

$$\log_b \xi = \frac{\log_2 \xi}{\log_2 b}, \tag{5.15}$$

0.43 could mean anything as, e.g.,

$$\text{if } b = 2: \qquad H(U) = 0.43 \text{ bits}, \tag{5.16}$$

$$\text{if } b = e: \qquad H(U) = 0.43 \text{ nats} \simeq 0.620 \text{ bits}, \tag{5.17}$$

$$\text{if } b = 256 = 2^8: \quad H(U) = 0.43 \text{ "bytes"} = 3.44 \text{ bits}. \tag{5.18}$$

Note that the term *bits* is used in two ways: its first meaning is the *unit* of entropy when the base of the logarithm is chosen to be 2; its second meaning is *binary digits*, i.e. in particular the number of digits of a binary codeword.

Remark 5.6 It is worth mentioning that if all r events are equally likely, Shannon's definition of entropy reduces to Hartley's measure:

$$p_i = \frac{1}{r},\ \forall i: \quad H(U) = -\sum_{i=1}^{r} \frac{1}{r} \log_b \frac{1}{r} = \frac{1}{r} \log_b r \cdot \underbrace{\sum_{i=1}^{r} 1}_{=r} = \log_b r. \tag{5.19}$$

Remark 5.7 Be careful not to confuse *uncertainty* with *information*. For motivation purposes, in Section 5.1 we talked a lot about "information." However, what we actually meant there is "self-information" or, more nicely put, "uncertainty." You will learn in Chapter 6 that information is what you get by reducing uncertainty and see a formal definition of information there.

Another important observation is that the entropy of U does not depend on the different possible values that U can take on, but only on the *probabilities* of these values. Hence,

$$U \in \Big\{ \underbrace{1}_{\substack{\text{with} \\ \text{prob. } \frac{1}{2}}}, \underbrace{2}_{\substack{\text{with} \\ \text{prob. } \frac{1}{3}}}, \underbrace{3}_{\substack{\text{with} \\ \text{prob. } \frac{1}{6}}} \Big\} \tag{5.20}$$

and

$$V \in \Big\{ \underbrace{34}_{\substack{\text{with} \\ \text{prob. } \frac{1}{2}}}, \underbrace{512}_{\substack{\text{with} \\ \text{prob. } \frac{1}{3}}}, \underbrace{981}_{\substack{\text{with} \\ \text{prob. } \frac{1}{6}}} \Big\} \tag{5.21}$$

have both the same entropy, which is

$$H(U) = H(V) = -\frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{3}\log_2 \frac{1}{3} - \frac{1}{6}\log_2 \frac{1}{6} \simeq 1.46 \text{ bits}. \tag{5.22}$$

5.2.2 Binary entropy function

One special case of entropy is so important that we introduce a specific name.

Definition 5.8 (Binary entropy function) If U is *binary* with two possible values u_1 and u_2 such that $\Pr[U = u_1] = p$ and $\Pr[U = u_2] = 1 - p$, then

$$H(U) = H_{\mathrm{b}}(p), \tag{5.23}$$

where $H_{\mathrm{b}}(\cdot)$ is called the *binary entropy function* and is defined as

$$H_{\mathrm{b}}(p) \triangleq -p \log_2 p - (1-p)\log_2(1-p), \qquad p \in [0,1]. \tag{5.24}$$

The function $H_{\mathrm{b}}(\cdot)$ is shown in Figure 5.2.

Exercise 5.9 *Show that the maximal value of $H_{\mathrm{b}}(p)$ is 1 bit and is taken on for $p = 1/2$.* ◊

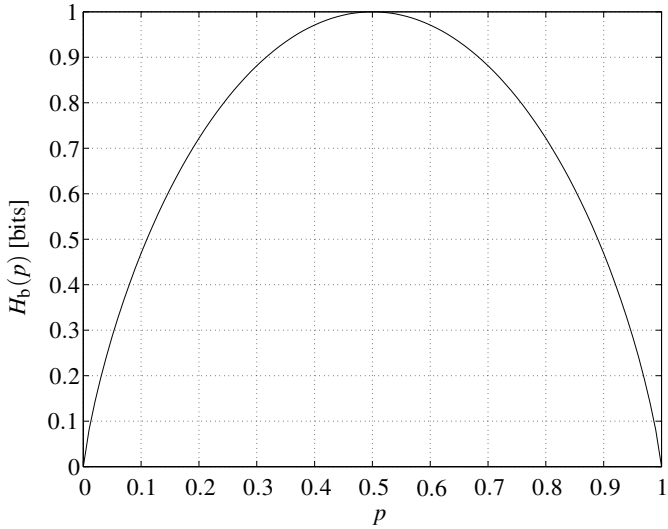

Figure 5.2 Binary entropy function $H_b(p)$ as a function of the probability p.

5.2.3 The Information Theory Inequality

The following inequality does not really have a name, but since it is so important in information theory, we will follow James Massey, retired professor at ETH in Zurich, and call it the *Information Theory Inequality* or the *IT Inequality*.

Lemma 5.10 (IT Inequality) *For any base $b > 0$ and any $\xi > 0$,*

$$\left(1 - \frac{1}{\xi}\right) \log_b e \leq \log_b \xi \leq (\xi - 1) \log_b e \tag{5.25}$$

with equalities on both sides if, and only if, $\xi = 1$.

Proof Actually, Figure 5.3 can be regarded as a proof. For those readers who would like a formal proof, we provide a mathematical derivation. We start with the upper bound. First note that

$$\log_b \xi \big|_{\xi=1} = 0 = (\xi - 1) \log_b e \big|_{\xi=1}. \tag{5.26}$$

Then have a look at the derivatives:

$$\frac{\mathrm{d}}{\mathrm{d}\xi} \log_b \xi = \frac{1}{\xi} \log_b e \begin{cases} > \log_b e & \text{if } 0 < \xi < 1, \\ < \log_b e & \text{if } \xi > 1, \end{cases} \tag{5.27}$$

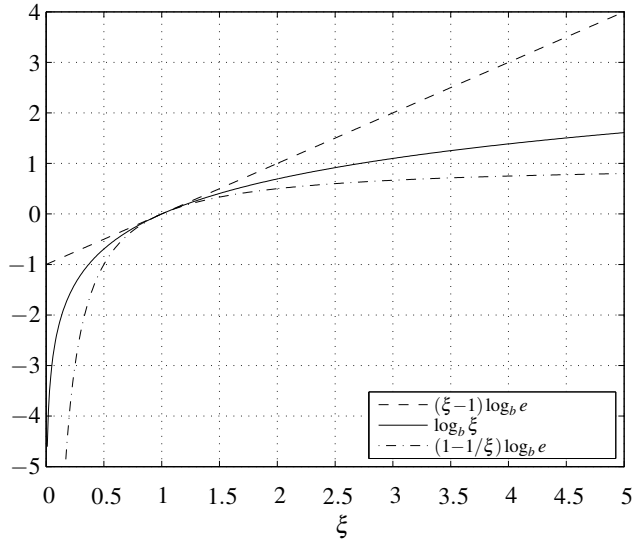

Figure 5.3 Illustration of the IT Inequality.

and

$$\frac{\mathrm{d}}{\mathrm{d}\xi}(\xi-1)\log_b e = \log_b e. \tag{5.28}$$

Hence, the two functions coincide at $\xi = 1$, and the linear function is above the logarithm for all other values.

To prove the lower bound again note that

$$\left(1-\frac{1}{\xi}\right)\log_b e\Big|_{\xi=1} = 0 = \log_b \xi\big|_{\xi=1} \tag{5.29}$$

and

$$\frac{\mathrm{d}}{\mathrm{d}\xi}\left(1-\frac{1}{\xi}\right)\log_b e = \frac{1}{\xi^2}\log_b e \begin{cases} > \frac{\mathrm{d}}{\mathrm{d}\xi}\log_b \xi = \frac{1}{\xi}\log_b e & \text{if } 0<\xi<1, \\ < \frac{\mathrm{d}}{\mathrm{d}\xi}\log_b \xi = \frac{1}{\xi}\log_b e & \text{if } \xi>1, \end{cases} \tag{5.30}$$

similarly to above. □

5.2.4 Bounds on the entropy

Lemma 5.11 *If U has r possible values, then*

$$0 \le H(U) \le \log_2 r \text{ bits}, \tag{5.31}$$

where

$$H(U) = 0 \qquad \text{if, and only if, } p_i = 1 \text{ for some } i, \tag{5.32}$$

$$H(U) = \log_2 r \text{ bits} \qquad \text{if, and only if, } p_i = \frac{1}{r}\ \forall i. \tag{5.33}$$

Proof Since $0 \le p_i \le 1$, we have

$$-p_i \log_2 p_i \begin{cases} = 0 & \text{if } p_i = 1, \\ > 0 & \text{if } 0 < p_i < 1. \end{cases} \tag{5.34}$$

Hence, $H(U) \ge 0$. Equality can only be achieved if $-p_i \log_2 p_i = 0$ for all i, i.e. $p_i = 1$ for one i and $p_i = 0$ for the rest.

To derive the upper bound, we use a trick that is quite common in information theory: we take the difference and try to show that it must be nonpositive. In the following we arrange the probabilities in descending order and assume that r' ($r' \le r$) of the r values of the probabilities p_i are strictly positive, i.e. $p_i > 0$ for all $i = 1, \ldots, r'$, and $p_i = 0$ for $i = r'+1, \ldots, r$. Then

$$H(U) - \log_2 r = -\sum_{i=1}^{r} p_i \log_2 p_i - \log_2 r \tag{5.35}$$

$$= -\sum_{i=1}^{r'} p_i \log_2 p_i - \log_2 r \cdot \underbrace{\sum_{i=1}^{r'} p_i}_{=1} \tag{5.36}$$

$$= -\sum_{i=1}^{r'} p_i \log_2 p_i - \sum_{i=1}^{r'} p_i \log_2 r \tag{5.37}$$

$$= -\sum_{i=1}^{r'} p_i \log_2 (p_i \cdot r) \tag{5.38}$$

$$= \sum_{i=1}^{r'} p_i \log_2 \underbrace{\left(\frac{1}{p_i \cdot r}\right)}_{\triangleq \xi} \tag{5.39}$$

$$\le \sum_{i=1}^{r'} p_i \left(\frac{1}{p_i \cdot r} - 1\right) \cdot \log_2 e \tag{5.40}$$

$$= \left(\sum_{i=1}^{r'} \frac{1}{r} - \underbrace{\sum_{i=1}^{r'} p_i}_{=1}\right) \cdot \log_2 e \tag{5.41}$$

$$= \left(\frac{r'}{r} - 1\right) \cdot \log_2 e \tag{5.42}$$

$$\le (1-1) \cdot \log_2 e = 0. \tag{5.43}$$

Here, (5.40) follows from the IT Inequality (Lemma 5.10), and (5.43) follows because $r' \le r$. Hence, $H(U) \le \log_2 r$.

Equality can only be achieved if both

(1) in the IT Inequality $\xi = 1$, i.e. if $1/p_i r = 1$ for all i, i.e. if $p_i = 1/r$ for all i; and
(2) $r' = r$.

Note that if the first condition is satisfied, then the second condition is automatically satisfied. □

5.3 Trees revisited

The most elegant way to connect our new definition of entropy with the codes introduced in Chapter 4 is to rely again on trees with probabilities.

Consider a binary tree with probabilities. We remind the reader of our notation:

- n denotes the total number of leaves;
- p_i, $i = 1, \ldots, n$, denote the probabilities of the leaves;
- N denotes the number of nodes (including the root, but excluding the leaves); and
- P_ℓ, $\ell = 1, \ldots, \mathsf{N}$, denote the probabilities of the nodes, where by definition $P_1 = 1$ is the root probability.

Moreover, we will use $q_{\ell,j}$ to denote the probability of the jth node/leaf that is one step forward from node ℓ (the jth child of node ℓ), where $j = 0, 1$. That is, we have

$$q_{\ell,0} + q_{\ell,1} = P_\ell. \tag{5.44}$$

Now we give the following definitions.

Definition 5.12 The *leaf entropy* is defined as

$$H_{\text{leaf}} \triangleq -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{5.45}$$

Definition 5.13 Denoting by $P_1, P_2, \ldots, P_{\mathsf{N}}$ the probabilities of all nodes (including the root) and by $q_{\ell,j}$ the probability of the nodes and leaves one step forward from node ℓ, we define the *branching entropy* H_ℓ of node ℓ as

$$H_\ell \triangleq -\frac{q_{\ell,0}}{P_\ell} \log_2 \frac{q_{\ell,0}}{P_\ell} - \frac{q_{\ell,1}}{P_\ell} \log_2 \frac{q_{\ell,1}}{P_\ell}. \tag{5.46}$$

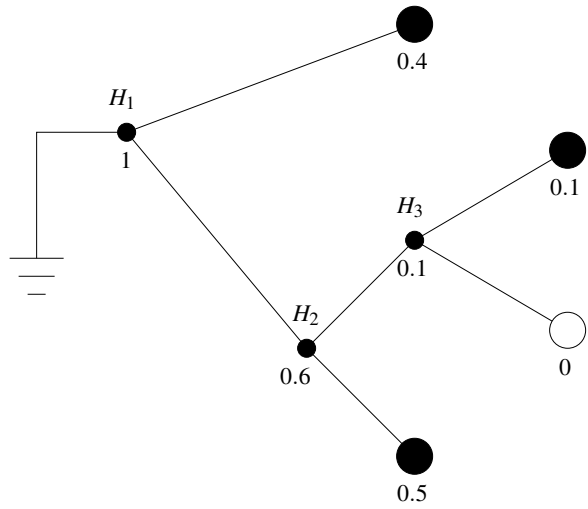

Figure 5.4 An example of a binary tree with probabilities to illustrate the calculations of the leaf entropy and the branching entropies.

Note that following Remark 5.5 we implicitly assume that the sum is only over those j for which $q_{\ell,j} > 0$, i.e. we have $H_\ell = 0$ if one of the $q_{\ell,j}$ is zero. Note further that $q_{\ell,j}/P_\ell$ is the conditional probability of going along the jth branch given that we are at node ℓ (normalization!).

Example 5.14 As an example consider the tree in Figure 5.4. We have

$$H_{\text{leaf}} = -0.4\log_2 0.4 - 0.1\log_2 0.1 - 0.5\log_2 0.5 \simeq 1.361 \text{ bits}; \quad (5.47)$$

$$H_1 = -\frac{0.4}{1}\log_2\frac{0.4}{1} - \frac{0.6}{1}\log_2\frac{0.6}{1} \simeq 0.971 \text{ bits}; \quad (5.48)$$

$$H_2 = -\frac{0.1}{0.6}\log_2\frac{0.1}{0.6} - \frac{0.5}{0.6}\log_2\frac{0.5}{0.6} \simeq 0.650 \text{ bits}; \quad (5.49)$$

$$H_3 = -\frac{0.1}{0.1}\log_2\frac{0.1}{0.1} = 0 \text{ bits}. \quad (5.50)$$

◊

We will next prove a very interesting relationship between the leaf entropy and the branching entropy that will turn out to be fundamental for the understanding of codes.

Theorem 5.15 (Leaf Entropy Theorem) *In any tree with probabilities we have that*

$$H_{\text{leaf}} = \sum_{\ell=1}^{\mathrm{N}} P_\ell H_\ell. \quad (5.51)$$

Proof Recall that by the definition of trees and trees with probabilities we have, for every node ℓ,

$$P_\ell = q_{\ell,0} + q_{\ell,1}. \tag{5.52}$$

Using the definition of branching entropy, we obtain

$$\begin{aligned} P_\ell H_\ell &= P_\ell \cdot \left(-\frac{q_{\ell,0}}{P_\ell} \log_2 \frac{q_{\ell,0}}{P_\ell} - \frac{q_{\ell,1}}{P_\ell} \log_2 \frac{q_{\ell,1}}{P_\ell} \right) && (5.53)\\ &= -q_{\ell,0} \log_2 \frac{q_{\ell,0}}{P_\ell} - q_{\ell,1} \log_2 \frac{q_{\ell,1}}{P_\ell} && (5.54)\\ &= -q_{\ell,0} \log_2 q_{\ell,0} - q_{\ell,1} \log_2 q_{\ell,1} + q_{\ell,0} \log_2 P_\ell + q_{\ell,1} \log_2 P_\ell && (5.55)\\ &= -q_{\ell,0} \log_2 q_{\ell,0} - q_{\ell,1} \log_2 q_{\ell,1} + \underbrace{\left(q_{\ell,0} + q_{\ell,1}\right)}_{=P_\ell} \log_2 P_\ell && (5.56)\\ &= -q_{\ell,0} \log_2 q_{\ell,0} - q_{\ell,1} \log_2 q_{\ell,1} + P_\ell \log_2 P_\ell, && (5.57) \end{aligned}$$

where the last equality follows from (5.52).

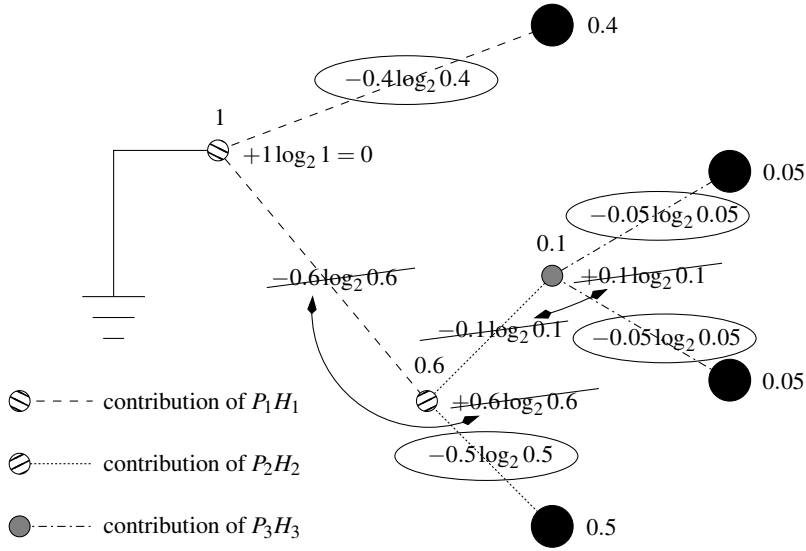

Figure 5.5 Graphical proof of the Leaf Entropy Theorem. There are three nodes: we see that all contributions cancel apart from the root node (whose contribution is 0) and the leaves.

Hence, for every node $\tilde{\ell}$, we see that it will contribute to $\sum_{\ell=1}^{N} P_\ell H_\ell$ twice:

- firstly it will add $P_{\tilde{\ell}} \log_2 P_{\tilde{\ell}}$ when the node counter ℓ passes through $\tilde{\ell}$; but
- secondly it will subtract $q_{\ell,j} \log_2 q_{\ell,j} = P_{\tilde{\ell}} \log_2 P_{\tilde{\ell}}$ when the node counter ℓ points to the parent node of $\tilde{\ell}$.

Hence, the contributions of all nodes will be canceled out – apart from the root that does not have a parent! The root only contributes $P_1 \log_2 P_1$ for $\ell = 1$. However, since $P_1 = 1$, we have $P_1 \log_2 P_1 = 1 \log_2 1 = 0$. So the root does not contribute either.

It only remains to consider the leaves. Note that the node counter ℓ will not pass through leaves by definition. Hence, a leaf only contributes when the node counter points to its parent node and its contribution is $-q_{\ell,j} \log_2 q_{\ell,j} = -p_i \log_2 p_i$. Since the sum of all $-p_i \log_2 p_i$ equals the leaf entropy by definition, this proves the claim.

In Figure 5.5 we have tried to depict this proof graphically. □

Example 5.16 (Continuation from Example 5.14) Using the values from (5.47)–(5.50) we obtain

$$P_1 H_1 + P_2 H_2 + P_3 H_3 = 1 \cdot 0.971 + 0.6 \cdot 0.650 + 0.1 \cdot 0 \text{ bits} \quad (5.58)$$

$$= 1.361 \text{ bits} = H_{\text{leaf}} \quad (5.59)$$

as expected. ◊

5.4 Bounds on the efficiency of codes

The main strength of information theory is that it can provide some fundamental statements about what is possible and what is not possible to achieve. So a typical information theoretic result will consist of an upper bound and a lower bound or, in general, an achievability part and a converse part. The achievability part of a theorem tells us what we can do, and the converse part tells us what we cannot do.

Sometimes, the theorem will also tell us how to do it, but usually the result is theoretic in the sense that it only proves what is possible without actually saying how it could be done. To put it pointedly: information theory tells us what is possible; coding theory tells us how to do it.

5.4.1 What we cannot do: fundamental limitations of source coding

Let us quickly summarize what we know about codes and their corresponding trees.

- The most efficient codes can always be chosen to be prefix-free (Theorem 4.19).

- Every prefix-free code can be represented by a tree where every codeword corresponds to a leaf in the tree.
- Every codeword has a certain probability corresponding to the probability of the symbol it represents.
- Unused leaves can be regarded as symbols that never occur, i.e. we assign probability zero to them.

Hence, from these observations we immediately see that the entropy $H(U)$ of a random message U with probabilities $p_1, \ldots, p_r$ and the leaf entropy of the corresponding tree are the same:

$$H_{\text{leaf}} = H(U). \tag{5.60}$$

Note that the unused leaves do not contribute to H_{leaf} since they have zero probability.

Moreover, the average codeword length L_{av} is equivalent to the average depth of the leaves. (Again we can ignore the unused leaves, since they have probability zero and therefore do not contribute to the average.)

Now note that since we consider binary trees where each node branches into two different children, we know from Lemma 5.11 that the branching entropy can be upper-bounded as follows:

$$H_\ell \le \log_2 2 = 1 \text{ bit}. \tag{5.61}$$

Hence, using this together with the Leaf Entropy Theorem (Theorem 5.15) and the Path Length Lemma (Lemma 4.11) we obtain the following:

$$H(U) = H_{\text{leaf}} = \sum_{\ell=1}^{\mathrm{N}} P_\ell H_\ell \le \sum_{\ell=1}^{\mathrm{N}} P_\ell \cdot 1 \text{ bit} = \sum_{\ell=1}^{\mathrm{N}} P_\ell = \mathrm{L}_{\text{av}} \text{ bits}. \tag{5.62}$$

In other words,

$$\mathrm{L}_{\text{av}} \ge H(U) \text{ bits}. \tag{5.63}$$

This is the *converse part* of the *Coding Theorem for a Single Random Message*. It says that whatever code you try to design, the average codeword length of *any* binary code for an r-ary random message U cannot be smaller than the entropy of U (using the correct unit of bits)!

Note that to prove this statement we have not designed any code, but instead we have been able to prove something that holds for *every* code that exists!

When do we have equality? From the above derivation we see that we have equality if the branching entropy is always 1 bit, $H_\ell = 1$ bit, i.e. the branching probabilities are all uniform. This is only possible if p_i is a negative integer power of 2 for all i:

$$p_i = 2^{-v_i} \tag{5.64}$$

with v_i a natural number (and, of course, if we design an optimal code).

5.4.2 What we can do: analysis of the best codes

In practice, it is not only important to know where the limitations are, but also perhaps even more so to know how close we can get to these limitations. So as a next step we would like to analyze the best codes (i.e. the Huffman codes derived in Section 4.6) and see how close they get to the limitations shown in Section 5.4.1.

Unfortunately, it is rather difficult to analyze Huffman codes. To circumvent this problem, we will design a new code, called the *Fano code*, and analyze its performance instead. Fano codes are not optimal in general, i.e. their performance is worse than the performance of Huffman codes. Therefore any upper bound on L_{av} that can be achieved by a Fano code can definitely also be achieved by a Huffman code.

Definition 5.17 (Fano code) The *Fano code*[2] is generated according to the following algorithm:

Step 1 Arrange the symbols in order of nonincreasing probability.

Step 2 Divide the list of ordered symbols into two parts, with the total probability of the left part being as close to the total probability of the right part as possible.

Step 3 Assign the binary digit 0 to the left part of the list, and the digit 1 to the right part. This means that the codewords for the symbols in the first part will all start with 0, and the codewords for the symbols in the second part will all start with 1.

Step 4 Recursively apply Step 2 and Step 3 to each of the two parts, subdividing into further parts and adding bits to the codewords until each symbol is the single member of a part.

Note that effectively this algorithm constructs a tree. Hence, the Fano code is prefix-free.

Example 5.18 Let us generate the Fano code for a random message with five symbols having probabilities

$$\begin{aligned} &p_1 = 0.35, \qquad p_2 = 0.25, \qquad p_3 = 0.15, \\ &p_4 = 0.15, \qquad p_5 = 0.1. \end{aligned} \tag{5.65}$$

[2] Note that this code is usually known as the *Shannon–Fano code*. However, this is a misnaming because it was Fano's invention. Shannon proposed a slightly different code, which unfortunately is also known as the *Shannon–Fano code*. For more details on this confusion, we refer to the discussion in Section 5.6.

Since the symbols are already ordered in decreasing order of probability, Step 1 can be omitted. We hence want to split the list into two parts, both having as similar total probability as possible. If we split $\{1\}$ and $\{2,3,4,5\}$, we have a total probability 0.35 on the left and 0.65 on the right; the split $\{1,2\}$ and $\{3,4,5\}$ yields 0.6 and 0.4; and $\{1,2,3\}$ and $\{4,5\}$ gives 0.75 and 0.25. We see that the second split is best. So we assign 0 as a first digit to $\{1,2\}$ and 1 to $\{3,4,5\}$.

Now we repeat the procedure with both subgroups. Firstly, we split $\{1,2\}$ into $\{1\}$ and $\{2\}$. This is trivial. Secondly, we split $\{3,4,5\}$ into $\{3\}$ and $\{4,5\}$ because 0.15 and 0.25 is closer than 0.3 and 0.1 that we would have obtained by dividing into $\{3,4\}$ and $\{5\}$. Again we assign the corresponding second digits.

Finally, we split the last group $\{4,5\}$ into $\{4\}$ and $\{5\}$. We end up with the five codewords $\{00,01,10,110,111\}$. This whole procedure is shown in Figure 5.6. ◊

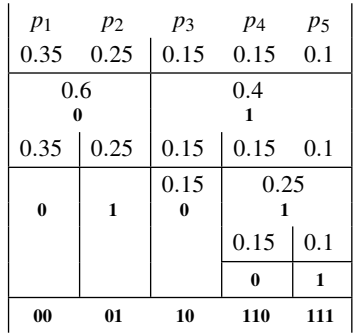

Figure 5.6 Construction of the Fano code of Example 5.18.

Exercise 5.19 *Construct the Fano code for the random message U of Example 4.16 with four symbols having probabilities*

$$p_1 = 0.4, \qquad p_2 = 0.3, \qquad p_3 = 0.2, \qquad p_4 = 0.1, \tag{5.66}$$

and show that it is identical to the corresponding Huffman code. ◊

Remark 5.20 We would like to point out that there are cases where the algorithm given in Definition 5.17 does not lead to a unique design: there might be two different ways of dividing the list into two parts such that the total probabilities are as similar as possible. Since the algorithm does not specify what to

do in such a case, you are free to choose any possible way. Unfortunately, however, these different choices can lead to codes with different performance.[3] As an example, consider a random message U with seven possible symbols having the following probabilities:

$$\begin{aligned} &p_1 = 0.35, \quad p_2 = 0.3, \quad p_3 = 0.15, \quad p_4 = 0.05, \\ &p_5 = 0.05, \quad p_6 = 0.05, \quad p_7 = 0.05. \end{aligned} \tag{5.67}$$

Figures 5.7 and 5.8 show two different possible Fano codes for this random message. The first has an average codeword length of $\mathrm{L_{av}} = 2.45$, while the latter's performance is better with $\mathrm{L_{av}} = 2.4$.

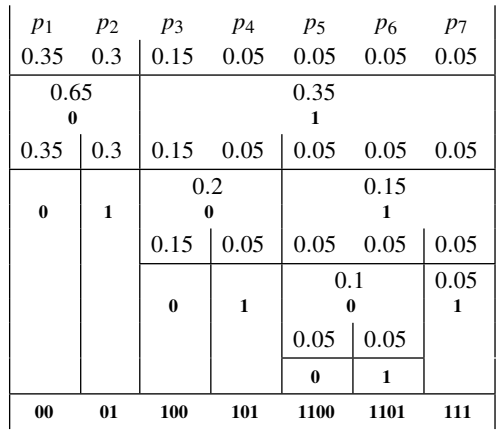

Figure 5.7 One possible Fano code for the random message given in (5.67).

Exercise 5.21 *In total there are six different possible designs of a Fano code for the random message given in Remark 5.20. Design all of them and compare their performances.* ◊

We next prove a simple property of the Fano code.

Lemma 5.22 *The codeword lengths l_i of a Fano code satisfy the following:*

$$l_i \le \left\lceil \log_2 \frac{1}{p_i} \right\rceil, \tag{5.68}$$

where $\lceil \xi \rceil$ denotes the smallest integer not smaller than ξ.

[3] This cannot happen in the case of a Huffman code! Even though the algorithm of the Huffman code is not unique either, it always will result in codes of equal (optimal) performance. The reason for this is clear: we have proven that the Huffman algorithm results in an *optimal* code.

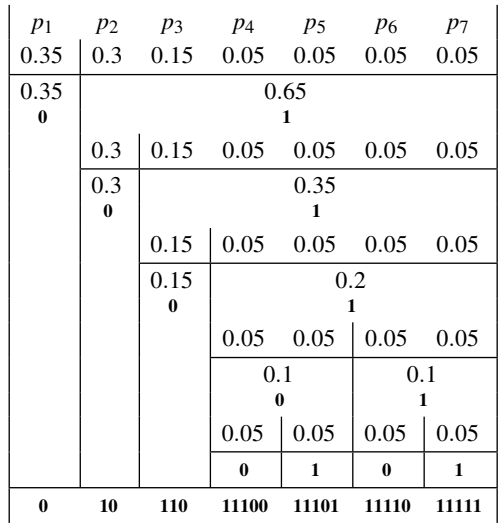

Figure 5.8 A second possible Fano code for the random message given in (5.67).

Proof By construction, any symbol with probability $p_i \geq 1/2$ will be alone in one part in the first round of the algorithm. Hence,

$$l_i = 1 = \left\lceil \log_2 \frac{1}{p_i} \right\rceil. \tag{5.69}$$

If $1/4 \leq p_i < 1/2$, then at the latest in the second round of the algorithm the symbol will occupy one partition. (Note that it is possible that the symbol is already the single element of one partition in the first round. For example, for $p_1 = 3/4$ and $p_2 = 1/4$, p_2 will have $l_2 = 1$.) Hence, we have

$$l_i \leq 2 = \left\lceil \log_2 \frac{1}{p_i} \right\rceil. \tag{5.70}$$

In the same fashion we show that for $1/8 \leq p_i < 1/4$,

$$l_i \leq 3 = \left\lceil \log_2 \frac{1}{p_i} \right\rceil; \tag{5.71}$$

for $1/16 \leq p_i < 1/8$,

$$l_i \leq 4 = \left\lceil \log_2 \frac{1}{p_i} \right\rceil; \tag{5.72}$$

etc. □

Next, let us see how efficient the Fano code is. To that goal, we note that from (5.68) we have

$$l_i \leq \left\lceil \log_2 \frac{1}{p_i} \right\rceil < \log_2 \frac{1}{p_i} + 1. \tag{5.73}$$

We get

$$\begin{aligned} \mathrm{L}_{\mathrm{av}} &= \sum_{i=1}^{r} p_i l_i && (5.74)\\ &< \sum_{i=1}^{r} p_i \left(\log_2 \frac{1}{p_i} + 1 \right) && (5.75)\\ &= \sum_{i=1}^{r} p_i \log_2 \frac{1}{p_i} + \sum_{i=1}^{r} p_i && (5.76)\\ &= -\sum_{i=1}^{r} p_i \log_2 p_i + 1 && (5.77)\\ &= H(U) + 1 \text{ bits}, && (5.78) \end{aligned}$$

where the entropy is based on the binary logarithm, i.e. it is measured in bits. Hence, the Fano code (even though it is *not* an optimal code) approaches the ultimate lower bound (5.63) by less than 1 bit! A Huffman code will be even better than that.

5.4.3 Coding Theorem for a Single Random Message

We summarize this so far most important result of this chapter.

Theorem 5.23 (Coding Theorem for a Single Random Message) *For an optimal binary prefix-free code (i.e. a binary Huffman code) for an r-ary random message U, the average codeword length L_{av} satisfies*

$$H(U) \text{ bits} \leq \mathrm{L}_{\mathrm{av}} < H(U) + 1 \text{ bits} \tag{5.79}$$

(where the entropy is measured in bits). We have equality on the left if, and only if, p_i is a negative integer power of 2, $\forall\, i$.
Moreover, this statement also holds true for Fano coding.

Example 5.24 We consider a random message U with seven symbols having probabilities

$$\begin{aligned} &p_1 = 0.4, \quad p_2 = 0.1, \quad p_3 = 0.1, \quad p_4 = 0.1,\\ &p_5 = 0.1, \quad p_6 = 0.1, \quad p_7 = 0.1, \end{aligned} \tag{5.80}$$

i.e. $H(U) \simeq 2.52$ bits. We firstly design a Fano code; see Figure 5.9. The corresponding tree is shown in Figure 5.10. Note that the construction algorithm is not unique in this case: in the second round we could split the second group either to $\{3,4\}$ and $\{5,6,7\}$ or $\{3,4,5\}$ and $\{6,7\}$. In this case, both ways will result in a code of identical performance. The same situation occurs in the third round.

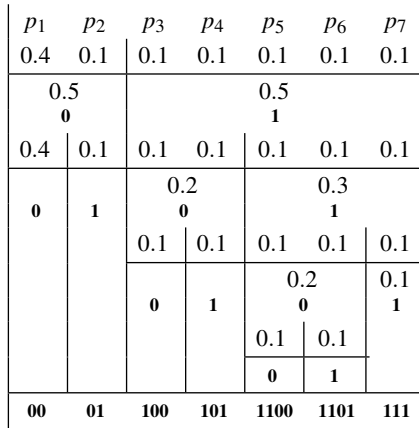

Figure 5.9 Construction of the Fano code of Example 5.24.

The efficiency of this Fano code is given by

$$\mathrm{L_{av}} = 1 + 0.5 + 0.5 + 0.2 + 0.3 + 0.2 = 2.7 \text{ bits}, \tag{5.81}$$

which satisfies, as predicted,

$$2.52 \text{ bits} \leq 2.7 \text{ bits} < 3.52 \text{ bits}. \tag{5.82}$$

A corresponding Huffman code for U is shown in Figure 5.11. Its performance is $\mathrm{L_{av}} = 2.6$ bits, i.e. it is better than the Fano code, but of course it still holds that

$$2.52 \text{ bits} \leq 2.6 \text{ bits} < 3.52 \text{ bits}. \tag{5.83}$$

◊

Exercise 5.25 *Design a Huffman code and a Fano code for the random message U with probabilities*

$$\begin{aligned} &p_1 = 0.25, \quad p_2 = 0.2, \quad p_3 = 0.2, \quad p_4 = 0.1, \\ &p_5 = 0.1, \quad p_6 = 0.1, \quad p_7 = 0.05, \end{aligned} \tag{5.84}$$

and compare their performances. ◊

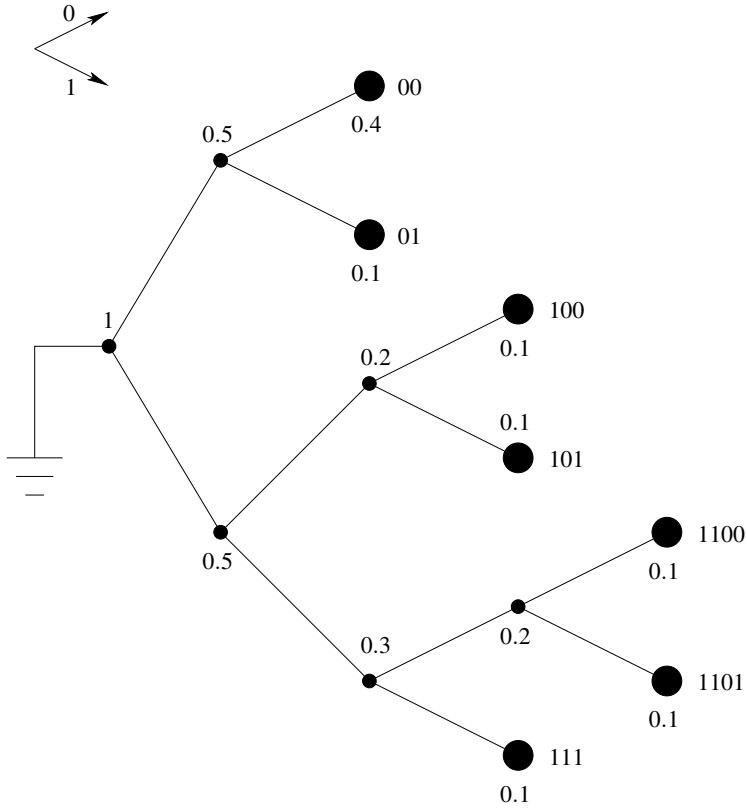

Figure 5.10 A Fano code for the message U of Example 5.24.

We have seen in the above examples and exercises that, even though the Fano code is not optimal, its performance is usually very similar to the optimal Huffman code. In particular, we know from Theorem 5.23 that the performance gap is less than one binary digit.

We are going to see next that once we start encoding not a single random message, but a sequence of such messages emitted by a random source, this difference becomes negligible.

5.5 Coding of an information source

So far we have only considered a single random message, but in reality we are much more likely to encounter a situation where we have a *stream* of messages

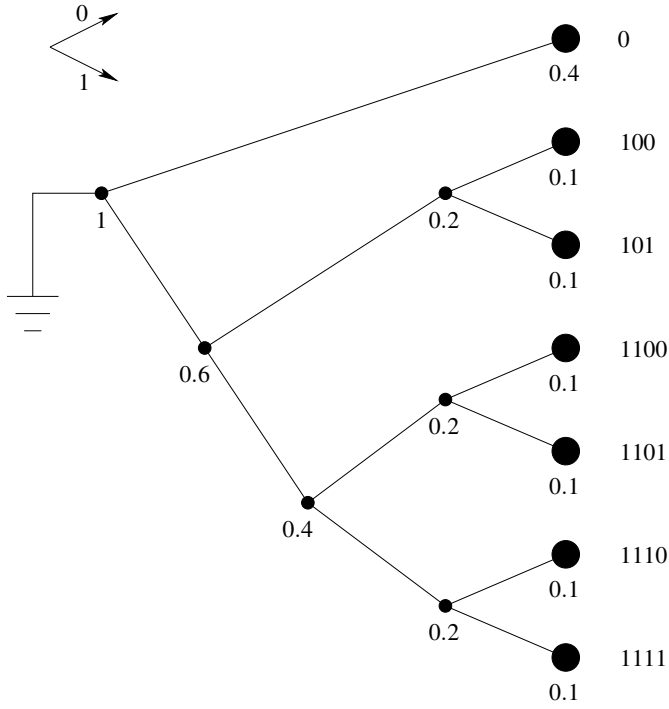

Figure 5.11 A Huffman code for the message U of Example 5.24.

that should be encoded continuously. Luckily we have prepared ourselves for this situation already by considering prefix-free codes only, which make sure that a sequence of codewords can be separated easily into the individual codewords.

In the following we will consider only the simplest case where the random source is memoryless, i.e. each symbol that is emitted by the source is independent of all past symbols. A formal definition is given as follows.

Definition 5.26 (DMS) An r-ary *discrete memoryless source (DMS)* is a device whose output is a sequence of random messages $U_1, U_2, U_3, \ldots$, where

- each U_ℓ can take on r different values with probability $p_1, \ldots, p_r$, and
- the different messages U_ℓ are independent of each other.

The obvious way of designing a compression system for such a source is to design a Huffman code for U, continuously use it for each message U_ℓ, and concatenate the codewords together. The receiver can easily separate the code-

words (because the Huffman code is prefix-free) and decode them to recover the sequence of messages $\{U_\ell\}$.

However, the question is whether this is the most efficient approach. Note that it is also possible to combine two or more messages

$$(U_\ell, U_{\ell+1}, \ldots, U_{\ell+\nu})$$

together and design a Huffman code for these combined messages! We will show below that this latter approach is actually more efficient. But before doing so, we need to think about such *random vector messages*.

Remark 5.27 Note that a random vector message $\mathbf{V} = (U_1, \ldots, U_\nu)$ is, from the mathematical point of view, no different from any other random message: it takes on a certain finite number of different values with certain probabilities. If U_ℓ is r-ary, then $\mathbf{V}$ is r^ν-ary, but otherwise there is no fundamental difference.

We can even express the entropy of $\mathbf{V}$ as a function of the entropy of U. Let q_j denote the probability of the jth symbol of $\mathbf{V}$. Since the different messages U_ℓ are independent, we have

$$q_j = p_{i_1} \cdot p_{i_2} \cdots p_{i_\nu}, \tag{5.85}$$

where p_{i_ℓ} denotes the probability of the i_ℓth symbol of U_ℓ. Hence,

$$\begin{aligned}
H(\mathbf{V}) &= -\sum_{j=1}^{r^\nu} q_j \log_2 q_j && (5.86)\\
&= -\sum_{i_1=1}^{r} \cdots \sum_{i_\nu=1}^{r} (p_{i_1} \cdot p_{i_2} \cdots p_{i_\nu}) \log_2 (p_{i_1} \cdot p_{i_2} \cdots p_{i_\nu}) && (5.87)\\
&= -\sum_{i_1=1}^{r} \cdots \sum_{i_\nu=1}^{r} (p_{i_1} \cdot p_{i_2} \cdots p_{i_\nu}) \big(\log_2 p_{i_1} + \cdots + \log_2 p_{i_\nu}\big) && (5.88)\\
&= -\sum_{i_1=1}^{r} \cdots \sum_{i_\nu=1}^{r} p_{i_1} \cdot p_{i_2} \cdots p_{i_\nu} \cdot \log_2 p_{i_1}\\
&\quad - \cdots - \sum_{i_1=1}^{r} \cdots \sum_{i_\nu=1}^{r} p_{i_1} \cdot p_{i_2} \cdots p_{i_\nu} \cdot \log_2 p_{i_\nu} && (5.89)\\
&= -\left(\sum_{i_1=1}^{r} p_{i_1} \log_2 p_{i_1}\right) \cdot \underbrace{\left(\sum_{i_2=1}^{r} p_{i_2}\right)}_{=1} \cdots \underbrace{\left(\sum_{i_\nu=1}^{r} p_{i_\nu}\right)}_{=1}\\
&\quad - \cdots - \underbrace{\left(\sum_{i_1=1}^{r} p_{i_1}\right)}_{=1} \cdots \underbrace{\left(\sum_{i_{\nu-1}=1}^{r} p_{i_{\nu-1}}\right)}_{=1} \cdot \left(\sum_{i_\nu=1}^{r} p_{i_\nu} \log_2 p_{i_\nu}\right) && (5.90)
\end{aligned}$$

$$= -\left(\sum_{i_1=1}^{r} p_{i_1} \log_2 p_{i_1}\right) - \cdots - \left(\sum_{i_\nu=1}^{r} p_{i_\nu} \log_2 p_{i_\nu}\right) \tag{5.91}$$

$$= H(U_1) + \cdots + H(U_\nu) \tag{5.92}$$

$$= \nu H(U). \tag{5.93}$$

Here the last equality follows because the entropy of all U_ℓ is identical.

In other words, since $\mathbf{V}$ consists of ν *independent* random messages U, its uncertainty is simply ν times the uncertainty of U.

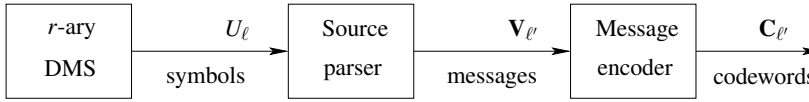

Figure 5.12 A coding scheme for an information source: the source parser groups the source output sequence $\{U_\ell\}$ into messages $\{\mathbf{V}_{\ell'}\}$. The message encoder then assigns a codeword $\mathbf{C}_{\ell'}$ to each possible message $\mathbf{V}_{\ell'}$.

Now our compression system looks as shown in Figure 5.12. The *source parser* is a device that groups ν incoming source symbols $(U_1, \ldots, U_\nu)$ together to a new message $\mathbf{V}$. Note that because the source $\{U_\ell\}$ is memoryless, the different messages $\{\mathbf{V}_{\ell'}\}$ are independent. Therefore we only need to look at one such message $\mathbf{V}$ (where we omit the time index ℓ').

So let us now use an optimal code (i.e. a Huffman code) or at least a good code (e.g. a Fano code) for the message $\mathbf{V}$. Then from the Coding Theorem for a Single Random Message (Theorem 5.23) we know that

$$H(\mathbf{V}) \text{ bits} \le \mathrm{L}_{\mathrm{av}} < H(\mathbf{V}) + 1 \text{ bits}, \tag{5.94}$$

where L_{av} denotes the average codeword length for the codewords describing the *vector* messages $\mathbf{V}$.

Next note that it is not really fair to compare different L_{av} because, for larger ν, L_{av} also will be larger. So, to be correct we should compute the average codeword length necessary to describe *one* source symbol. Since $\mathbf{V}$ contains ν source symbols U_ℓ, the correct measure of performance is $\mathrm{L}_{\mathrm{av}}/\nu$.

Hence, we divide the whole expression (5.94) by ν:

$$\frac{H(\mathbf{V})}{\nu} \text{ bits} \le \frac{\mathrm{L}_{\mathrm{av}}}{\nu} < \frac{H(\mathbf{V})}{\nu} + \frac{1}{\nu} \text{ bits}, \tag{5.95}$$

and make use of (5.93),

$$\frac{\nu H(U)}{\nu} \text{ bits} \le \frac{\mathrm{L}_{\mathrm{av}}}{\nu} < \frac{\nu H(U)}{\nu} + \frac{1}{\nu} \text{ bits}; \tag{5.96}$$

i.e.,

$$H(U) \text{ bits} \leq \frac{\mathrm{L}_{\mathrm{av}}}{\nu} < H(U) + \frac{1}{\nu} \text{ bits}. \tag{5.97}$$

Note that again we assume that the entropies are measured in bits.

We immediately get the following main result, also known as *Shannon's Source Coding Theorem*.

Theorem 5.28 (Coding Theorem for a DMS)
There exists a binary prefix-free code of a ν-block message from a discrete memoryless source (DMS) such that the average number $\mathrm{L}_{\mathrm{av}}/\nu$ of binary code digits per source letter satisfies

$$\frac{\mathrm{L}_{\mathrm{av}}}{\nu} < H(U) + \frac{1}{\nu} \textit{ bits}, \tag{5.98}$$

where $H(U)$ is the entropy of a single source letter measured in bits. Conversely, for every *binary code of a ν-block message,*

$$\frac{\mathrm{L}_{\mathrm{av}}}{\nu} \geq H(U) \textit{ bits}. \tag{5.99}$$

Note that everywhere we need to choose the units of the entropy to be in bits.

We would like to discuss this result briefly. The main point to note here is that by choosing ν large enough, we can approach the ultimate limit of compression $H(U)$ *arbitrarily closely* when using a Huffman or a Fano code. Hence, the entropy $H(U)$ is the amount of information that is packed in the output of the discrete memoryless source U! In other words, we can compress any DMS to $H(U)$ bits on average, but not less. This is the first real justification of the usefulness of Definition 5.4.

We also see that in the end it does not make much difference whether we use a Huffman code or a suboptimal Fano code as both approach the ultimate limit for ν large enough.

On the other hand, note the price we have to pay: by making ν large, we not only increase the number of possible messages, and thereby make the code complicated, but also we introduce *delay* into the system as the encoder can only encode the message *after it has received a complete block of ν source symbols*! Basically, the more closely we want to approach the ultimate limit of entropy, the larger is our potential delay in the system.

We would like to mention that our choice of a source parser that splits the

source sequence into blocks of equal length is not the only choice. It is actually possible to design source parsers that will choose blocks of varying length depending on the arriving source symbols and their probabilities. By trying to combine more likely source symbols to larger blocks, while less likely symbols are grouped to smaller blocks, we can further improve on the compression rate of our system. A parser that is optimal in a specific sense is the so-called *Tunstall source parser* [Tun67], [Mas96]. The details are outside the scope of this introduction. However, note that whatever source parser and whatever message encoder we choose, we can never beat the lower bound in (5.99).

All the systems we have discussed here contain one common drawback: we always have assumed that the probability statistics of the source is known in advance when designing the system. In a practical situation this is often not the case. What is the probability distribution of a digitized speech in a telephone system? Or of English ASCII text in comparison to French ASCII text?[4] Or of different types of music? A really practical system should work independently of the source; i.e., it should estimate the probabilities of the source symbols on the fly and adapt to it automatically. Such a system is called a *universal compression scheme*. Again, the details are outside of the scope of this introduction, but we would like to mention that such schemes exist and that commonly used compression algorithms like, e.g., ZIP successfully implement such schemes.

5.6 Some historical background

The Fano code is in the literature usually known as the *Shannon–Fano code*, even though it is an invention of Professor Robert Fano from MIT [Fan49] and not of Shannon. To make things even worse, there exists another code that is also known as the *Shannon–Fano code*, but actually should be called the *Shannon code* because it was proposed by Shannon [Sha48, Sec. 9]: the construction of the Shannon code also starts with the ordering of the symbols according to decreasing probability. The ith codeword with probability p_i is then obtained by writing the cumulative probability

$$F_i \triangleq \sum_{j=1}^{i-1} p_j \tag{5.100}$$

in binary form. For example, $F_i = 0.625$ in binary form is .101 yielding a codeword 101, or $F_i = 0.3125$ is written in binary as .0101, which then results

[4] Recall the definition of ASCII given in Table 2.2.

in a codeword 0101. Since in general this binary expansion might be infinitely long, Shannon gave the additional rule that the expansion shall be carried out to exactly l_i positions, where

$$l_i \triangleq \left\lceil \log_2 \frac{1}{p_i} \right\rceil. \tag{5.101}$$

So if $F_i = 0.625$ (with binary form .101) and p_i is such that $l_i = 5$, then the resulting codeword is 10100, or if $F_i = 0.6$, which in binary form is

$$.10011001100\ldots,$$

and p_i is such that $l_i = 3$, then the resulting codeword is 100.

It is not difficult to show that this code is prefix-free. In particular it is straightforward to show that the Kraft Inequality (Theorem 4.8) is satisfied:

$$\sum_{i=1}^{r} 2^{-l_i} = \sum_{i=1}^{r} 2^{-\left\lceil \log_2 \frac{1}{p_i} \right\rceil} \leq \sum_{i=1}^{r} 2^{-\log_2 \frac{1}{p_i}} = \sum_{i=1}^{r} p_i = 1, \tag{5.102}$$

where we have used that

$$l_i = \left\lceil \log_2 \frac{1}{p_i} \right\rceil \geq \log_2 \frac{1}{p_i}. \tag{5.103}$$

Shannon's code performs similarly to the Fano code of Definition 5.17, but Fano's code is in general slightly better, as can be seen by the fact that in (5.68) we have an inequality while in (5.101) we have, by definition, equality always. However – and that is probably one of the reasons[5] why the two codes are mixed up and both are known under the same name *Shannon–Fano code* – both codes satisfy the Coding Theorem for a Single Random Message (Theorem 5.23).

Actually, one also finds that *any* code that satisfies (5.101) is called a *Shannon–Fano code*! And to complete the confusion, sometimes the Shannon–Fano code is also known as *Shannon–Fano–Elias code* [CT06, Sec. 5.9]. The reason is that the Shannon code was the origin of *arithmetic coding*, which is an elegant and efficient extension of Shannon's idea, applied to the compression of the output sequence of a random source. It is based on the insight that it is not necessary to order the output sequences $\mathbf{u}$ according to their probabilities, but that it is sufficient to have them ordered lexicographically (according to the alphabet). The codeword for $\mathbf{u}$ is then the truncated binary form of the

[5] Another reason is that Shannon, when introducing his code in [Sha48, Sec. 9], also refers to Fano's code construction.

cumulative probability

$$F_{\mathbf{u}} \triangleq \sum_{\substack{\text{all sequences } \tilde{\mathbf{u}} \\ \text{that are alphabetically} \\ \text{before } \mathbf{u}}} p_{\tilde{\mathbf{u}}}, \tag{5.104}$$

where $p_{\tilde{\mathbf{u}}}$ is the probability of $\tilde{\mathbf{u}}$. However, in order to guarantee that the code is prefix-free and because the output sequences are ordered lexicographically and not according to probability, it is necessary to increase the codeword length by 1; i.e., for arithmetic coding we have the rule that the codeword length is

$$l_{\mathbf{u}} \triangleq \left\lceil \log_2 \frac{1}{p_{\mathbf{u}}} \right\rceil + 1. \tag{5.105}$$

Note further that, since the sequences are ordered lexicographically, it is also possible to compute the cumulative probability (5.104) of a particular source output sequence $\mathbf{u}$ iteratively without having to know the probabilities of all other source sequences. These ideas have been credited to the late Professor Peter Elias from MIT (hence the name *Shannon–Fano–Elias coding*), but actually Elias denied this. The concept has probably come from Shannon himself during a talk that he gave at MIT.

For an easy-to-read introduction to arithmetic coding including its history, the introductory chapter of [Say99] is highly recommended.

5.7 Further reading

For more information about data compression, the lecture notes [Mas96] of Professor James L. Massey from ETH, Zurich, are highly recommended. They read very easily and are very precise. The presentation of the material in Chapters 4 and 5 is strongly inspired by these notes. Besides the generalization of Huffman codes to D-ary alphabets and the variable-length–to–block Tunstall codes, one also finds there details of a simple, but powerful, universal data compression scheme called *Elias–Willems coding*.

For the more commonly used *Lempel–Ziv universal compression scheme* we refer to [CT06]. This is also a good place to learn more about entropy and its properties.

One of the best books on information theory is by Robert Gallager [Gal68]; however, it is written at an advanced level. It is fairly old and therefore does not cover more recent discoveries, but it gives a very deep treatment of the foundations of information theory.

5.8 Appendix: Uniqueness of the definition of entropy

In Section 5.1 we tried to motivate the definition of entropy. Even though we partially succeeded, we were not able to provide a full justification of Definition 5.4. While Shannon did provide a mathematical justification [Sha48, Sec. 6], he did not consider it very important. We omit Shannon's argument, but instead we will now quickly summarize a slightly different result that was presented in 1956 by Aleksandr Khinchin. Khinchin specified four properties that entropy is supposed to have and then proved that, given these four properties, (5.10) is the only possible definition.

We define $H_r(p_1,\dots,p_r)$ to be a function of r probabilities $p_1,\dots,p_r$ that sum up to 1:

$$\sum_{i=1}^{r} p_i = 1. \tag{5.106}$$

We ask this function to satisfy the following four properties.

(1) For any r, $H_r(p_1,\dots,p_r)$ is continuous (i.e. a slight change to the values of p_i will only cause a slight change to H_r) and symmetric in $p_1,\dots,p_r$ (i.e. changing the order of the probabilities does not affect the value of H_r).

(2) Any event of probability zero does not contribute to H_r:

$$H_{r+1}(p_1,\dots,p_r,0) = H_r(p_1,\dots,p_r). \tag{5.107}$$

(3) H_r is maximized by the uniform distribution:

$$H_r(p_1,\dots,p_r) \le H_r\left(\frac{1}{r},\dots,\frac{1}{r}\right). \tag{5.108}$$

(4) If we partition the $m\cdot r$ possible outcomes of a random experiment into m groups, each group containing r elements, then we can do the experiment in two steps:

(i) determine the group to which the actual outcome belongs,

(ii) find the outcome in this group.

Let $p_{j,i}$, $1\le j\le m$, $1\le i\le r$, be the probabilities of the outcomes in this random experiment. Then the total probability of all outcomes in group j is given by

$$q_j = \sum_{i=1}^{r} p_{j,i}, \tag{5.109}$$

and the conditional probability of outcome i from group j is then given by

$$\frac{p_{j,i}}{q_j}. \tag{5.110}$$

Now $H_{m\cdot r}$ can be written as follows:

$$H_{m\cdot r}(p_{1,1}, p_{1,2}, \ldots, p_{m,r}) = H_m(q_1, \ldots, q_m) + \sum_{j=1}^{m} q_j H_r\left(\frac{p_{j,1}}{q_j}, \ldots, \frac{p_{j,r}}{q_j}\right); \quad (5.111)$$

i.e., the uncertainty can be split into the uncertainty of choosing a group and the uncertainty of choosing one particular outcome of the chosen group, averaged over all groups.

Theorem 5.29 *The only functions H_r that satisfy the above four conditions are of the form*

$$H_r(p_1, \ldots, p_r) = -c \sum_{i=1}^{r} p_i \log_2 p_i, \quad (5.112)$$

where the constant $c > 0$ decides about the units of H_r.

Proof This theorem was proven by Aleksandr Khinchin in 1956, i.e. after Shannon had defined entropy. The article was first published in Russian [Khi56], and then in 1957 it was translated into English [Khi57]. We omit the details. □

References

[CT06] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, 2nd edn. John Wiley & Sons, Hoboken, NJ, 2006.

[Fan49] Robert M. Fano, "The transmission of information," Research Laboratory of Electronics, Massachusetts Institute of Technology (MIT), Technical Report No. 65, March 17, 1949.

[Gal68] Robert G. Gallager, *Information Theory and Reliable Communication*. John Wiley & Sons, New York, 1968.

[Har28] Ralph Hartley, "Transmission of information," *Bell System Technical Journal*, vol. 7, no. 3, pp. 535–563, July 1928.

[Khi56] Aleksandr Y. Khinchin, "On the fundamental theorems of information theory," (in Russian), *Uspekhi Matematicheskikh Nauk XI*, vol. 1, pp. 17–75, 1956.

[Khi57] Aleksandr Y. Khinchin, *Mathematical Foundations of Information Theory*. Dover Publications, New York, 1957.

[Mas96] James L. Massey, *Applied Digital Information Theory I and II,* Lecture notes, Signal and Information Processing Laboratory, ETH Zurich, 1995/1996. Available: http://www.isiweb.ee.ethz.ch/archive/massey_scr/

[Say99] Jossy Sayir, "On coding by probability transformation," Ph.D. dissertation, ETH Zurich, 1999, Diss. ETH No. 13099. Available: http://e-collection.ethbib.ethz.ch/view/eth:23000

[Sha37] Claude E. Shannon, "A symbolic analysis of relay and switching circuits," Master's thesis, Massachusetts Institute of Technology (MIT), August 1937.

[Sha48] Claude E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July and October 1948. Available: http://moser.cm.nctu.edu.tw/nctu/doc/shannon1948.pdf

[Tun67] Brian P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Institute of Technology, September 1967.