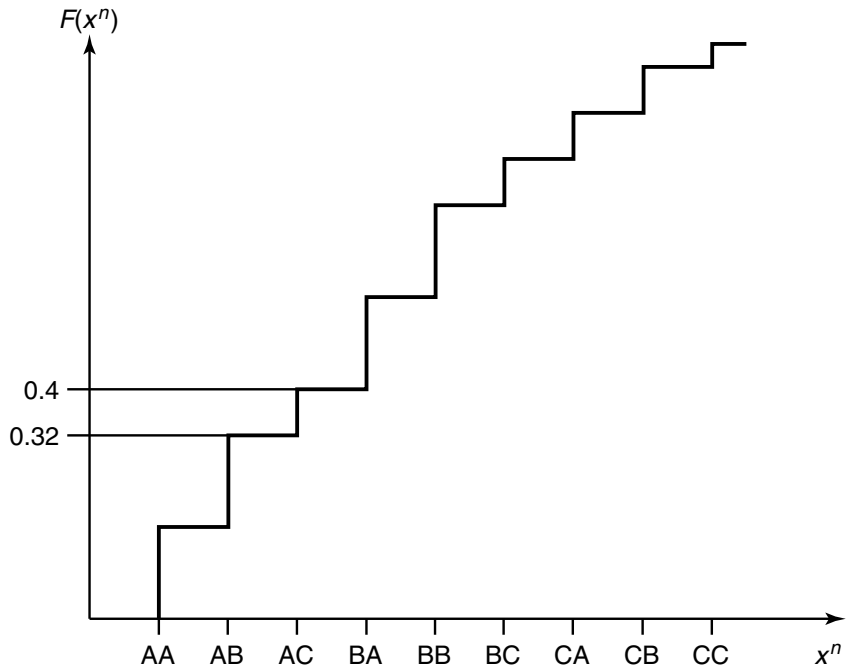

FIGURE 13.3. Cumulative distribution function after the second symbol.

[0, 0.4); after the second symbol, C, it is [0.32, 0.4) (Figure 13.3); after the third symbol A, it is [0.32,0.352); and after the fourth symbol A, it is [0.32, 0.3328). Since the probability of this sequence is 0.0128, we will use $\log(1/0.0128) + 2$ (i.e., 9 bits) to encode the midpoint of the interval sequence using Shannon–Fano–Elias coding (0.3264, which is 0.010100111 binary).

In summary, the arithmetic coding procedure, given any length n and probability mass function $q(x_1x_2 \cdots x_n)$, enables one to encode the sequence $x_1x_2 \cdots x_n$ in a code of length $\log \frac{1}{q(x_1x_2 \cdots x_n)} + 2$ bits. If the source is i.i.d. and the assumed distribution q is equal to the true distribution p of the data, this procedure achieves an average length for the block that is within 2 bits of the entropy. Although this is not necessarily optimal for any fixed block length (a Huffman code designed for the distribution could have a lower average codeword length), the procedure is incremental and can be used for any blocklength.

13.4 LEMPEL–ZIV CODING

In Section 13.3 we discussed the basic ideas of arithmetic coding and mentioned some results on worst-case redundancy for coding a sequence from an unknown distribution. We now discuss a popular class of techniques for source coding that are universally optimal (their asymptotic

compression rate approaches the entropy rate of the source for any stationary ergodic source) and simple to implement. This class of algorithms is termed Lempel–Ziv, named after the authors of two seminal papers [603, 604] that describe the two basic algorithms that underlie this class. The algorithms could also be described as adaptive dictionary compression algorithms.

The notion of using dictionaries for compression dates back to the invention of the telegraph. At the time, companies were charged by the number of letters used, and many large companies produced codebooks for the frequently used phrases and used the codewords for their telegraphic communication. Another example is the notion of greetings telegrams that are popular in India—there is a set of standard greetings such as "25:Merry Christmas" and "26:May Heaven's choicest blessings be showered on the newly married couple." A person wishing to send a greeting only needs to specify the number, which is used to generate the actual greeting at the destination.

The idea of adaptive dictionary-based schemes was not explored until Ziv and Lempel wrote their papers in 1977 and 1978. The two papers describe two distinct versions of the algorithm. We refer to these versions as LZ77 or sliding window Lempel–Ziv and LZ78 or tree-structured Lempel–Ziv. (They are sometimes called LZ1 and LZ2, respectively.)

We first describe the basic algorithms in the two cases and describe some simple variations. We later prove their optimality, and end with some practical issues. The key idea of the Lempel–Ziv algorithm is to parse the string into phrases and to replace phrases by pointers to where the same string has occurred in the past. The differences between the algorithms is based on differences in the set of possible match locations (and match lengths) the algorithm allows.

13.4.1 Sliding Window Lempel–Ziv Algorithm

The algorithm described in the 1977 paper encodes a string by finding the longest match anywhere within a window of past symbols and represents the string by a pointer to location of the match within the window and the length of the match. There are many variations of this basic algorithm, and we describe one due to Storer and Szymanski [507].

We assume that we have a string $x_1, x_2, \ldots$ to be compressed from a finite alphabet. A *parsing* S of a string $x_1 x_2 \cdots x_n$ is a division of the string into phrases, separated by commas. Let W be the length of the window. Then the algorithm can be described as follows: Assume that we have compressed the string until time $i-1$. Then to find the next phrase, find the largest k such that for some j, $i-1-W \leq j \leq i-1$,

the string of length k starting at x_j is equal to the string (of length k) starting at x_i (i.e., $x_{j+l} = x_{i+l}$ for all $0 \leq l < k$). The next phrase is then of length k (i.e., $x_i \ldots x_{i+k-1}$) and is represented by the pair (P, L), where P is the location of the beginning of the match and L is the length of the match. If a match is not found in the window, the next character is sent uncompressed. To distinguish between these two cases, a flag bit is needed, and hence the phrases are of two types: (F, P, L) or (F, C), where C represents an uncompressed character.

Note that the target of a (pointer,length) pair could extend beyond the window, so that it overlaps with the new phrase. In theory, this match could be arbitrarily long; in practice, though, the maximum phrase length is restricted to be less than some parameter.

For example, if $W = 4$ and the string is ABBABBABBBAABABA and the initial window is empty, the string will be parsed as follows: A,B,B,ABBABB,BA,A,BA,BA, which is represented by the sequence of "pointers": (0,A),(0,B),(1,1,1),(1,3,6),(1,4,2),(1,1,1),(1,3,2),(1,2,2), where the flag bit is 0 if there is no match and 1 if there is a match, and the location of the match is measured backward from the end of the window. [In the example, we have represented every match within the window using the (P, L) pair; however, it might be more efficient to represent short matches as uncompressed characters. See Problem 13.8 for details.]

We can view this algorithm as using a dictionary that consists of all substrings of the string in the window and of all single characters. The algorithm finds the longest match within the dictionary and sends a pointer to that match. We later show that a simple variation on this version of LZ77 is asymptotically optimal. Most practical implementations of LZ77, such as gzip and pkzip, are also based on this version of LZ77.

13.4.2 Tree-Structured Lempel–Ziv Algorithms

In the 1978 paper, Ziv and Lempel described an algorithm that parses a string into phrases, where each phrase is the shortest phrase not seen earlier. This algorithm can be viewed as building a dictionary in the form of a tree, where the nodes correspond to phrases seen so far. The algorithm is particularly simple to implement and has become popular as one of the early standard algorithms for file compression on computers because of its speed and efficiency. It is also used for data compression in high-speed modems.

The source sequence is sequentially parsed into strings that have not appeared so far. For example, if the string is ABBABBABBBAABABAA . . . , we parse it as A,B,BA,BB,AB,BBA,ABA,BAA . . . . After every comma, we look along the input sequence until we come to the shortest string that has not been marked off before. Since this is the shortest such string,

all its prefixes must have occurred earlier. (Thus, we can build up a tree of these phrases.) In particular, the string consisting of all but the last bit of this string must have occurred earlier. We code this phrase by giving the location of the prefix and the value of the last symbol. Thus, the string above would be represented as (0,A),(0,B),(2,A),(2,B),(1,B),(4,A),(5,A), (3,A), ...

Sending an uncompressed character in each phrase results in a loss of efficiency. It is possible to get around this by considering the extension character (the last character of the current phrase) as part of the next phrase. This variation, due to Welch [554], is the basis of most practical implementations of LZ78, such as compress on Unix, in compression in modems, and in the image files in the GIF format.

13.5 OPTIMALITY OF LEMPEL–ZIV ALGORITHMS

13.5.1 Sliding Window Lempel–Ziv Algorithms

In the original paper of Ziv and Lempel [603], the authors described the basic LZ77 algorithm and proved that it compressed any string as well as any finite-state compressor acting on that string. However, they did not prove that this algorithm achieved asymptotic optimality (i.e., that the compression ratio converged to the entropy for an ergodic source). This result was proved by Wyner and Ziv [591].

The proof relies on a simple lemma due to Kac: the average length of time that you need to wait to see a particular symbol is the reciprocal of the probability of a symbol. Thus, we are likely to see the high-probability strings within the window and encode these strings efficiently. The strings that we do not find within the window have low probability, so that asymptotically, they do not influence the compression achieved.

Instead of proving the optimality of the practical version of LZ77, we will present a simpler proof for a different version of the algorithm, which, though not practical, captures some of the basic ideas. This algorithm assumes that both the sender and receiver have access to the infinite past of the string, and represents a string of length n by pointing to the last time it occurred in the past.

We assume that we have a stationary and ergodic process defined for time from $-\infty$ to ∞, and that both the encoder and decoder have access to $\ldots, X_{-2}, X_{-1}$, the infinite past of the sequence. Then to encode $X_0, X_1, \ldots, X_{n-1}$ (a block of length n), we find the last time we have seen these n symbols in the past. Let

$$R_n(X_0, X_1, \ldots, X_{n-1}) = \max\{j < 0 : (X_{-j}, X_{-j+1} \ldots X_{-j+n-1}) = (X_0, \ldots, X_{n-1})\}. \quad (13.60)$$

Then to represent $X_0, \ldots, X_{n-1}$, we need only to send R_n to the receiver, who can then look back R_n bits into the past and recover $X_0, \ldots, X_{n-1}$. Thus, the cost of the encoding is the cost of representing R_n. We will show that this cost is approximately $\log R_n$ and that asymptotically $\frac{1}{n} E \log R_n \rightarrow H(\mathcal{X})$, thus proving the asymptotic optimality of this algorithm.

We will need the following lemmas.

Lemma 13.5.1 *There exists a prefix-free code for the integers such that the length of the codeword for integer k is* $\log k + 2 \log \log k + O(1)$.

Proof: If we knew that $k \leq m$, we could encode k with $\log m$ bits. However, since we don't have an upper limit for k, we need to tell the receiver the length of the encoding of k (i.e., we need to specify $\log k$). Consider the following encoding for the integer k: We first represent $\lceil \log k \rceil$ in unary, followed by the binary representation of k:

$$C_1(k) = \underbrace{00\cdots0}_{\lceil \log k \rceil \text{ 0's}} \; 1 \; \underbrace{x x \cdots x}_{k \text{ in binary}} \quad . \tag{13.61}$$

It is easy to see that the length of this representation is $2\lceil \log k \rceil + 1 \leq 2 \log k + 3$. This is more than the length we are looking for since we are using the very inefficient unary code to send $\log k$. However, if we use C_1 to represent $\log k$, it is now easy to see that this representation has a length less than $\log k + 2 \log \log k + 4$, which proves the lemma. A similar method is presented in the discussion following Theorem 14.2.3. □

The key result that underlies the proof of the optimality of LZ77 is Kac's lemma, which relates the average recurrence time to the probability of a symbol for any stationary ergodic process. For example, if $X_1, X_2, \ldots, X_n$ is an i.i.d. process, we ask what is the expected waiting time to see the symbol a again, conditioned on the fact that $X_1 = a$. In this case, the waiting time has a geometric distribution with parameter $p = p(X_0 = a)$, and thus the expected waiting time is $1/p(X_0 = a)$. The somewhat surprising result is that the same is true even if the process is not i.i.d., but stationary and ergodic. A simple intuitive reason for this is that in a long sample of length n, we would expect to see a about $np(a)$ times, and the average distance between these occurrences of a is $n/(np(a))$ (i.e., $1/p(a)$).

Lemma 13.5.2 (*Kac*) *Let $\ldots, U_2, U_1, U_0, U_1, \ldots$ be a stationary ergodic process on a countable alphabet. For any u such that $p(u) > 0$*

and for $i = 1, 2, \ldots,$ *let*

$$Q_u(i) = \Pr\left\{U_{-i} = u; U_j \neq u \text{ for } -i < j < 0 | U_0 = u\right\} \tag{13.62}$$

[i.e., $Q_u(i)$ *is the conditional probability that the most recent previous occurrence of the symbol* u *is* i*, given that* $U_0 = u$*]. Then*

$$E(R_1(U)|X_0 = u) = \sum_i i\, Q_u(i) = \frac{1}{p(u)}. \tag{13.63}$$

Thus, the conditional expected waiting time to see the symbol u *again, looking backward from zero, is* $1/p(u)$*.*

Note the amusing fact that the expected recurrence time

$$E\,R_1(U) = \sum p(u)\frac{1}{p(u)} = m, \tag{13.64}$$

where m is the alphabet size.

Proof: Let $U_0 = u$. Define the events for $j = 1, 2, \ldots$ and $k = 0, 1, 2, \ldots$:

$$A_{jk} = \left\{U_{-j} = u,\ U_l \neq u, -j < l < k,\ U_k = u\right\}. \tag{13.65}$$

Event A_{jk} corresponds to the event where the last time before zero at which the process is equal to u is at $-j$, the first time after zero at which the process equals u is k. These events are disjoint, and by ergodicity, the probability $\Pr\{\cup_{j,k} A_{jk}\} = 1$. Thus,

$$1 = \Pr\left\{\cup_{j,k} A_{jk}\right\} \tag{13.66}$$

$$\stackrel{(a)}{=} \sum_{j=1}^{\infty}\sum_{k=0}^{\infty} \Pr\{A_{jk}\} \tag{13.67}$$

$$= \sum_{j=1}^{\infty}\sum_{k=0}^{\infty} \Pr(U_k = u)\Pr\left\{U_{-j} = u, U_l \neq u, -j < l < k | U_k = u\right\} \tag{13.68}$$

$$\stackrel{(b)}{=} \sum_{j=1}^{\infty}\sum_{k=0}^{\infty} \Pr(U_k = u) Q_u(j+k) \tag{13.69}$$

$$\stackrel{(c)}{=} \sum_{j=1}^{\infty}\sum_{k=0}^{\infty} \Pr(U_0 = u)\, Q_u(j+k) \tag{13.70}$$

$$= \Pr(U_0 = u) \sum_{j=1}^{\infty}\sum_{k=0}^{\infty} Q_u(j+k) \tag{13.71}$$

$$\stackrel{(d)}{=} \Pr(U_0 = u) \sum_{i=1}^{\infty} i\, Q_u(i), \tag{13.72}$$

where (a) follows from the fact that the A_{jk} are disjoint, (b) follows from the definition of $Q_u(\cdot)$, (c) follows from stationarity, and (d) follows from the fact that there are i pairs (j, k) such that $j + k = i$ in the sum. Kac's lemma follows directly from this equation. □

Corollary *Let* $\ldots, X_{-1}, X_0, X_1, \ldots$ *be a stationary ergodic process and let* $R_n(X_0, \ldots, X_{n-1})$ *be the recurrence time looking backward as defined in (13.60). Then*

$$E\Big[R_n(X_0, \ldots, X_{n-1}) | (X_0, \ldots, X_{n-1}) = x_0^{n-1}\Big] = \frac{1}{p(x_0^{n-1})}. \tag{13.73}$$

Proof: Define a new process with $U_i = (X_i, X_{i+1}, \ldots, X_{i+n-1})$. The U process is also stationary and ergodic, and thus by Kac's lemma the average recurrence time for U conditioned on $U_0 = u$ is $1/p(u)$. Translating this to the X process proves the corollary. □

We are now in a position to prove the main result, which shows that the compression ratio for the simple version of Lempel–Ziv using recurrence time approaches the entropy. The algorithm describes X_0^{n-1} by describing $R_n(X_0^{n-1})$, which by Lemma 13.5.1 can be done with $\log R_n + 2\log\log R_n + 4$ bits. We now prove the following theorem.

Theorem 13.5.1 *Let* $L_n(X_0^{n-1}) = \log R_n + 2\log\log R_n + O(1)$ *be the description length for* X_0^{n-1} *in the simple algorithm described above. Then*

$$\frac{1}{n} E L_n(X_0^{n-1}) \to H(\mathcal{X}) \tag{13.74}$$

as $n \to \infty$, *where* $H(\mathcal{X})$ *is the entropy rate of the process* $\{X_i\}$.

Proof: We will prove upper and lower bounds for EL_n. The lower bound follows directly from standard source coding results (i.e., $EL_n \geq nH$ for any prefix-free code). To prove the upper bound, we first show that

$$\overline{\lim} \frac{1}{n} E \log R_n \leq H \tag{13.75}$$

and later bound the other terms in the expression for L_n. To prove the bound for $E \log R_n$, we expand the expectation by conditioning on the value of X_0^{n-1} and then applying Jensen's inequality. Thus,

$$\frac{1}{n} E \log R_n = \frac{1}{n} \sum_{x_0^{n-1}} p(x_0^{n-1}) E[\log R_n(X_0^{n-1}) | X_0^{n-1} = x_0^{n-1}] \tag{13.76}$$

$$\leq \frac{1}{n} \sum_{x_0^{n-1}} p(x_0^{n-1}) \log E[R_n(X_0^{n-1}) | X_0^{n-1} = x_0^{n-1}] \tag{13.77}$$

$$= \frac{1}{n} \sum_{x_0^{n-1}} p(x_0^{n-1}) \log \frac{1}{p(x_0^{n-1})} \tag{13.78}$$

$$= \frac{1}{n} H(X_0^{n-1}) \tag{13.79}$$

$$\searrow H(\mathcal{X}). \tag{13.80}$$

The second term in the expression for L_n is $\log \log R_n$, and we wish to show that

$$\frac{1}{n} E[\log \log R_n(X_0^{n-1})] \to 0. \tag{13.81}$$

Again, we use Jensen's inequality,

$$\frac{1}{n} E \log \log R_n \leq \frac{1}{n} \log E[\log R_n(X_0^{n-1})] \tag{13.82}$$

$$\leq \frac{1}{n} \log H(X_0^{n-1}), \tag{13.83}$$

where the last inequality follows from (13.79). For any $\epsilon > 0$, for large enough n, $H(X_0^{n-1}) < n(H + \epsilon)$, and therefore $\frac{1}{n} \log \log R_n < \frac{1}{n} \log n + \frac{1}{n} \log(H + \epsilon) \to 0$. This completes the proof of the theorem. □

Thus, a compression scheme that represents a string by encoding the last time it was seen in the past is asymptotically optimal. Of course, this scheme is not practical, since it assumes that both sender and receiver

have access to the infinite past of a sequence. For longer strings, one would have to look further and further back into the past to find a match. For example, if the entropy rate is $\frac{1}{2}$ and the string has length 200 bits, one would have to look an average of $2^{100} \approx 10^{30}$ bits into the past to find a match. Although this is not feasible, the algorithm illustrates the basic idea that matching the past is asymptotically optimal. The proof of the optimality of the practical version of LZ77 with a finite window is based on similar ideas. We will not present the details here, but refer the reader to the original proof in [591].

13.5.2 Optimality of Tree-Structured Lempel–Ziv Compression

We now consider the tree-structured version of Lempel–Ziv, where the input sequence is parsed into phrases, each phrase being the shortest string that has not been seen so far. The proof of the optimality of this algorithm has a very different flavor from the proof for LZ77; the essence of the proof is a counting argument that shows that the number of phrases cannot be too large if they are all distinct, and the probability of any sequence of symbols can be bounded by a function of the number of distinct phrases in the parsing of the sequence.

The algorithm described in Section 13.4.2 requires two passes over the string—in the first pass, we parse the string and calculate $c(n)$, the number of phrases in the parsed string. We then use that to decide how many bits $[\log c(n)]$ to allot to the pointers in the algorithm. In the second pass, we calculate the pointers and produce the coded string as indicated above. The algorithm can be modified so that it requires only one pass over the string and also uses fewer bits for the initial pointers. These modifications do not affect the asymptotic efficiency of the algorithm. Some of the implementation details are discussed by Welch [554] and Bell et al. [41].

We will show that like the sliding window version of Lempel–Ziv, this algorithm asymptotically achieves the entropy rate for the unknown ergodic source. We first define a parsing of the string to be a decomposition into phrases.

Definition A *parsing* S of a binary string $x_1x_2 \cdots x_n$ is a division of the string into phrases, separated by commas. A *distinct parsing* is a parsing such that no two phrases are identical. For example, 0,111,1 is a distinct parsing of 01111, but 0,11,11 is a parsing that is not distinct.

The LZ78 algorithm described above gives a distinct parsing of the source sequence. Let $c(n)$ denote the number of phrases in the LZ78 parsing of a sequence of length n. Of course, $c(n)$ depends on the sequence X^n. The compressed sequence (after applying the Lempel–Ziv algorithm)

consists of a list of $c(n)$ pairs of numbers, each pair consisting of a pointer to the previous occurrence of the prefix of the phrase and the last bit of the phrase. Each pointer requires $\log c(n)$ bits, and hence the total length of the compressed sequence is $c(n)[\log c(n) + 1]$ bits. We now show that $\frac{c(n)(\log c(n)+1)}{n} \to H(\mathcal{X})$ for a stationary ergodic sequence $X_1, X_2, \ldots, X_n$. Our proof is based on the simple proof of asymptotic optimality of LZ78 coding due to Wyner and Ziv [575].

Before we proceed to the details of the proof, we provide an outline of the main ideas. The first lemma shows that the number of phrases in a distinct parsing of a sequence is less than $n/\log n$; the main argument in the proof is based on the fact that there are not enough distinct short phrases. This bound holds for any distinct parsing of the sequence, not just the LZ78 parsing.

The second key idea is a bound on the probability of a sequence based on the number of distinct phrases. To illustrate this, consider an i.i.d. sequence of random variables X_1, X_2, X_3, X_4 that take on four possible values, $\{A, B, C, D\}$, with probabilities p_A, p_B, p_C, and p_D, respectively. Now consider the probability of a sequence $P(D, A, B, C) = p_D p_A p_B p_C$. Since $p_A + p_B + p_C + p_D = 1$, the product $p_D p_A p_B p_C$ is maximized when the probabilities are equal (i.e., the maximum value of the probability of a sequence of four distinct symbols is $1/256$). On the other hand, if we consider a sequence A, B, A, B, the probability of this sequence is maximized if $p_A = p_B = \frac{1}{2}$, $p_C = p_D = 0$, and the maximum probability for A, B, A, B is $\frac{1}{16}$. A sequence of the form A, A, A, A could have a probability of 1. All these examples illustrate a basic point—sequences with a large number of distinct symbols (or phrases) cannot have a large probability. Ziv's inequality (Lemma 13.5.5) is the extension of this idea to the Markov case, where the distinct symbols are the phrases of the distinct parsing of the source sequence.

Since the description length of a sequence after the parsing grows as $c \log c$, the sequences that have very few distinct phrases can be compressed efficiently and correspond to strings that could have a high probability. On the other hand, strings that have a large number of distinct phrases do not compress as well; but the probability of these sequences could not be too large by Ziv's inequality. Thus, Ziv's inequality enables us to connect the logarithm of the probability of the sequence with the number of phrases in its parsing, and this is finally used to show that the tree-structured Lempel–Ziv algorithm is asymptotically optimal.

We first prove a few lemmas that we need for the proof of the theorem. The first is a bound on the number of phrases possible in a distinct parsing of a binary sequence of length n.

Lemma 13.5.3 (*Lempel and Ziv* [*604*]) *The number of phrases $c(n)$ in a distinct parsing of a binary sequence $X_1, X_2, \ldots, X_n$ satisfies*

$$c(n) \leq \frac{n}{(1-\epsilon_n)\log n}, \tag{13.84}$$

where $\epsilon_n = \min\{1, \frac{\log(\log n)+4}{\log n}\} \to 0$ *as* $n \to \infty$.

Proof: Let

$$n_k = \sum_{j=1}^{k} j2^j = (k-1)2^{k+1} + 2 \tag{13.85}$$

be the sum of the lengths of all distinct strings of length less than or equal to k. The number of phrases c in a distinct parsing of a sequence of length n is maximized when all the phrases are as short as possible. If $n = n_k$, this occurs when all the phrases are of length $\leq k$, and thus

$$c(n_k) \leq \sum_{j=1}^{k} 2^j = 2^{k+1} - 2 < 2^{k+1} \leq \frac{n_k}{k-1}. \tag{13.86}$$

If $n_k \leq n < n_{k+1}$, we write $n = n_k + \Delta$, where $\Delta < (k+1)2^{k+1}$. Then the parsing into shortest phrases has each of the phrases of length $\leq k$ and $\Delta/(k+1)$ phrases of length $k+1$. Thus,

$$c(n) \leq \frac{n_k}{k-1} + \frac{\Delta}{k+1} \leq \frac{n_k + \Delta}{k-1} = \frac{n}{k-1}. \tag{13.87}$$

We now bound the size of k for a given n. Let $n_k \leq n < n_{k+1}$. Then

$$n \geq n_k = (k-1)2^{k+1} + 2 \geq 2^k, \tag{13.88}$$

and therefore

$$k \leq \log n. \tag{13.89}$$

Moreover,

$$n \leq n_{k+1} = k2^{k+2} + 2 \leq (k+2)2^{k+2} \leq (\log n + 2)2^{k+2}, \tag{13.90}$$

by (13.89), and therefore

$$k + 2 \geq \log \frac{n}{\log n + 2}, \tag{13.91}$$

or for all $n \geq 4$,

$$k - 1 \geq \log n - \log(\log n + 2) - 3 \tag{13.92}$$

$$= \left(1 - \frac{\log(\log n + 2) + 3}{\log n}\right) \log n \tag{13.93}$$

$$\geq \left(1 - \frac{\log(2 \log n) + 3}{\log n}\right) \log n \tag{13.94}$$

$$= \left(1 - \frac{\log(\log n) + 4}{\log n}\right) \log n \tag{13.95}$$

$$= (1 - \epsilon_n) \log n. \tag{13.96}$$

Note that $\epsilon_n = \min\{1, \frac{\log(\log n)+4}{\log n}\}$. Combining (13.96) with (13.87), we obtain the lemma. □

We will need a simple result on maximum entropy in the proof of the main theorem.

Lemma 13.5.4 *Let Z be a nonnegative integer-valued random variable with mean μ. Then the entropy $H(Z)$ is bounded by*

$$H(Z) \leq (\mu + 1) \log(\mu + 1) - \mu \log \mu. \tag{13.97}$$

Proof: The lemma follows directly from the results of Theorem 12.1.1, which show that the geometric distribution maximizes the entropy of a nonnegative integer-valued random variable subject to a mean constraint. □

Let $\{X_i\}_{i=-\infty}^{\infty}$ be a binary stationary ergodic process with probability mass function $P(x_1, x_2, \ldots, x_n)$. (Ergodic processes are discussed in greater detail in Section 16.8.) For a fixed integer k, define the kth-order Markov approximation to P as

$$Q_k(x_{-(k-1)}, \ldots, x_0, x_1, \ldots, x_n) \stackrel{\triangle}{=} P(x_{-(k-1)}^0) \prod_{j=1}^{n} P(x_j | x_{j-k}^{j-1}), \tag{13.98}$$

where $x_i^j \stackrel{\triangle}{=} (x_i, x_{i+1}, \ldots, x_j)$, $i \leq j$, and the initial state $x_{-(k-1)}^0$ will be part of the specification of Q_k. Since $P(X_n | X_{n-k}^{n-1})$ is itself an ergodic

process, we have

$$-\frac{1}{n}\log Q_k(X_1, X_2, \ldots, X_n|X_{-(k-1)}^0) = -\frac{1}{n}\sum_{j=1}^{n}\log P(X_j|X_{j-k}^{j-1}) \tag{13.99}$$

$$\to -E\log P(X_j|X_{j-k}^{j-1}) \tag{13.100}$$

$$= H(X_j|X_{j-k}^{j-1}). \tag{13.101}$$

We will bound the rate of the LZ78 code by the entropy rate of the kth-order Markov approximation for all k. The entropy rate of the Markov approximation $H(X_j|X_{j-k}^{j-1})$ converges to the entropy rate of the process as $k \to \infty$, and this will prove the result.

Suppose that $X_{-(k-1)}^n = x_{-(k-1)}^n$, and suppose that x_1^n is parsed into c distinct phrases, $y_1, y_2, \ldots, y_c$. Let v_i be the index of the start of the ith phrase (i.e., $y_i = x_{v_i}^{v_{i+1}-1}$). For each $i = 1, 2, \ldots, c$, define $s_i = x_{v_i-k}^{v_i-1}$. Thus, s_i is the k bits of x preceding y_i. Of course, $s_1 = x_{-(k-1)}^0$.

Let c_{ls} be the number of phrases y_i with length l and preceding state $s_i = s$ for $l = 1, 2, \ldots$ and $s \in \mathcal{X}^k$. We then have

$$\sum_{l,s} c_{ls} = c \tag{13.102}$$

and

$$\sum_{l,s} l c_{ls} = n. \tag{13.103}$$

We now prove a surprising upper bound on the probability of a string based on the parsing of the string.

Lemma 13.5.5 (*Ziv's inequality*) *For any distinct parsing (in particular, the LZ78 parsing) of the string $x_1x_2\cdots x_n$, we have*

$$\log Q_k(x_1, x_2, \ldots, x_n|s_1) \le -\sum_{l,s} c_{ls}\log c_{ls}. \tag{13.104}$$

Note that the right-hand side does not depend on Q_k.

Proof: We write

$$Q_k(x_1, x_2, \ldots, x_n|s_1) = Q_k(y_1, y_2, \ldots, y_c|s_1) \tag{13.105}$$

$$= \prod_{i=1}^{c} P(y_i|s_i) \tag{13.106}$$

or

$$\log Q_k(x_1, x_2, \ldots, x_n|s_1) = \sum_{i=1}^{c} \log P(y_i|s_i) \tag{13.107}$$

$$= \sum_{l,s} \sum_{i:|y_i|=l, s_i=s} \log P(y_i|s_i) \tag{13.108}$$

$$= \sum_{l,s} c_{ls} \sum_{i:|y_i|=l, s_i=s} \frac{1}{c_{ls}} \log P(y_i|s_i) \tag{13.109}$$

$$\leq \sum_{l,s} c_{ls} \log \left(\sum_{i:|y_i|=l, s_i=s} \frac{1}{c_{ls}} P(y_i|s_i) \right), \tag{13.110}$$

where the inequality follows from Jensen's inequality and the concavity of the logarithm.

Now since the y_i are distinct, we have $\sum_{i:|y_i|=l, s_i=s} P(y_i|s_i) \leq 1$. Thus,

$$\log Q_k(x_1, x_2, \ldots, x_n|s_1) \leq \sum_{l,s} c_{ls} \log \frac{1}{c_{ls}}, \tag{13.111}$$

proving the lemma. □

We can now prove the main theorem.

Theorem 13.5.2 *Let $\{X_n\}$ be a binary stationary ergodic process with entropy rate $H(\mathcal{X})$, and let $c(n)$ be the number of phrases in a distinct parsing of a sample of length n from this process. Then*

$$\limsup_{n\to\infty} \frac{c(n) \log c(n)}{n} \leq H(\mathcal{X}) \tag{13.112}$$

with probability 1.

Proof: We begin with Ziv's inequality, which we rewrite as

$$\log Q_k(x_1, x_2, \ldots, x_n|s_1) \leq -\sum_{l,s} c_{ls} \log \frac{c_{ls} c}{c} \tag{13.113}$$

$$= -c \log c - c \sum_{ls} \frac{c_{ls}}{c} \log \frac{c_{ls}}{c}. \tag{13.114}$$

Writing $\pi_{ls} = \frac{c_{ls}}{c}$, we have

$$\sum_{l,s} \pi_{ls} = 1, \qquad \sum_{l,s} l\pi_{ls} = \frac{n}{c}, \tag{13.115}$$

from (13.102) and (13.103). We now define random variables U, V such that

$$\Pr(U = l, V = s) = \pi_{ls}. \tag{13.116}$$

Thus, $EU = \frac{n}{c}$ and

$$\log Q_k(x_1, x_2, \ldots, x_n | s_1) \leq cH(U, V) - c \log c \tag{13.117}$$

or

$$-\frac{1}{n} \log Q_k(x_1, x_2, \ldots, x_n | s_1) \geq \frac{c}{n} \log c - \frac{c}{n} H(U, V). \tag{13.118}$$

Now

$$H(U, V) \leq H(U) + H(V) \tag{13.119}$$

and $H(V) \leq \log |\mathcal{X}|^k = k$. By Lemma 13.5.4, we have

$$H(U) \leq (EU + 1) \log(EU + 1) - (EU) \log(EU) \tag{13.120}$$

$$= \left(\frac{n}{c} + 1\right) \log \left(\frac{n}{c} + 1\right) - \frac{n}{c} \log \frac{n}{c} \tag{13.121}$$

$$= \log \frac{n}{c} + \left(\frac{n}{c} + 1\right) \log \left(\frac{c}{n} + 1\right). \tag{13.122}$$

Thus,

$$\frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1). \tag{13.123}$$

For a given n, the maximum of $\frac{c}{n} \log \frac{n}{c}$ is attained for the maximum value of c (for $\frac{c}{n} \leq \frac{1}{e}$). But from Lemma 13.5.3, $c \leq \frac{n}{\log n}(1 + o(1))$. Thus,

$$\frac{c}{n} \log \frac{n}{c} \leq O\left(\frac{\log \log n}{\log n}\right), \tag{13.124}$$

and therefore $\frac{c}{n}H(U, V) \to 0$ as $n \to \infty$. Therefore,

$$\frac{c(n)\log c(n)}{n} \le -\frac{1}{n}\log Q_k(x_1, x_2, \ldots, x_n|s_1) + \epsilon_k(n), \tag{13.125}$$

where $\epsilon_k(n) \to 0$ as $n \to \infty$. Hence, with probability 1,

$$\limsup_{n\to\infty} \frac{c(n)\log c(n)}{n} \le \lim_{n\to\infty} -\frac{1}{n}\log Q_k(X_1, X_2, \ldots, X_n|X^0_{-(k-1)}) \tag{13.126}$$

$$= H(X_0|X_{-1}, \ldots, X_{-k}) \tag{13.127}$$

$$\to H(\mathcal{X}) \qquad \text{as } k \to \infty. \quad \square \tag{13.128}$$

We now prove that LZ78 coding is asymptotically optimal.

Theorem 13.5.3 *Let $\{X_i\}_{-\infty}^{\infty}$ be a binary stationary ergodic stochastic process. Let $l(X_1, X_2, \ldots, X_n)$ be the LZ78 codeword length associated with $X_1, X_2, \ldots, X_n$. Then*

$$\limsup_{n\to\infty} \frac{1}{n} l(X_1, X_2, \ldots, X_n) \le H(\mathcal{X}) \quad \textit{with probability 1}, \tag{13.129}$$

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

Proof: We have shown that $l(X_1, X_2, \ldots, X_n) = c(n)(\log c(n) + 1)$, where $c(n)$ is the number of phrases in the LZ78 parsing of the string $X_1, X_2, \ldots, X_n$. By Lemma 13.5.3, $\limsup c(n)/n = 0$, and thus Theorem 13.5.2 establishes that

$$\limsup \frac{l(X_1, X_2, \ldots, X_n)}{n} = \limsup \left(\frac{c(n)\log c(n)}{n} + \frac{c(n)}{n} \right)$$

$$\le H(\mathcal{X}) \quad \text{with probability 1.} \ \square \tag{13.130}$$

Thus, the length per source symbol of the LZ78 encoding of an ergodic source is asymptotically no greater than the entropy rate of the source. There are some interesting features of the proof of the optimality of LZ78 that are worth noting. The bounds on the number of distinct phrases and Ziv's inequality apply to any distinct parsing of the string, not just the incremental parsing version used in the algorithm. The proof can be extended in many ways with variations on the parsing algorithm; for example, it is possible to use multiple trees that are context or state

dependent [218, 426]. Ziv's inequality (Lemma 13.5.5) remains particularly intriguing since it relates a probability on one side with a purely deterministic function of the parsing of a sequence on the other.

The Lempel–Ziv codes are simple examples of a universal code (i.e., a code that does not depend on the distribution of the source). This code can be used without knowledge of the source distribution and yet will achieve an asymptotic compression equal to the entropy rate of the source.

SUMMARY

Ideal word length

$$l^*(x) = \log \frac{1}{p(x)}. \tag{13.131}$$

Average description length

$$E_p l^*(x) = H(p). \tag{13.132}$$

Estimated probability distribution $\hat{p}(x)$. If $\hat{l}(x) = \log \frac{1}{\hat{p}(x)}$, then

$$E_p \hat{l}(x) = H(p) + D(p||\hat{p}). \tag{13.133}$$

Average redundancy

$$R_p = E_p l(X) - H(p). \tag{13.134}$$

Minimax redundancy. For $X \sim p_\theta(x)$, $\theta \in \theta$,

$$D^* = \min_l \max_p R_p = \min_q \max_\theta D(p_\theta||q). \tag{13.135}$$

Minimax theorem. $D^* = C$, where C is the capacity of the channel $\{\theta, p_\theta(x), \mathcal{X}\}$.

Bernoulli sequences. For $X^n \sim$Bernoulli(θ), the redundancy is

$$D_n^* = \min_q \max_\theta D(p_\theta(x^n)||q(x^n)) \approx \frac{1}{2}\log n + o(\log n). \tag{13.136}$$

Arithmetic coding. nH bits of $F(x^n)$ reveal approximately n bits of x^n.

Lempel–Ziv coding (recurrence time coding). Let $R_n(X^n)$ be the last time in the past that we have seen a block of n symbols X^n. Then $\frac{1}{n}\log R_n \to H(\mathcal{X})$, and encoding by describing the recurrence time is asymptotically optimal.

Lempel–Ziv coding (sequence parsing). If a sequence is parsed into the shortest phrases not seen before (e.g., 011011101 is parsed to 0,1,10,11,101,...) and $l(x^n)$ is the description length of the parsed sequence, then

$$\limsup \frac{1}{n} l(X^n) \le H(\mathcal{X}) \qquad \text{with probability 1} \tag{13.137}$$

for every stationary ergodic process $\{X_i\}$.

PROBLEMS

13.1 *Minimax regret data compression and channel capacity.* First consider universal data compression with respect to two source distributions. Let the alphabet $V = \{1, e, 0\}$ and let $p_1(v)$ put mass $1-\alpha$ on $v = 1$ and mass α on $v = e$. Let $p_2(v)$ put mass $1-\alpha$ on 0 and mass α on $v = e$. We assign word lengths to V according to $l(v) = \log \frac{1}{p(v)}$, the ideal codeword length with respect to a cleverly chosen probability mass function $p(v)$. The worst-case excess description length (above the entropy of the true distribution) is

$$\max_i \left(E_{p_i} \log \frac{1}{p(V)} - E_{p_i} \log \frac{1}{p_i(V)} \right) = \max_i D(p_i \parallel p). \tag{13.138}$$

Thus, the minimax regret is $D^* = \min_p \max_i D(p_i \parallel p)$.

(a) Find D^*.

(b) Find the $p(v)$ achieving D^*.

(c) Compare D^* to the capacity of the binary erasure channel

$$\begin{bmatrix} 1-\alpha & \alpha & 0 \\ 0 & \alpha & 1-\alpha \end{bmatrix}$$

and comment.