Probability and Computing

Randomization and Probabilistic Techniques in Algorithms and Data Analysis

Second Edition

Michael Mitzenmacher   Eli Upfal

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107154889
10.1017/9781316651124

First published 2017

Printed in the United Kingdom by Clays, St Ives plc

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data
Names: Mitzenmacher, Michael, 1969– author. | Upfal, Eli, 1954– author.
Title: Probability and computing / Michael Mitzenmacher Eli Upfal.
Description: Second edition. | Cambridge, United Kingdom ;
New York, NY, USA : Cambridge University Press, [2017] |
Includes bibliographical references and index.
Identifiers: LCCN 2016041654 | ISBN 9781107154889
Subjects: LCSH: Algorithms. | Probabilities. | Stochastic analysis.
Classification: LCC QA274.M574 2017 | DDC 518/.1 – dc23
LC record available at https://lccn.loc.gov/2016041654

ISBN 978-1-107-15488-9 Hardback

Additional resources for this publication at www.cambridge.org/Mitzenmacher.

CHAPTER TEN

Entropy, Randomness, and Information

Suppose that we have two biased coins. One comes up heads with probability 3/4, and the other comes up heads with probability 7/8. Which coin produces more randomness per flip? In this chapter, we introduce the *entropy* function as a universal measure of randomness. In particular, we show that the number of independent unbiased random bits that can be extracted from a sequence of biased coin flips corresponds to the entropy of the coin. Entropy also plays a fundamental role in information and communication. To demonstrate this role, we examine some basic results in compression and coding and see how they relate to entropy. The main result we prove is Shannon's coding theorem for the binary symmetric channel, one of the fundamental results of the field of information theory. Our proof of Shannon's theorem uses several ideas that we have developed in previous chapters, including Chernoff bounds, Markov's inequality, and the probabilistic method.

10.1. The Entropy Function

The *entropy* of a random variable is a function of its distribution that, as we shall see, gives a measure of the randomness of the distribution.

Definition 10.1:

1. *The* entropy *in bits of a discrete random variable X is given by*
$$H(X) = -\sum_{x} \Pr(X = x)\log_2 \Pr(X = x),$$
where the summation is over all values x in the range of X. Equivalently, we may write
$$H(X) = \mathbf{E}\left[\log_2 \frac{1}{\Pr(X)}\right].$$
2. *The* binary entropy function $H(p)$ *for a random variable that assumes only two possible outcomes, one of which occurs with probability p, is*
$$H(p) = -p \log_2 p - (1-p)\log_2(1-p).$$

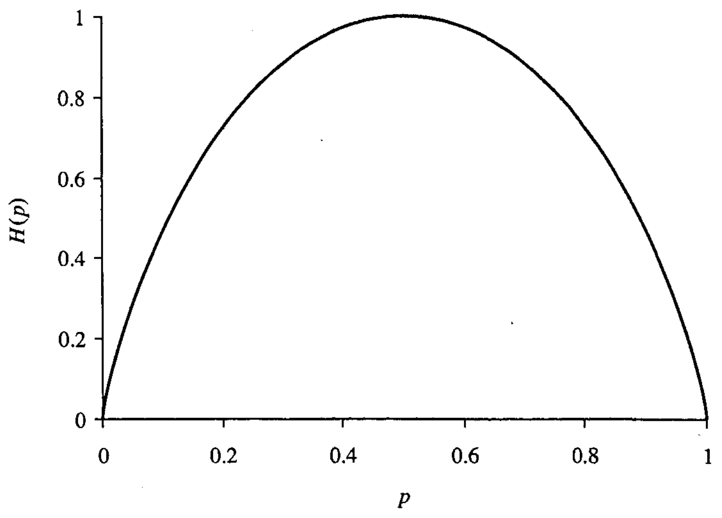

Figure 10.1: The binary entropy function.

We define $H(0) = H(1) = 0$, so the binary entropy function is continuous in the interval $[0, 1]$. The function is drawn in Figure 10.1.

For our two biased coins, the entropy of the coin that comes up heads with probability $3/4$ is

$$H\left(\frac{3}{4}\right) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 2 - \frac{3}{4}\log_2 3 \approx 0.8113,$$

while the entropy of the coin that comes up heads with probability $7/8$ is

$$H\left(\frac{7}{8}\right) = -\frac{7}{8}\log_2\frac{7}{8} - \frac{1}{8}\log_2\frac{1}{8} = 3 - \frac{7}{8}\log_2 7 \approx 0.5436.$$

Hence the coin that comes up heads with probability $3/4$ has a larger entropy.

Taking the derivative of $H(p)$,

$$\frac{dH(p)}{dp} = -\log_2 p + \log_2(1-p) = \log_2\frac{1-p}{p},$$

we see that $H(p)$ is maximized when $p = 1/2$ and that $H(1/2) = 1$ bit. One way of interpreting this statement is to say: each time we flip a two-sided coin, we get out at most 1 bit worth of randomness, and we obtain exactly 1 bit of randomness when the coin is fair. Although this seems quite clear, it is not yet clear in what sense $H(3/4) = 2 - \frac{3}{4}\log_2 3$ means that we obtain $H(3/4)$ random bits each time we flip a coin that lands heads with probability $3/4$. We clarify this later in the chapter.

As another example, the entropy of a standard six-sided die that comes up on each side with probability $1/6$ has entropy $\log_2 6$. In general, a random variable that has n equally likely outcomes has entropy

$$-\sum_{i=1}^{n}\frac{1}{n}\log_2\frac{1}{n} = \log_2 n.$$

The entropy of an eight-sided die is therefore 3 bits. This result should seem quite natural; if the faces of the die were numbered from 0 to 7 written in binary, then the outcome of the die roll would give a sequence of 3 bits uniform over the set $\{0, 1\}^3$, which is equivalent to 3 bits generated independently and uniformly at random.

It is worth emphasizing that the entropy of a random variable X depends not on the values that X can take but only on the probability distribution of X over those values. The entropy of an eight-sided die does not depend on what numbers are on the faces of the die; it only matters that all eight sides are equally likely to come up. This property does not hold for the expectation or variance of X, but it does makes sense for a measure of randomness. To measure the randomness in a die, we should not care about what numbers are on the faces but only about how often the die comes up on each side.

Often in this chapter we consider the entropy of a sequence of independent random variables, such as the entropy of a sequence of independent coin flips. For such situations, the following lemma allows us to consider the entropy of each random variable to find the entropy of the sequence.

Lemma 10.1: *Let X_1 and X_2 be independent random variables, and let $Y = (X_1, X_2)$. Then*

$$H(Y) = H(X_1) + H(X_2).$$

Of course, the lemma is trivially extended by induction to the case where Y is any finite sequence of independent random variables.

Proof: In what follows, the summations are to be taken over all possible values that can be assumed by X_1 and X_2. The result follows by using the independence of X_1 and X_2 to simplify the expression:

$$\begin{aligned}
H(Y) &= -\sum_{x_1,x_2} \Pr((X_1,X_2) = (x_1,x_2)) \log_2 \Pr((X_1,X_2) = (x_1,x_2)) \\
&= -\sum_{x_1,x_2} \Pr(X_1 = x_1)\Pr(X_2 = x_2) \log_2(\Pr(X_1 = x_1)\Pr(X_2 = x_2)) \\
&= -\sum_{x_1,x_2} \Pr(X_1 = x_1)\Pr(X_2 = x_2)(\log_2 \Pr(X_1 = x_1) + \log_2 \Pr(X_2 = x_2)) \\
&= -\sum_{x_1}\sum_{x_2} \Pr(X_2 = x_2)\Pr(X_1 = x_1)\log_2 \Pr(X_1 = x_1) \\
&\quad -\sum_{x_2}\sum_{x_1} \Pr(X_1 = x_1)\Pr(X_2 = x_2)\log_2 \Pr(X_2 = x_2) \\
&= -\left(\sum_{x_1} \Pr(X_1 = x_1)\log_2 \Pr(X_1 = x_1)\right)\left(\sum_{x_2} \Pr(X_2 = x_2)\right) \\
&\quad -\left(\sum_{x_2} \Pr(X_2 = x_2)\log_2 \Pr(X_2 = x_2)\right)\left(\sum_{x_1} \Pr(X_1 = x_1)\right) \\
&= -\sum_{x_1} \Pr(X_1 = x_1)\log_2 \Pr(X_1 = x_1) - \sum_{x_2} \Pr(X_2 = x_2)\log_2 \Pr(X_2 = x_2) \\
&= H(X_1) + H(X_2).
\end{aligned}$$

■

10.2. Entropy and Binomial Coefficients

As a prelude to showing the various applications of entropy, we first demonstrate how it naturally arises in a purely combinatorial context.

Lemma 10.2: *Suppose that nq is an integer in the range* $[0, n]$. *Then*

$$\frac{2^{nH(q)}}{n+1} \le \binom{n}{nq} \le 2^{nH(q)}.$$

Proof: The statement is trivial if $q = 0$ or $q = 1$, so assume that $0 < q < 1$. To prove the upper bound, notice that by the binomial theorem we have

$$\binom{n}{nq} q^{qn}(1-q)^{(1-q)n} \le \sum_{k=0}^{n} \binom{n}{k} q^k (1-q)^{n-k} \le (q + (1-q))^n = 1.$$

Hence,

$$\binom{n}{nq} \le q^{-qn}(1-q)^{-(1-q)n} = 2^{-qn\log_2 q} 2^{-(1-q)n\log_2(1-q)} = 2^{nH(q)}.$$

For the lower bound, we know that $\binom{n}{nq} q^{qn}(1-q)^{(1-q)n}$ is one term of the expression $\sum_{k=0}^{n}\binom{n}{k} q^k(1-q)^{n-k}$. We show that it is the *largest* term. Consider the difference between two consecutive terms as follows:

$$\binom{n}{k} q^k(1-q)^{n-k} - \binom{n}{k+1} q^{k+1}(1-q)^{n-k-1}$$
$$= \binom{n}{k} q^k(1-q)^{n-k}\left(1 - \frac{n-k}{k+1}\frac{q}{1-q}\right).$$

This difference is nonnegative whenever

$$1 - \frac{n-k}{k+1}\frac{q}{1-q} \ge 0$$

or (equivalently, after some algebra) whenever

$$k \ge qn - 1 + q.$$

The terms are therefore increasing up to $k = qn$ and decreasing after that point. Thus $k = qn$ gives the largest term in the summation.

Since the summation has $n + 1$ terms and since $\binom{n}{nq} q^{qn}(1-q)^{(1-q)n}$ is the largest term, we have

$$\binom{n}{nq} q^{qn}(1-q)^{(1-q)n} \ge \frac{1}{n+1}$$

or

$$\binom{n}{nq} \ge \frac{q^{-qn}(1-q)^{-(1-q)n}}{n+1} = \frac{2^{nH(q)}}{n+1}. \qquad \blacksquare$$

We often use the following slightly more specific corollary.

Corollary 10.3: *When* $0 \le q \le 1/2$,

$$\binom{n}{\lfloor nq \rfloor} \le 2^{nH(q)}; \tag{10.1}$$

similarly, when $1/2 \le q \le 1$,

$$\binom{n}{\lceil nq \rceil} \le 2^{nH(q)}. \tag{10.2}$$

For $1/2 \le q \le 1$,

$$\frac{2^{nH(q)}}{n+1} \le \binom{n}{\lfloor nq \rfloor}; \tag{10.3}$$

similarly, when $0 \le q \le 1/2$,

$$\frac{2^{nH(q)}}{n+1} \le \binom{n}{\lceil nq \rceil}. \tag{10.4}$$

Proof: We first prove Eqn. (10.1); the proof of Eqn. (10.2) is entirely similar. When $0 \le q \le 1/2$,

$$\binom{n}{\lfloor nq \rfloor} q^{qn}(1-q)^{(1-q)n} \le \binom{n}{\lfloor nq \rfloor} q^{\lfloor qn \rfloor}(1-q)^{n-\lfloor qn \rfloor} \le \sum_{k=0}^{n} \binom{n}{k} q^k (1-q)^{n-k} = 1,$$

from which we can proceed exactly as in Lemma 10.2.

Eqn. (10.3) holds because, when $q \ge 1/2$, Lemma 10.2 gives

$$\binom{n}{\lfloor nq \rfloor} \ge \frac{2^{nH(\lfloor nq \rfloor/n)}}{n+1} \ge \frac{2^{nH(q)}}{n+1};$$

Eqn. (10.4) is derived similarly. ■

Although these bounds are loose, they are sufficient for our purposes. The relation between the combinatorial coefficients and the entropy function arises repeatedly in the proofs of this chapter when we consider a sequence of biased coin tosses, where the coin lands heads with probability $p > 1/2$. Applying the Chernoff bound, we know that, for sufficiently large n, the number of heads will almost always be close to np. Thus the sequence will almost always be one of roughly $\binom{n}{np} \approx 2^{nH(p)}$ sequences, where the approximation follows from Lemma 10.2. Moreover, each such sequence occurs with probability roughly

$$p^{np}(1-p)^{n(1-p)} \approx 2^{-nH(p)}.$$

Hence, when we consider the outcome of n flips with a biased coin, we can essentially restrict ourselves to the roughly $2^{nH(p)}$ outcomes that occur with roughly equal probability.

10.3. Entropy: A Measure of Randomness

One way of interpreting the entropy of a random variable is as a measure of how many unbiased, independent bits can be extracted, on average, from one instantiation of the random variable. We consider this question in the context of a biased coin, showing that, for sufficiently large n, the expected number of bits that can be extracted from n flips of a coin that comes up heads with probability $p > 1/2$ is essentially $nH(p)$. In other words, on average, one can generate approximately $H(p)$ independent bits from each flip of a coin with entropy $H(p)$. This result can be generalized to other random variables, but we focus on the specific case of biased coins here (and throughout this chapter) to keep the arguments more transparent.

We begin with a definition that clarifies what we mean by extracting random bits.

Definition 10.2: *Let $|y|$ be the number of bits in a sequence of bits y. An* extraction function Ext *takes as input the value of a random variable X and outputs a sequence of bits y such that*

$$\Pr(\text{Ext}(X) = y \mid |y| = k) = 1/2^k$$

whenever $\Pr(|y| = k) > 0$.

In the case of a biased coin, the input X is the outcome of n flips of our biased coin. The number of bits in the output is not fixed but instead can depend on the input. If the extraction function outputs k bits, we can think of these bits as having been generated independently and uniformly at random, since each sequence of k bits is equally likely to appear. Also, there is nothing in the definition that requires that the extraction function be efficient to compute. We do not concern ourselves with efficiency here, although we do consider an efficient extraction algorithm in Exercise 10.12.

As a first step toward proving our results about extracting unbiased bits from biased coins, we consider the problem of extracting random bits from a uniformly distributed integer random variable. For example, let X be an integer chosen uniformly at random from $\{0, \dots, 7\}$, and let Y be the sequence of 3 bits obtained when we write X as a binary number. If $X = 0$ then $Y = 000$, and if $X = 7$ then $Y = 111$. It is easy to check that every sequence of 3 bits is equally likely to arise, so we have a trivial extraction function Ext by associating any input X with the corresponding output Y.

Things are slightly harder when X is uniform over $\{0, \dots, 11\}$. If $X \le 7$, then we can again let Y be the sequence of 3 bits obtained when we write X in binary. This leaves the case $X \in \{8, 9, 10, 11\}$. We can associate each of these four possibilities with a distinct sequence of 2 bits, for example, by letting Y be the sequence of 2 bits obtained from writing $X - 8$ as a binary number. Thus, if $X = 8$ then $Y = 00$, and if $X = 11$ then

Input	0	1	2	3	4	5	6	7
Output	000	001	010	011	100	101	110	111

Input	0	1	2	3	4	5	6	7	8	9	10	11
Output	000	001	010	011	100	101	110	111	00	01	10	11

Figure 10.2: Extraction functions for numbers that are chosen uniformly at random from $\{0, \dots, 7\}$ and $\{0, \dots, 11\}$.

$Y = 11$. The entire extraction function is shown in Figure 10.2. Every 3-bit sequence arises with the same probability $1/12$, and every 2-bit sequence arises with the same probability $1/12$, so Definition 10.2 is satisfied.

We generalize from these examples to the following theorem.

Theorem 10.4: *Suppose that the value of a random variable X is chosen uniformly at random from the integers $\{0, \dots, m-1\}$, so that $H(X) = \log_2 m$. Then there is an extraction function for X that outputs on average at least $\lfloor \log_2 m \rfloor - 1 = \lfloor H(X) \rfloor - 1$ independent and unbiased bits.*

Proof: If $m > 1$ is a power of 2, then the extraction function can simply output the bit representation of the input X using $\log_2 m$ bits. (If $m = 1$, then it outputs nothing or, equivalently, an empty sequence.) All output sequences have $\log_2 m$ bits, and all sequences of $\log_2 m$ bits are equally likely to appear, so this satisfies Definition 10.2. If m is not a power of 2, then matters become more complicated. We describe the extraction function recursively. (A nonrecursive description is given in Exercise 10.8.) Let $\alpha = \lfloor \log_2 m \rfloor$. If $X \le 2^\alpha - 1$, then the function outputs the α-bit binary representation of X; all sequences of α bits are equally likely to be output in this case. If $X \ge 2^\alpha$, then $X - 2^\alpha$ is uniformly distributed in the set $\{0, \dots, m - 2^\alpha - 1\}$, which is smaller than the set $\{0, \dots, m\}$. The extraction function can then recursively produce the output from the extraction function for the variable $X - 2^\alpha$.

The recursive extraction function maintains the property that, for every k, each of the 2^k sequences of k bits is output with the same probability. We claim by induction that the expected number of unbiased, independent bits produced by this extraction function is at least $\lfloor \log_2 m \rfloor - 1$. The cases where m is a power of 2 are trivial. Otherwise, by induction, the number of bits Y in the output satisfies

$$\begin{aligned}\mathbf{E}[Y] &\ge \frac{2^\alpha}{m}\alpha + \frac{m-2^\alpha}{m}(\lfloor \log_2(m-2^\alpha) \rfloor - 1) \\ &= \alpha - \frac{m-2^\alpha}{m}(\alpha - \lfloor \log_2(m-2^\alpha) \rfloor + 1).\end{aligned}$$

Suppose $\lfloor \log_2(m - 2^\alpha) \rfloor = \beta$, where $0 \le \beta \le \alpha - 1$. Then $(m - 2^\alpha)/m$ is maximized when m is as large as possible, which corresponds to $m = 2^\alpha + 2^{\beta+1} - 1$. Hence

$$\begin{aligned}\mathbf{E}[Y] &\ge \alpha - \frac{2^{\beta+1} - 1}{2^\alpha + 2^{\beta+1} - 1}(\alpha - \beta + 1)\\ &\ge \alpha - \frac{1}{2^{\alpha-\beta-1} + 1}(\alpha - \beta + 1)\\ &\ge \alpha - 1,\end{aligned}$$

where we use $2^\alpha + 2^{\beta+1} - 1 \ge (2^{\beta+1} - 1)(2^{\alpha-\beta-1} + 1)$ for the second line, and $x/(2^{x-2} + 1) \le 1$ for integers $x \ge 2$ for the third line.

This completes the induction. ■

We use Theorem 10.4 in our proof of the main result of this section.

Theorem 10.5: *Consider a coin that comes up heads with probability $p > 1/2$. For any constant $\delta > 0$ and for n sufficiently large:*

1. *there exists an extraction function* Ext *that outputs, on an input sequence of n independent flips, an average of at least $(1 - \delta)nH(p)$ independent random bits; and*
2. *the average number of bits output by any extraction function* Ext *on an input sequence of n independent flips is at most $nH(p)$.*

Proof: We begin by describing an extraction function that generates, on average, at least $(1 - \delta)nH(p)$ random bits from n flips of the biased coin. We saw before that, in the case of a biased coin, the outcome of n flips is most likely to be one of roughly $2^{nH(p)}$ sequences, each occurring with probability roughly $2^{-nH(p)}$. If we actually had a uniform distribution of this type, we could use the extraction function that we have just described for numbers chosen uniformly at random to obtain on average almost $nH(p)$ uniform random bits. In what follows, we handle the technical details that arise because the distribution is not exactly uniform.

There are $\binom{n}{j}$ possible sequences with exactly j heads, and each of them has the same probability of occurring, $p^j(1 - p)^{n-j}$. For each value of j, $0 \le j \le n$, we map each of the $\binom{n}{j}$ sequences with j heads to a unique integer in the set $\left\{0, \dots, \binom{n}{j} - 1\right\}$. When j heads come up, we map the sequence to the corresponding number. Conditioned on there being j heads, this number is uniform on the integers $\left\{0, \dots, \binom{n}{j} - 1\right\}$, and hence we can apply the extraction function of Theorem 10.4 designed for this case. Let Z be a random variable representing the number of heads flipped, and let B be the random variable representing the number of bits our extraction function produces. Then

$$\mathbf{E}[B] = \sum_{k=0}^{n} \Pr(Z = k)\mathbf{E}[B \mid Z = k]$$

and, by Theorem 10.4,

$$\mathbf{E}[B \mid Z = k] \ge \left\lfloor \log_2 \binom{n}{k} \right\rfloor - 1.$$

Let $\varepsilon < \min(p - 1/2, 1 - p)$ represent a constant to be determined. We compute a lower bound for $\mathbf{E}[B]$ by considering only values of k with $n(p - \varepsilon) \le k \le n(p + \varepsilon)$.

For every such k,

$$\binom{n}{k} \ge \binom{n}{\lfloor n(p+\varepsilon)\rfloor} \ge \frac{2^{nH(p+\varepsilon)}}{n+1},$$

where the last inequality follows from Corollary 10.3. Hence

$$\begin{aligned}
\mathbf{E}[B] &\ge \sum_{k=\lfloor n(p-\varepsilon)\rfloor}^{\lceil n(p+\varepsilon)\rceil} \Pr(Z=k)\mathbf{E}[B \mid Z=k] \\
&\ge \sum_{k=\lfloor n(p-\varepsilon)\rfloor}^{\lceil n(p+\varepsilon)\rceil} \Pr(Z=k)\left(\left\lfloor \log_2 \binom{n}{k} \right\rfloor - 1\right) \\
&\ge \left(\log_2 \frac{2^{nH(p+\varepsilon)}}{n+1} - 2\right) \sum_{k=\lfloor n(p-\varepsilon)\rfloor}^{\lceil n(p+\varepsilon)\rceil} \Pr(Z=k) \\
&\ge (nH(p+\varepsilon) - \log_2(n+1) - 2)\Pr(|Z-np| \le \varepsilon n).
\end{aligned}$$

Now $\mathbf{E}[Z] = np$, and $\Pr(|Z-np| > \varepsilon n)$ can be bounded by using the Chernoff bound of Eqn. (4.6), giving

$$\Pr(|Z-np| > \varepsilon n) \le 2\mathrm{e}^{-n\varepsilon^2/3p}.$$

Hence

$$\mathbf{E}[B] \ge (nH(p+\varepsilon) - \log_2(n+1) - 2)(1 - 2\mathrm{e}^{-n\varepsilon^2/3p}).$$

We conclude that, for any constant $\delta > 0$, we can have

$$\mathbf{E}[B] \ge (1-\delta)nH(p)$$

by choosing ε sufficiently small and n sufficiently large. For example, for sufficiently small ε,

$$nH(p+\varepsilon) \ge (1-\delta/3)nH(p),$$

and when $n > (3p/\varepsilon^2)\ln(6/\delta)$ we have

$$1 - 2\mathrm{e}^{-n\varepsilon^2/3p} \ge 1 - \delta/3.$$

Hence, with these choices,

$$\mathbf{E}[B] \ge ((1-\delta/3)nH(p) - \log_2(n+1) - 2)(1-\delta/3).$$

As long as we now also choose n sufficiently large that $(\delta/3)nH(p)$ is greater than $\log_2(n+1)+2$, we have

$$\mathbf{E}[B] \ge ((1-2\delta/3)nH(p))(1-\delta/3) \ge (1-\delta)nH(p),$$

proving there exists an extraction function that can extract $(1-\delta)nH(p)$ independent and uniform bits on average from n flips of the biased coin.

We now show that no extraction function can obtain more than $nH(p)$ bits on average. The proof relies on the following basic fact: If an input sequence x occurs with probability q, then the corresponding output sequence $\mathrm{Ext}(x)$ can have at most

$|\text{Ext}(x)| \le \log_2(1/q)$ bits. This is because all sequences with $|\text{Ext}(x)|$ bits would have probability at least q, so

$$2^{|\text{Ext}(x)|}q \le 1,$$

giving the desired bound on $\text{Ext}(x)$. Given any extraction function, if B is a random variable representing the number of bits our extraction function produces on input X, then

$$\begin{aligned}\mathbf{E}[B] &= \sum_x \Pr(X = x)|\text{Ext}(x)| \\ &\le \sum_x \Pr(X = x)\log_2 \frac{1}{\Pr(X = x)} \\ &= \mathbf{E}\left[\log_2 \frac{1}{\Pr(X)}\right] \\ &= H(X).\end{aligned}$$ ■

Another natural question to ask is how we can generate biased bits from an unbiased coin. This question is partially answered in Exercise 10.11.

10.4. Compression

A second way of interpreting the entropy value comes from compression. Again suppose we have a coin that comes up heads with probability $p > 1/2$ and that we flip it n times, keeping track of which flips are heads and which flips are tails. We could represent every outcome by using one bit per flip, with 0 representing heads and 1 representing tails, and use a total of n bits. If we take advantage of the fact that the coin is biased, we can do better on average. For example, suppose that $p = 3/4$. For a pair of consecutive flips, we use 0 to represent that both flips were heads, 10 to represent that the first flip was heads and the second tails, 110 to represent that the first flip was tails and the second heads, and 111 to represent that both flips were tails. Then the average number of bits we use per pair of flips is

$$1 \cdot \frac{9}{16} + 2 \cdot \frac{3}{16} + 3 \cdot \frac{3}{16} + 3 \cdot \frac{1}{16} = \frac{27}{16} < 2.$$

Hence, on average, we can use less than the 1 bit per flip of the standard scheme by breaking a sequence of n flips into pairs and representing each pair in the manner shown. This is an example of *compression.*

It is worth emphasizing that the representation that we used here has a special property: if we write the representation of a sequence of flips, it can be uniquely decoded simply by parsing it from left to right. For example, the sequence

011110

corresponds to two heads, followed by two tails, followed by a heads and a tails. There is no ambiguity, because no other sequence of flips could produce this output.

Our representation has this property because no bit sequence we use to represent a pair of flips is the prefix of another bit sequence used in the representation. Representations with this property are called *prefix codes,* which are discussed further in Exercise 10.15.

Compression continues to be a subject of considerable study. When storing or transmitting information, saving bits usually corresponds to saving resources, so finding ways to reduce the number of used bits by taking advantage of the data's structure is often worthwhile.

We consider here the special case of compressing the outcome of a sequence of biased coin flips. For a biased coin with entropy $H(p)$, we show (a) that the outcome of n flips of the coin can be represented by approximately $nH(p)$ bits on average and (b) that approximately $nH(p)$ bits on average are necessary. In particular, any representation of the outcome of n flips of a fair coin essentially requires n bits. The entropy is therefore a measure of the average number of bits generated by each coin flip after compression. This argument can be generalized to any discrete random variable X, so that n independent, identically distributed random variables $X_1, X_2, \dots, X_n$ with the same distribution X can be represented using approximately $nH(X)$ bits on average. In the setting of compression, entropy can be viewed as measuring the amount of information in the input sequence. The larger the entropy of the sequence, the more bits are needed in order to represent it.

We begin with a definition that clarifies what we mean by compression in this context.

Definition 10.3: *A compression function* Com *takes as input a sequence of n coin flips, given as an element of $\{H, T\}^n$, and outputs a sequence of bits such that each input sequence of n flips yields a distinct output sequence.*

Definition 10.3 is rather weak, but it will prove sufficient for our purposes. Usually, compression functions must satisfy stronger requirements; for example, we may require a prefix code to simplify decoding. Using this weaker definition makes our lower-bound proof stronger. Also, though we are not concerned here with the efficiency of compressing and decompressing procedures, there are very efficient compression schemes that perform nearly optimally in many situations. We will consider an efficient compression scheme in Exercise 10.17.

The following theorem formalizes the relationship between the entropy of a biased coin and compression.

Theorem 10.6: *Consider a coin that comes up heads with probability $p > 1/2$. For any constant $\delta > 0$, when n is sufficiently large:*

1. *there exists a compression function* Com *such that the expected number of bits output by* Com *on an input sequence of n independent coin flips is at most $(1 + \delta)nH(p)$; and*
2. *the expected number of bits output by any compression function on an input sequence of n independent coin flips is at least $(1 - \delta)nH(p)$.*

Theorem 10.6 is quite similar to Theorem 10.5. The lower bound on the expected number of bits output by any compression function is slightly weaker. In fact, we could raise this lower bound to $nH(p)$ if we insisted that the code be a prefix code – so that no output is the prefix of any other – but we do not prove this here. The compression function we design to prove an upper bound on the expected number of output bits does yield a prefix code. Our construction of this compression function follows roughly the same intuition as Theorem 10.5. We know that, with high probability, the outcome from the n flips will be one of roughly $2^{nH(p)}$ sequences with roughly np heads. We can use about $nH(p)$ bits to represent each one of these sequences, yielding the existence of an appropriate compression function.

Proof of Theorem 10.6: We first show that there exists a compression function as guaranteed by the theorem. Let $\varepsilon > 0$ be a suitably small constant with $p - \varepsilon > 1/2$. Let X be the number of heads in n flips of the coin. The first bit output by the compression function we use as a flag. We set it to 0 if there are at least $n(p - \varepsilon)$ heads in the sequence and to 1 otherwise. When the first bit is a 1, the compression function uses the expensive default scheme, using 1 bit for each of the n flips. This requires that $n + 1$ total bits be output; however, by the Chernoff bound of Eqn. (4.5), the probability that this case happens is bounded by

$$\Pr(X < n(p - \varepsilon)) \le \mathrm{e}^{-n\varepsilon^2/2p}.$$

Now let us consider the case where there are at least $n(p - \varepsilon)$ heads. The number of coin-flip sequences of this form is

$$\sum_{j=\lceil n(p-\varepsilon)\rceil}^{n} \binom{n}{j} \le \sum_{j=\lceil n(p-\varepsilon)\rceil}^{n} \binom{n}{\lceil n(p-\varepsilon)\rceil} \le \frac{n}{2} 2^{nH(p-\varepsilon)}.$$

The first inequality arises because the binomial terms are decreasing as long as $j > n/2$, and the second is a consequence of Corollary 10.3. For each such sequence of coin flips, the compression function can assign a unique sequence of exactly $\lfloor nH(p - \varepsilon) + \log_2 n \rfloor$ bits to represent it, since

$$2^{\lfloor nH(p-\varepsilon)+\log_2 n \rfloor} \ge \frac{n}{2} 2^{nH(p-\varepsilon)}.$$

Including the flag bit, it therefore takes at most $nH(p - \varepsilon) + \log_2 n + 1$ bits to represent the sequences of coin flips with this many heads.

Totaling these results, we find that the expected number of bits required by the compression function is at most

$$\mathrm{e}^{-n\varepsilon^2/2p}(n + 1) + (1 - \mathrm{e}^{-n\varepsilon^2/2p})(nH(p - \varepsilon) + \log_2 n + 1) \le (1 + \delta)nH(p),$$

where the inequality holds by first taking ε sufficiently small and then taking n sufficiently large in a manner similar to that of Theorem 10.5.

We now show the lower bound. To begin, recall that the probability that a specific sequence with k heads is flipped is $p^k(1 - p)^{n-k}$. Because $p > 1/2$, if sequence S_1 has more heads than another sequence S_2, then S_1 is more likely to appear than S_2. Also, we have the following lemma.

Lemma 10.7: *If sequence S_1 is more likely than S_2, then the compression function that minimizes the expected number of bits in the output assigns a bit sequence to S_2 that is at least as long as S_1.*

Proof: Suppose that a compression function assigns a bit sequence to S_2 that is shorter than the bit sequence it assigns to S_1. We can improve the expected number of bits output by the compression function by switching the output sequences associated with S_1 and S_2, and therefore this compression function is not optimal. ∎

Hence sequences with more heads should get shorter strings from an optimal compression function.

We also make use of the following simple fact. If the compression function assigns distinct sequences of bits to represent each of s coin-flip sequences, then one of the output bit sequences for the s input sequences must have length at least $\log_2 s - 1$ bits. This is because there are at most $1 + 2 + 4 + \cdots + 2^b = 2^{b+1} - 1$ distinct bit sequences with up to b bits, so if each of s sequences of coin flips is assigned a bit sequence of at most b bits, then we must have $2^{b+1} > s$ and hence $b > \log_2 s - 1$.

Fix a suitably small $\varepsilon > 0$ and count the number of input sequences that have $\lfloor (p+\varepsilon)n \rfloor$ heads. There are $\binom{n}{\lfloor (p+\varepsilon)n \rfloor}$ sequences with $\lfloor (p+\varepsilon)n \rfloor$ heads and, by Corollary 10.3,

$$\binom{n}{\lfloor (p+\varepsilon)n \rfloor} \geq \frac{2^{nH(p+\varepsilon)}}{n+1}.$$

Hence any compression function must output at least $\log_2(2^{nH(p+\varepsilon)}/(n+1)) - 1 = nH(p+\varepsilon) - \log_2(n+1) - 1$ bits on at least one of the sequences of coin flips with $\lfloor (p+\varepsilon)n \rfloor$ heads. The compression function that minimizes the expected output length must therefore use at least this many bits to represent any sequence with fewer heads, by Lemma 10.7.

By the Chernoff bound of Eqn. (4.2), the number of heads X satisfies

$$\Pr(X \geq \lfloor n(p+\varepsilon) \rfloor) \leq \Pr(X \geq n(p+\varepsilon - 1/n)) \leq e^{-n(\varepsilon - 1/n)^2/3p} \leq e^{-n\varepsilon^2/12p}$$

as long as n is sufficiently large (specifically, $n > 2/\varepsilon$). We thus obtain, with probability at least $1 - e^{-n\varepsilon^2/12p}$, an input sequence with fewer than $\lfloor n(p+\varepsilon) \rfloor$ heads, and by our previous reasoning the compression function that minimizes the expected output length must still output at least $nH(p+\varepsilon) - \log_2(n+1) - 1$ bits in this case. The expected number of output bits is therefore at least

$$(1 - e^{-n\varepsilon^2/12p})(nH(p+\varepsilon) - \log_2(n+1) - 1).$$

This can be made to be at least $(1-\delta)nH(p)$ by first taking ε to be sufficiently small and then taking n to be sufficiently large. ∎

10.5.* Coding: Shannon's Theorem

We have seen how compression can reduce the expected number of bits required to represent data by changing the representation of the data. Coding also changes the representation of the data. Instead of reducing the number of bits required to represent the data, however, coding adds redundancy in order to protect the data against loss or errors.

In coding theory, we model the information being passed from a sender to a receiver through a *channel*. The channel may introduce noise, distorting the value of some of the bits during the transmission. The channel can be a wired connection, a wireless connection, or a storage network. For example, if I store data on a recordable medium and later try to read it back, then I am both the sender and the receiver, and the storage medium acts as the channel. In this section, we focus on one specific type of channel.

Definition 10.4: *The input to a* binary symmetric channel *with parameter p is a sequence of bits $x_1, x_2, \ldots$, and the output is a sequence of bits $y_1, y_2, \ldots$, such that* $\Pr(x_i = y_i) = 1 - p$ *independently for each i. Informally, each bit sent is flipped to the wrong value independently with probability p.*

To get useful information out of the channel, we may introduce redundancy to help protect against the introduction of errors. As an extreme example, suppose the sender wants to send the receiver a single bit of information over a binary symmetric channel. To protect against the possibility of error, the sender and receiver agree to repeat the bit n times. If $p < 1/2$, a natural decoding scheme for the receiver is to look at the n bits received and decide that the value that was received more frequently is the bit value the sender intended. The larger n is, the more likely the receiver determines the correct bit; by repeating the bit enough times, the probability of error can be made arbitrarily small. This example is considered more extensively in Exercise 10.18.

Coding theory studies the trade-off between the amount of redundancy required and the probability of a decoding error over various types of channels. For the binary symmetric channel, simply repeating bits may not be the best use of redundancy. Instead we consider more general *encoding functions*.

Definition 10.5: *A* (k, n) encoding function Enc: $\{0,1\}^k \rightarrow \{0,1\}^n$ *takes as input a sequence of k bits and outputs a sequence of n bits. A* (k, n) decoding function Dec: $\{0,1\}^n \rightarrow \{0,1\}^k$ *takes as input a sequence of n bits and outputs a sequence of k bits.*

With coding, the sender takes a k-bit message and encodes it into a block of $n \geq k$ bits via the encoding function. These bits are then sent over the channel. The receiver examines the n bits received and attempts to determine the original k-bit message using the decoding function.

Given a binary channel with parameter p and a target encoding length of n, we wish to determine the largest value of k so that there exist (k, n) encoding and decoding functions with the property that, for any input sequence of k bits, with suitably large probability the receiver decodes the correct input from the corresponding n-bit encoding sequence after it has been distorted by the channel.

Let $m \in \{0,1\}^k$ be the message to be sent and Enc(m) the sequence of bits sent over the channel. Let the random variable X denote the sequence of received bits. We require that Dec$(X) = m$ with probability at least $1 - \gamma$ for all possible messages m and a prechosen constant γ. If there were no noise, then we could send the original k bits over the channel. The noise reduces the information that the receiver can extract from each bit sent, and so the sender can reliably send messages of only about $k = n(1 - H(p))$ bits within each block of n bits. This result is known as Shannon's theorem, which we prove in the following form.

Theorem 10.8: *For a binary symmetric channel with parameter $p < 1/2$ and for any constants $\delta, \gamma > 0$, when n is sufficiently large:*

1. *for any $k \leq n(1 - H(p) - \delta)$, there exist (k, n) encoding and decoding functions such that the probability the receiver fails to obtain the correct message is at most γ for every possible k-bit input message; and*
2. *there are no (k, n) encoding and decoding functions with $k \geq n(1 - H(p) + \delta)$ such that the probability of decoding correctly is at least γ for a k-bit input message chosen uniformly at random.*

Proof: We first prove the existence of suitable (k, n) encoding and decoding functions when $k \leq n(1 - H(p) - \delta)$ by using the probabilistic method. In the end, we want our encoding and decoding functions to have error probability at most γ on *every* possible input. We begin with a weaker result, showing that there exist appropriate coding functions when the input is chosen uniformly at random from all k-bit inputs.

The encoding function assigns to each of the 2^k strings an n-bit *codeword* chosen independently and uniformly at random from the space of all n-bit sequences. Label these codewords $X_0, X_1, \dots, X_{2^k-1}$. The encoding function simply outputs the codeword assigned to the k-bit message using a large lookup table containing an entry for each k-bit string. (You may be concerned that two codewords may turn out to be the same; the probability of this is very small and is handled in the analysis that follows.)

To describe the decoding function, we provide a decoding algorithm based on the lookup table for the encoding function, which we may assume the receiver possesses. The decoding algorithm makes use of the fact that the receiver expects the channel to make roughly pn errors. The receiver therefore looks for a codeword that differs from the n bits received in between $(p - \varepsilon)n$ and $(p + \varepsilon)n$ places for some suitably small constant $\varepsilon > 0$. If just one codeword has this property, then the receiver will assume that this was the codeword sent and will recover the message accordingly. If more than one codeword has this property, the decoding algorithm fails. The decoding algorithm described here requires exponential time and space. As in the rest of this chapter, we are not now concerned with efficiency issues.

The corresponding (k, n) decoding function can be obtained from the algorithm by simply running through all possible n-bit sequences. Whenever a sequence decodes properly with the foregoing algorithm, the output of the decoding function for that sequence is set to the k-bit sequence associated with the corresponding codeword. Whenever the algorithm fails, the output for the sequence can be any arbitrary sequence of k bits. For the decoding function to fail, at least one of the two following events must occur:

- the channel does not make between $(p - \varepsilon)n$ and $(p + \varepsilon)n$ errors; or
- when a codeword is sent, the received sequence differs from some other codeword in between $(p - \varepsilon)n$ and $(p + \varepsilon)n$ places.

The path of the proof is now clear. A Chernoff bound can be used to show that, with high probability, the channel does not make too few or too many errors. Conditioning on the number of errors being neither too few nor too many, the question becomes how large k can be while ensuring that, with the required probability, the received sequence does not differ from multiple codewords in between $(p-\varepsilon)n$ and $(p+\varepsilon)n$ places.

Now that we have described the encoding and decoding functions, we establish the notation to be used in the analysis. Let R be the received sequence of bits. For sequences s_1 and s_2 of n bits, we write $\Delta(s_1, s_2)$ for the number of positions where these sequences differ. This value $\Delta(s_1, s_2)$ is referred to as the *Hamming distance* between the two strings. We say that the pair (s_1, s_2) has *weight*

$$w(s_1, s_2) = p^{\Delta(s_1,s_2)}(1-p)^{n-\Delta(s_1,s_2)}.$$

The weight corresponds to the probability that s_2 is received when s_1 is sent over the channel. We introduce random variables $S_0, S_1, \dots, S_{2^k-1}$ and $W_0, W_1, \dots, W_{2^k-1}$ defined as follows. The set S_i is the set of all received sequences that decode to X_i. The value W_i is given by

$$W_i = \sum_{r \notin S_i} w(X_i, r).$$

The S_i and W_i are random variables that depend only on the random choices of $X_0, X_1, \dots, X_{2^k-1}$. The variable W_i represents the probability that, when X_i is sent, the received sequence R does not lie in S_i and hence is decoded incorrectly. It is also helpful to express W_i in the following way: letting $I_{i,s}$ be an indicator random variable that is 1 if $s \notin S_i$ and 0 otherwise, we can write

$$W_i = \sum_r I_{i,r} w(X_i, r).$$

We start by bounding $\mathbf{E}[W_i]$. By symmetry, $\mathbf{E}[W_i]$ is the same for all i, so we bound $\mathbf{E}[W_0]$. Now

$$\begin{aligned}\mathbf{E}[W_0] &= \mathbf{E}\left[\sum_r I_{0,r} w(X_0, r)\right]\\ &= \sum_r \mathbf{E}[w(X_0, r) I_{0,r}].\end{aligned}$$

We split the sum into two parts. Let $T_1 = \{s : |\Delta(X_0, s) - pn| > \varepsilon n\}$ and $T_2 = \{s : |\Delta(X_0, s) - pn| \le \varepsilon n\}$, where $\varepsilon > 0$ is some constant to be determined. Then

$$\sum_r \mathbf{E}[w(X_0, r) I_{0,r}] = \sum_{r \in T_1} \mathbf{E}[w(X_0, r) I_{0,r}] + \sum_{r \in T_2} \mathbf{E}[w(X_0, r) I_{0,r}],$$

and we bound each term.

We first bound

$$\begin{aligned}\sum_{r\in T_1}\mathbf{E}[w(X_0,r)I_{0,r}] &\le \sum_{r\in T_1} w(X_0,r)\\ &= \sum_{r:|\Delta(X_0,r)-pn|>\varepsilon n} p^{\Delta(X_0,r)}(1-p)^{n-\Delta(X_0,r)}\\ &= \Pr(|\Delta(X_0,R)-np| > \varepsilon n).\end{aligned}$$

That is, to bound the first term, we simply bound the probability that the receiver fails to decode correctly and the number of errors is not in the range $[(p-\varepsilon)n, (p+\varepsilon)n]$ by the probability that the number of errors is not in this range. Equivalently, we obtain our bound by assuming that, whenever there are too many or too few errors introduced by the channel, we fail to decode correctly. This probability is very small, as we can see by using the Chernoff bound of Eqn. (4.6):

$$\Pr(|\Delta(X_0,R)-np| > \varepsilon n) \le 2\mathrm{e}^{-\varepsilon^2 n/3p}.$$

For any $\varepsilon > 0$, we can choose n sufficiently large so that this probability, and hence $\sum_{r\in T_1}\mathbf{E}[w(X_0,r)I_{0,r}]$, is less than $\gamma/2$.

We now find an upper bound for $\sum_{r\in T_2}\mathbf{E}[w(X_0,r)I_{0,r}]$. For every $r\in T_2$, the decoding algorithm will be successful when r is received unless r differs from some other codeword X_i in between $(p-\varepsilon)n$ and $(p+\varepsilon)n$ places. Hence $I_{0,r}$ will be 1 only if such an X_i exists, and thus for any values of X_0 and $r\in T_2$ we have

$$\begin{aligned}&\mathbf{E}[w(X_0,r)I_{0,r}]\\ &\quad = w(X_0,r)\Pr(\text{for some } X_i \text{ with } 1\le i\le 2^k-1,\ |\Delta(X_i,r)-pn|\le\varepsilon n).\end{aligned}$$

It follows that if we obtain an upper bound

$$\Pr(\text{for some } X_i \text{ with } 1\le i\le 2^k-1,\ |\Delta(X_i,r)-pn|\le\varepsilon n)\le\gamma/2$$

for any values of X_0 and $r\in T_2$, then

$$\sum_{r\in T_2}\mathbf{E}[w(X_0,r)I_{0,r}]\le\sum_{r\in T_2}w(X_0,r)\frac{\gamma}{2}\le\frac{\gamma}{2}.$$

To obtain this upper bound, we recall that the other codewords $X_1, X_2, \dots, X_{2^k-1}$ are chosen independently and uniformly at random. The probability that any other specific codeword X_i, $i>0$, differs from any given string r of length n in between $(p-\varepsilon)n$ and $(p+\varepsilon)n$ places is therefore at most

$$\sum_{j=\lceil n(p-\varepsilon)\rceil}^{\lfloor n(p+\varepsilon)\rfloor}\frac{\binom{n}{j}}{2^n}\le n\frac{\binom{n}{\lfloor n(p+\varepsilon)\rfloor}}{2^n}.$$

Here we have bounded the summation by n times its largest term; $\binom{n}{j}$ is largest when $j=\lfloor n(p+\varepsilon)\rfloor$ over the range of j in the summation, as long as ε is chosen so that $p+\varepsilon<1/2$.

Using Corollary 10.3,

$$\begin{aligned}\frac{\binom{n}{\lfloor n(p+\varepsilon)\rfloor}}{2^n} &\le \frac{2^{H(p+\varepsilon)n}}{2^n}\\ &= 2^{-n(1-H(p+\varepsilon))}.\end{aligned}$$

Hence the probability that any specific X_i matches a string r on a number of bits so as to cause a decoding failure is at most $n2^{-n(1-H(p+\varepsilon))}$. By a union bound, the probability that any of the $2^k - 1$ other codewords cause a decoding failure when X_0 is sent is at most

$$n2^{-n(1-H(p+\varepsilon))}(2^k - 1) \le n2^{n(H(p+\varepsilon)-H(p)-\delta)},$$

where we have used the fact that $k \le n(1 - H(p) - \delta)$. By choosing ε small enough so that $H(p+\varepsilon) - H(p) - \delta$ is negative and then choosing n sufficiently large, we can make this term as small as desired, and in particular we can make it less than $\gamma/2$.

By summing the bounds over the two sets T_1 and T_2, which correspond to the two types of error in the decoding algorithm, we find that $\mathbf{E}[W_0] \le \gamma$.

We can bootstrap this result to show that there exists a specific code such that, if the k-bit message to be sent is chosen uniformly at random, then the code fails with probability γ. We use the linearity of expectations and the probabilistic method. We have that

$$\sum_{j=0}^{2^k-1} \mathbf{E}[W_j] = \mathbf{E}\left[\sum_{j=0}^{2^k-1} W_j\right] \le 2^k\gamma,$$

where again the expectation is over the random choices of the codewords $X_0, X_1, \dots, X_{2^k-1}$. By the probabilistic method, there must exist a specific set of codewords $x_0, x_1, \dots, x_{2^k-1}$ such that

$$\sum_{j=0}^{2^k-1} W_j \le 2^k\gamma.$$

When a k-bit message to be sent is chosen uniformly at random, the probability of error is

$$\frac{1}{2^k}\sum_{j=0}^{2^k-1} W_j \le \gamma$$

for this set of codewords, proving the claim.

We now prove the stronger statement in the theorem: we can choose the codewords so that the probability of failure for each individual codeword is bounded above by γ. Notice that this is not implied by the previous analysis, which simply shows that the *average* probability of failure over the codewords is bounded above by γ.

We have shown that there exists a set of codewords $x_0, x_1, \dots, x_{2^k-1}$ for which

$$\sum_{j=0}^{2^k-1} W_j \le 2^k \gamma.$$

Without loss of generality, let us assume that the x_i are sorted in increasing order of W_i. Suppose that we remove the half of the codewords that have the largest values W_i; that is, we remove the codewords that have the highest probability of yielding an error when being sent. We claim that each x_i, $i < 2^{k-1}$, must satisfy $W_i \le 2\gamma$. Otherwise we would have

$$\sum_{j=2^{k-1}}^{2^k-1} W_j > 2^{k-1}(2\gamma) = 2^k\gamma,$$

a contradiction. (We used similar reasoning in the proof of Markov's inequality in Section 3.1.)

We can set up new encoding and decoding functions on all $(k-1)$-bit strings using just these 2^{k-1} codewords, and now the error probability for every codeword is simultaneously at most 2γ. Hence we have shown that, when $k - 1 \le n(1 - H(p) - \delta)$, there exists a code such that the probability that the receiver fails to obtain the correct message is at most 2γ for any message that is sent. Since δ and γ were arbitrary constants, we see that this implies the first half of the theorem.

Having completed the first half of the theorem, we now move to the second half: for any constants $\delta, \gamma > 0$ and for n sufficiently large, there do not exist (k, n) encoding and decoding functions with $k \ge n(1 - H(p) + \delta)$ such that the probability of decoding correctly is at least γ for a k-bit input message chosen uniformly at random.

Before giving the proof, let us first consider some helpful intuition. We know that the number of errors introduced by the channel is, with high probability, between $\lceil (p-\varepsilon)n \rceil$ and $\lfloor (p+\varepsilon)n \rfloor$ for a suitable constant $\varepsilon > 0$. Suppose that we try to set up the decoding function so that each codeword is decoded properly whenever the number of errors is between $(p-\varepsilon)n$ and $(p+\varepsilon)n$. Then each codeword is associated with

$$\sum_{k=\lceil n(p-\varepsilon)\rceil}^{\lfloor n(p+\varepsilon)\rfloor} \binom{n}{k} \ge \binom{n}{\lceil np \rceil} \ge \frac{2^{nH(p)}}{n+1}$$

bit sequences by the decoding function; the last inequality follows from Corollary 10.3. But there are 2^k different codewords, and

$$2^k \frac{2^{nH(p)}}{n+1} \ge 2^{n(1-H(p)+\delta)} \frac{2^{nH(p)}}{n+1} > 2^n$$

when n is sufficiently large. Since there are only 2^n possible bit sequences that can be received, we cannot create a decoding function that always decodes properly whenever the number of errors is between $(p-\varepsilon)n$ and $(p+\varepsilon)n$.

We now need to extend the argument for *any* encoding and decoding functions. This argument is more complex, since we cannot assume that the decoding function necessarily tries to decode properly whenever the number of errors is between $(p-\varepsilon)n$ and $(p+\varepsilon)n$, even though this would seem to be the best strategy to pursue.

Given any fixed encoding function with codewords $x_0, x_1, \ldots, x_{2^k-1}$ and any fixed decoding function, let z be the probability of successful decoding. Define S_i to be the set of all received sequences that decode to x_i. Then

$$\begin{aligned} z &= \sum_{i=0}^{2^k-1} \sum_{s \in S_i} \Pr((x_i \text{ is sent}) \cap (R = s)) \\ &= \sum_{i=0}^{2^k-1} \sum_{s \in S_i} \Pr(x_i \text{ is sent}) \Pr(R = s \mid x_i \text{ is sent}) \\ &= \frac{1}{2^k} \sum_{i=0}^{2^k-1} \sum_{s \in S_i} \Pr(R = s \mid x_i \text{ is sent}) \\ &= \frac{1}{2^k} \sum_{i=0}^{2^k-1} \sum_{s \in S_i} w(x_i, s). \end{aligned}$$

The second line follows from the definition of conditional probability. The third line uses the fact that the message sent and hence the codeword sent is chosen uniformly at random from all codewords. The fourth line is just the definition of the weight function.

To bound this last line, we again split the summation $\sum_{i=0}^{2^k-1} \sum_{s \in S_i} w(x_i, s)$ into two parts. Let $S_{i,1} = \{s \in S_i : |\Delta(x_i, s) - pn| > \varepsilon n\}$ and $S_{i,2} = \{s \in S_i : |\Delta(x_i, s) - pn| \leq \varepsilon n\}$, where again $\varepsilon > 0$ is some constant to be determined. Then

$$\sum_{s \in S_i} w(x_i, s) = \sum_{s \in S_{i,1}} w(x_i, s) + \sum_{s \in S_{i,2}} w(x_i, s).$$

Now

$$\sum_{s \in S_{i,1}} w(x_i, s) \leq \sum_{s : |\Delta(x_i, s) - pn| > \varepsilon n} w(x_i, s),$$

which can be bounded using Chernoff bounds. The summation on the right is simply the probability that the number of errors introduced by the channel is not between $(p-\varepsilon)n$ and $(p+\varepsilon)n$, which we know from previous arguments is at most $2e^{-\varepsilon^2 n/3p}$. This bound is equivalent to assuming that decoding is successful even if there are too many or too few errors introduced by the channel; since the probability of too many or too few errors is small, this assumption still yields a good bound.

To bound $\sum_{s \in S_{i,2}} w(x_i, s)$, we note that $w(x_i, s)$ is decreasing in $\Delta(x_i, s)$. Hence, for $s \in S_{i,2}$,

$$\begin{aligned}
w(x_i, s) &\le p^{(p-\varepsilon)n}(1-p)^{(1-p+\varepsilon)n} \\
&= p^{pn}(1-p)^{(1-p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n} \\
&= 2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sum_{s \in S_{i,2}} w(x_i, s) &\le \sum_{s \in S_{i,2}} 2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n} \\
&= 2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n} |S_{i,2}|.
\end{aligned}$$

We continue with

$$\begin{aligned}
z &= \frac{1}{2^k}\sum_{i=0}^{2^k-1}\sum_{s \in S_i} w(x_i, s) \\
&= \frac{1}{2^k}\sum_{i=0}^{2^k-1}\left(\sum_{s \in S_{i,1}} w(x_i, s) + \sum_{s \in S_{i,2}} w(x_i, s)\right) \\
&\le \frac{1}{2^k}\sum_{i=0}^{2^k-1}\left(2e^{-\varepsilon^2 n/3p} + 2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n} |S_{i,2}|\right) \\
&= 2e^{-\varepsilon^2 n/3p} + \frac{1}{2^k}2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n}\sum_{i=0}^{2^k-1}|S_{i,2}| \\
&\le 2e^{-\varepsilon^2 n/3p} + \frac{1}{2^k}2^{-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n} 2^n.
\end{aligned}$$

In this last line, we have used the important fact that the sets of bit sequences S_i and hence all the $S_{i,2}$ are disjoint, so their total size is at most 2^n. This is where the fact that we are using a decoding function comes into play, allowing us to establish a useful bound.

To conclude,

$$\begin{aligned}
z &\le 2e^{-\varepsilon^2 n/3p} + 2^{n-(1-H(p)+\delta)n-H(p)n}\left(\frac{1-p}{p}\right)^{\varepsilon n} \\
&= 2e^{-\varepsilon^2 n/3p} + \left(\left(\frac{1-p}{p}\right)^{\varepsilon} 2^{-\delta}\right)^n.
\end{aligned}$$

As long as we choose ε sufficiently small that

$$\left(\frac{1-p}{p}\right)^{\varepsilon} 2^{-\delta} < 1,$$

then, when n is sufficiently large, $z < \gamma$, which proves Theorem 10.8. ■

Shannon's theorem demonstrates the existence of codes that transmit arbitrarily closely to the capacity of the binary symmetric channel over long enough blocks. It does not give explicit codes, nor does it say that such codes can be encoded and decoded efficiently. It took decades after Shannon's original work before practical codes with near-optimal performance were found.

10.6. Exercises

Exercise 10.1: **(a)** Let $S = \sum_{k=1}^{10} 1/k^2$. Consider a random variable X such that $\Pr(X = k) = 1/Sk^2$ for integers $k = 1, \ldots, 10$. Find $H(X)$.

(b) Let $S = \sum_{k=1}^{10} 1/k^3$. Consider a random variable X such that $\Pr(X = k) = 1/Sk^3$ for integers $k = 1, \ldots, 10$. Find $H(X)$.

(c) Consider $S_\alpha = \sum_{k=1}^{10} 1/k^\alpha$, where $\alpha > 1$ is a constant. Consider random variables X_α such that $\Pr(X_\alpha = k) = 1/S_\alpha k^\alpha$ for integers $k = 1, \ldots, 10$. Give an intuitive explanation explaining whether $H(X_\alpha)$ is increasing or decreasing with α and why.

Exercise 10.2: Consider an n-sided die, where the ith face comes up with probability p_i. Show that the entropy of a die roll is maximized when each face comes up with equal probability $1/n$.

Exercise 10.3: **(a)** A fair coin is repeatedly flipped until the first heads occurs. Let X be the number of flips required. Find $H(X)$.

(b) Your friend flips a fair coin repeatedly until the first heads occurs. You want to determine how many flips were required. You are allowed to ask a series of yes–no questions of the following form: you give your friend a set of integers, and your friend answers "yes" if the number of flips is in that set and "no" otherwise. Describe a strategy so that the expected number of questions you must ask before determining the number of flips is $H(X)$.

(c) Give an intuitive explanation of why you cannot come up with a strategy that would allow you to ask fewer than $H(X)$ questions on average.

Exercise 10.4: **(a)** Show that

$$S = \sum_{k=2}^{\infty} \frac{1}{k \ln^2 k}$$

is finite.

(b) Consider the integer-valued discrete random variable X given by

$$\Pr(X = k) = \frac{1}{Sk\ln^2 k}, \qquad k \geq 2.$$

Show that $H(X)$ is unbounded.

Exercise 10.5: Suppose p is chosen uniformly at random from the real interval $[0, 1]$. Calculate $\mathbf{E}[H(p)]$.

Exercise 10.6: The *conditional entropy* $H(Y \mid X)$ is defined by

$$H(Y \mid X) = \sum_{x,y} \Pr((X = x) \cap (Y = y)) \log_2 \Pr(Y = y \mid X = x).$$

If $Z = (X, Y)$, show that

$$H(Z) = H(X) + H(Y \mid X).$$

Exercise 10.7: One form of Stirling's formula is

$$\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n < n! < \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n \mathrm{e}^{1/(12n)}.$$

Using this, prove

$$\binom{n}{qn} \geq \frac{2^{nH(q)}}{2\sqrt{n}},$$

which is a tighter bound than that of Lemma 10.2.

Exercise 10.8: We have shown in Theorem 10.5 that we can use a recursive procedure to extract, on average, at least $\lfloor \log_2 m \rfloor - 1$ independent, unbiased bits from a number X chosen uniformly at random from $S = \{0, \dots, m - 1\}$. Consider the following extraction function: let $\alpha = \lfloor \log_2 m \rfloor$, and write

$$m = \beta_\alpha 2^\alpha + \beta_{\alpha-1} 2^{\alpha-1} + \cdots + \beta_0 2^0,$$

where each β_i is either 0 or 1.

Let k be the number of values of i for which β_i equals 1. Then we split S into k disjoint subsets in the following manner: there is one set for each value of β_i that equals 1, and the set for this i has 2^i elements. The assignment of S to sets can be arbitrary, as long as the resulting sets are disjoint. To get an extraction function, we map the elements of the subset with 2^i elements in a one-to-one manner with the 2^i binary strings of length i.

Show that this mapping is equivalent to the recursive extraction procedure given in Theorem 10.5 in that both produce i bits with the same probability for all i.

Exercise 10.9: We have shown that we can extract, on average, at least $\lfloor \log_2 m \rfloor - 1$ independent, unbiased bits from a number chosen uniformly at random from

$\{0, \ldots, m-1\}$. It follows that if we have k numbers chosen independently and uniformly at random from $\{0, \ldots, m-1\}$ then we can extract, on average, at least $k\lfloor \log_2 m \rfloor - k$ independent, unbiased bits from them. Give a better procedure that extracts, on average, at least $k\lfloor \log_2 m \rfloor - 1$ independent, unbiased bits from these numbers.

Exercise 10.10: Suppose that we have a means of generating independent, fair coin flips.

(a) Give an algorithm using the coin to generate a number uniformly from $\{0, 1, \ldots, n-1\}$, where n is a power of 2, using exactly $\log_2 n$ flips.
(b) Argue that, if n is *not* a power of 2, then no algorithm can generate a number uniformly from $\{0, 1, \ldots, n-1\}$ using exactly k coin flips for any fixed k.
(c) Argue that, if n is not a power of 2, then no algorithm can generate a number uniformly from $\{0, 1, \ldots, n-1\}$ using *at most* k coin flips for any fixed k.
(d) Give an algorithm using the coin to generate a number uniformly from $\{0, 1, \ldots, n-1\}$, even when n is not a power of 2, using at most $2\lceil \log_2 n \rceil$ *expected* flips.

Exercise 10.11: Suppose that we have a means of generating independent, fair coin flips.

(a) Give an algorithm using the fair coin that simulates flipping a biased coin that comes up heads with probability p. The expected number of flips your algorithm uses should be at most 2. (*Hint:* Think of p written as a decimal in binary, and use the fair coin to generate binary decimal digits.)
(b) Give an algorithm using the coin to generate a number uniformly from $\{0, 1, \ldots, n-1\}$. The expected number of flips your algorithm uses should be at most $\lceil \log_2 n \rceil + 2$.

Exercise 10.12: Here is an extraction algorithm $\mathcal{A}$ whose input is a sequence $X = x_1, x_2, \ldots, x_n$ of n independent flips of a coin that comes up heads with probability $p > 1/2$. Break the sequence into $\lfloor n/2 \rfloor$ pairs, $a_i = (x_{2i-1}, x_{2i})$ for $i = 1, \ldots, \lfloor n/2 \rfloor$. Consider the pairs in order. If $y_i =$ (heads, tails) then output a 0; if $a_i =$ (tails, heads) then output a 1; otherwise, move on to the next pair.

(a) Show that the bits extracted are independent and unbiased.
(b) Show that the expected number of extracted bits is $\lfloor n/2 \rfloor 2p(1-p) \approx np(1-p)$.
(c) We can derive another set of flips $Y = y_1, y_2, \ldots$ from the sequence X as follows. Start with $j, k = 1$. Repeat the following operations until $j = \lfloor n/2 \rfloor$: If $a_j =$ (heads, heads), set y_k to heads and increment j and k; if $a_j =$ (tails, tails), set y_k to tails and increment j and k; otherwise, increment j. See Figure 10.3 for an example.

The intuition here is that we take some of the randomness that $\mathcal{A}$ was unable to use effectively and re-use it. Show that the bits produced by running $\mathcal{A}$ on Y

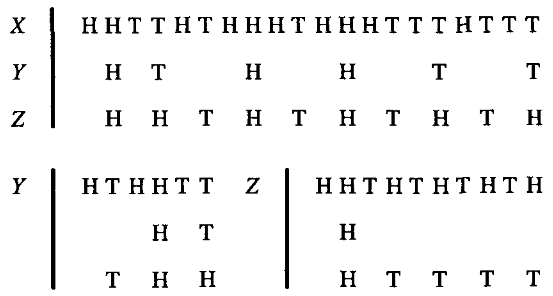

Figure 10.3: After running $\mathcal{A}$ on the input sequence X, we can derive further sequences Y and Z; after running $\mathcal{A}$ on each of Y and Z, we can derive further sequences from them; and so on.

are independent and unbiased, and further argue that they are independent of those produced from running $\mathcal{A}$ on X.

(d) We can derive a second set of flips $Z = z_1, z_2, \ldots, z_{\lfloor n/2 \rfloor}$ from the sequence X as follows: let z_i be heads if $a_i =$ (heads, heads) or (tails, tails), and let z_i be tails otherwise. See Figure 10.3 for an example. Show that the bits produced by running $\mathcal{A}$ on Z are independent and unbiased, and further argue that they are independent of those produced from running $\mathcal{A}$ on X and Y.

(e) After we derive and run $\mathcal{A}$ on Y and Z, we can recursively derive two further sequences from each of these sequences in the same way, run $\mathcal{A}$ on those, and so on. See Figure 10.3 for an example. Let $A(p)$ be the average number of bits extracted for each flip (with probability p of coming up heads) in the sequence X, in the limit as the length of the sequence X goes to infinity. Argue that $A(p)$ satisfies the recurrence

$$A(p) = p(1-p) + \frac{1}{2}(p^2 + q^2)\,A\left(\frac{p^2}{p^2 + q^2}\right) + \frac{1}{2}A(p^2 + (1-p)^2).$$

(f) Show that the entropy function $H(p)$ satisfies this recurrence for $A(p)$.

(g) Implement the recursive extraction procedure explained in part (e). Run it 1000 times on sequences of 1024 bits generated by a coin that comes up heads with probability $p = 0.7$. Give the distribution of the number of flips extracted over the 1000 runs and discuss how close your results are to $1024 \cdot H(0.7)$.

Exercise 10.13: Suppose that, instead of a biased coin, we have a biased six-sided die with entropy $h > 0$. Modify our extraction function for the case of biased coins so that it extracts, on average, almost h random bits per roll from a sequence of die rolls. Prove formally that your extraction function works by modifying Theorem 10.5 appropriately.

Exercise 10.14: Suppose that, instead of a biased coin, we have a biased six-sided die with entropy $h > 0$. Modify our compression function for the case of biased coins so that it compresses a sequence of n die rolls to almost nh bits on average. Prove formally that your compression function works by modifying Theorem 10.6 appropriately.

Exercise 10.15: We wish to compress a sequence of independent, identically distributed random variables $X_1, X_2, \ldots$. Each X_j takes on one of n values. We map the ith value to a codeword, which is a sequence of ℓ_i bits. We wish these codewords to have the property that no codeword is the prefix of any other codeword.

(a) Explain how this property can be used to easily decompress the string created by the compression algorithm when reading the bits sequentially.

(b) Prove that the ℓ_i must satisfy

$$\sum_{i=1}^{n} 2^{-\ell_i} \leq 1.$$

This is known as the Kraft inequality.

Exercise 10.16: We wish to compress a sequence of independent, identically distributed random variables $X_1, X_2, \ldots$. Each X_j takes on one of n values. The ith value occurs with probability p_i, where $p_1 \geq p_2 \geq \cdots \geq p_n$. The result is compressed as follows. Set

$$T_i = \sum_{j=1}^{i-1} p_j,$$

and let the ith codeword be the first $\lceil \log_2(1/p_i) \rceil$ bits of T_i. Start with an empty string, and consider the X_j in order. If X_j takes on the ith value, append the ith codeword to the end of the string.

(a) Show that no codeword is the prefix of any other codeword.

(b) Let z be the average number of bits appended for each random variable X_j. Show that

$$H(X) \leq z \leq H(X) + 1.$$

Exercise 10.17: *Arithmetic coding* is a standard compression method. In the case where the string to be compressed is a sequence of biased coin flips, it can be described as follows. Suppose that we have a sequence of bits $X = (X_1, X_2, \ldots, X_n)$, where each X_i is independently 0 with probability p and 1 with probability $1 - p$. The sequences can be ordered lexicographically, so for $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ we say $x < y$ if $x_i = 0$ and $y_i = 1$ in the first coordinate i such that $x_i \neq y_i$. If z_x is the number of zeroes in the string x, then define $p(x) = p^{z_x}(1-p)^{n-z_x}$ and $q(x) = \sum_{y<x} p(y)$.

(a) Suppose we are given $X = (X_1, X_2, \ldots, X_n)$ sequentially. Explain how to compute $q(X)$ in time $O(n)$. (You may assume that any operation on real numbers takes constant time.)

(b) Argue that the intervals $[q(x), q(x) + p(x))$ are disjoint subintervals of $[0, 1)$.

(c) Given (a) and (b), the sequence X can be represented by any point in the interval $[q(X), q(X) + p(X))$. Show that we can choose a codeword in $[q(X), q(X) +$

$p(X)$) with $\lceil \log_2(1/p(X)) \rceil + 1$ binary decimal digits to represent X in such a way that no codeword is the prefix of any other codeword.

(d) Given a codeword chosen as in (c), explain how to decompress it to determine the corresponding sequence $(X_1, X_2, \ldots, X_n)$.

(e) Using a Chernoff bound, argue that $\log_2(1/p(X))$ is close to $nH(p)$ with high probability. Hence this approach yields an effective compression scheme.

Exercise 10.18: Alice wants to send Bob the result of a fair coin flip over a binary symmetric channel that flips each bit with probability $p < 1/2$. To avoid errors in transmission, she encodes heads as a sequence of $2k + 1$ zeroes and tails as a sequence of $2k + 1$ ones.

(a) Consider the case where $k = 1$, so heads is encoded as 000 and tails as 111. For each of the eight possible sequences of 3 bits that can be received, determine the probability that Alice flipped a heads conditioned on Bob receiving that sequence.

(b) Bob decodes by examining the 3 bits. If two or three of the bits are 0, then Bob decides the corresponding coin flip was a heads. Prove that this rule minimizes the probability of error for each flip.

(c) Argue that, for general k, Bob minimized the probability of error by deciding the flip was heads if at least $k + 1$ of the bits are 0.

(d) Give a formula for the probability that Bob makes an error that holds for general k. Evaluate the formula for $p = 0.1$ and k ranging from 1 to 6.

(e) Give a bound on the probability computed in part (d) using Chernoff bounds.

Exercise 10.19: Consider the following channel. The sender can send a symbol from the set $\{0, 1, 2, 3, 4\}$. The channel introduces errors; when the symbol k is sent, the recipient receives $k + 1 \bmod 5$ with probability $1/2$ and receives $k - 1 \bmod 5$ with probability $1/2$. The errors are mutually independent when multiple symbols are sent.

Let us define encoding and decoding functions for this channel. A (j, n) encoding function Enc maps a number in $\{0, 1, \ldots, j - 1\}$ into sequences from $\{0, 1, 2, 3, 4\}^n$, and a (j, n) decoding function Dec maps sequences from $\{0, 1, 2, 3, 4\}^n$ back into $\{0, 1, \ldots, j - 1\}$. Notice that this definition is slightly different than the one we used for bit sequences over the binary symmetric channel.

There are $(1, 1)$ encoding and decoding functions with zero probability of error. The encoding function maps 0 to 0 and 1 to 1. When a 0 is sent, the receiver will receive either a 1 or 4, so the decoding function maps 1 and 4 back to 0. When a 1 is sent, the receiver will receiver either a 2 or 0, so the decoding function maps 2 and 0 back to 1. This guarantees that no error is made. Hence at least one bit can be sent without error per channel use.

(a) Show that there are $(5, 2)$ encoding and decoding functions with zero probability of error. Argue that this means more than one bit of information can be sent per use of the channel.

(b) Show that if there are (j, n) encoding and decoding functions with zero probability of error, then $n \geq \log_2 j/(\log_2 5 - 1)$.

Exercise 10.20: A *binary erasure channel* transfers a sequence of n bits. Each bit either arrives successfully without error or fails to arrive successfully and is replaced by a '?' symbol, denoting that it is not known if that bit is a 0 or a 1. Failures occur independently with probability p. We can define (k, n) encoding and decoding functions for the binary erasure channel in a similar manner as for the binary symmetric channel, except here the decoding function Dec: $\{0, 1, ?\}^n \rightarrow \{0, 1\}^k$ must handle sequences with the '?' symbol.

Prove that, for any $p > 0$ and any constants $\delta, \gamma > 0$, if n is sufficiently large then there exist (k, n) encoding and decoding functions with $k \leq n(1 - p - \delta)$ such that the probability that the receiver fails to obtain the correct message is at most γ for every possible k-bit input message.

Exercise 10.21: In proving Shannon's theorem, we used the following decoding method: Look for a codeword that differs from the received sequence of bits in between $(p - \varepsilon)n$ and $(p + \varepsilon)n$ places, for an appropriate choice of ε; if there is only one such codeword, the decoder concludes that that codeword was the one sent. Suppose instead that the decoder looks for the codeword that differs from the received sequence in the smallest number of bits (breaking ties arbitrarily), and concludes that that codeword was the one sent. Show how to modify the proof of Shannon's theorem for this decoding technique to obtain a similar result.