LEMPEL–ZIV CODING

A drawback of the Huffman code is that it requires knowledge of a probabilistic model of the source; unfortunately, in practice, source statistics are not always known *a priori*. Moreover, in modeling text we find that storage requirements prevent the Huffman code from capturing the higher-order relationships between words and phrases, thereby compromising the efficiency of the code. To overcome these practical limitations, we may use the *Lempel–Ziv algorithm*,[7] which is intrinsically *adaptive* and simpler to implement than Huffman coding.

Basically, encoding in the Lempel–Ziv algorithm is accomplished by *parsing the source data stream into segments that are the shortest subsequences not encountered previously*. To illustrate this simple yet elegant idea, consider the example of an input binary sequence specified as follows:

$$000101110010100101\ldots$$

It is assumed that the binary symbols 0 and 1 are already stored in that order in the code book. We thus write

Subsequences stored:	0, 1
Data to be parsed:	000101110010100101 . . .

The encoding process begins at the left. With symbols 0 and 1 already stored, the *shortest subsequence* of the data stream encountered for the first time and not seen before is 00; so we write

Subsequences stored:	0, 1, 00
Data to be parsed:	0101110010100101 . . .

The second shortest subsequence not seen before is 01; accordingly, we go on to write

Subsequences stored:	0, 1, 00, 01
Data to be parsed:	01110010100101 . . .

The next shortest subsequence not encountered previously is 011; hence, we write

Subsequences stored:	0, 1, 00, 01, 011
Data to be parsed:	10010100101 . . .

We continue in the manner described here until the given data stream has been completely parsed. Thus, for the example at hand, we get the *code book* of binary subsequences shown in the second row of Figure 9.6.

Numerical positions:	1	2	3	4	5	6	7	8	9
Subsequences:	0	1	00	01	011	10	010	100	101
Numerical representations:			11	12	42	21	41	61	62
Binary encoded blocks:			0010	0011	1001	0100	1000	1100	1101

FIGURE 9.6 Illustrating the encoding process performed by the Lempel-Ziv algorithm on the binary sequence 000101110010100101. . . .

The first row shown in this figure merely indicates the numerical positions of the individual subsequences in the code book. We now recognize that the first subsequence of the data stream, 00, is made up of the concatenation of the *first* code book entry, 0, with itself; it is therefore represented by the number 11. The second subsequence of the data stream, 01, consists of the *first* code book entry, 0, concatenated with the *second* code book entry, 1; it is therefore represented by the number 12. The remaining subsequences are treated in a similar fashion. The complete set of numerical representations for the various subsequences in the code book is shown in the third row of Figure 9.6. As a further example illustrating the composition of this row, we note that the subsequence 010 consists of the concatenation of the subsequence 01 in position 4 and symbol 0 in position 1; hence, the numerical representation 41. The last row shown in Figure 9.6 is the binary encoded representation of the different subsequences of the data stream.

The last symbol of each subsequence in the code book (i.e., the second row of Figure 9.6) is an *innovation symbol*, which is so called in recognition of the fact that its appendage to a particular subsequence distinguishes it from all previous subsequences stored in the code book. Correspondingly, the last bit of each uniform block of bits in the binary encoded representation of the data stream (i.e., the fourth row in Figure 9.6) represents the innovation symbol for the particular subsequence under consideration. The remaining bits provide the equivalent binary representation of the "pointer" to the *root subsequence* that matches the one in question except for the innovation symbol.

The decoder is just as simple as the encoder. Specifically, it uses the pointer to identify the root subsequence and then appends the innovation symbol. Consider, for example, the binary encoded block 1101 in position 9. The last bit, 1, is the innovation symbol. The remaining bits, 110, point to the root subsequence 10 in position 6. Hence, the block 1101 is decoded into 101, which is correct.

From the example described here, we note that, in contrast to Huffman coding, the Lempel–Ziv algorithm uses fixed-length codes to represent a variable number of source symbols; this feature makes the Lempel–Ziv code suitable for synchronous transmission. In practice, fixed blocks of 12 bits long are used, which implies a code book of 4096 entries.

For a long time, Huffman coding was unchallenged as the algorithm of choice for data compaction. However, the Lempel–Ziv algorithm has taken over almost completely from the Huffman algorithm. The Lempel–Ziv algorithm is now the standard algorithm for file compression. When it is applied to ordinary English text, the Lempel–Ziv algorithm achieves a compaction of approximately 55 percent. This is to be contrasted with a compaction of approximately 43 percent achieved with Huffman coding. The reason for this behavior is that, as mentioned previously, Huffman coding does not take advantage of the intercharacter redundancies of the language. On the other hand, the Lempel–Ziv algorithm is able to do the best possible compaction of text (within certain limits) by working effectively at higher levels.