Multimedia Data Compression

INTRODUCTION

The emergence of multimedia technologies has made *digital libraries* a reality. Nowadays, libraries, museums, film studios, and governments are converting more and more data and archives into digital form. Some of the data (e.g., precious books and paintings) indeed need to be stored without any loss.
As a start, suppose we want to encode the call numbers of the 120 million or so items in the Library of Congress (a mere 20 million, if we consider just books). Why don't we just transmit each item as a 27-bit number, giving each item a unique binary code (since227 > 120,000, OOO)?
The main problem is that this "great idea" requires too many bits. And in fact there exist many coding techniques that will effectively reduce the total number of bits needed to represent the above information. The process involved is generally referred to as *compression* [1,2J].

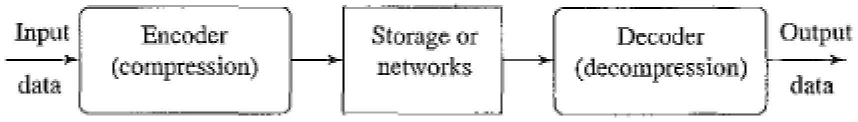

FIGURE 7.1: A general data compression scheme.

If the total number of bits required to represent the data before compression is *Eo* and the total number of bits require

$$\textit{compression ratio} = \frac{B_0}{B_1}$$

represent the data before compression is *Eo* and the total number of bits required to represent the data after compression is 81, then we define the *compression ratio* as (7.1)In general, we would desire any *codec* (encoder/decoder scheme) to have a *compression ratio* much larger than 1.0. The higher the *compression ratio*, the better the lossless compression scheme, as long as it is computationally feasible.

BASICS OF INFORMATION THEORY

According to the famous scientist Claude E. Shannon, of Bell Labs [3, 4J, the *entropy* 1) of an information *source* with alphabet *S* = {SI, *S2,* ••. *,511 }* is defined as:

$$\begin{aligned}\eta = H(S) &= \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} \\ &= -\sum_{i=1}^{n} p_i \log_2 p_i\end{aligned}$$

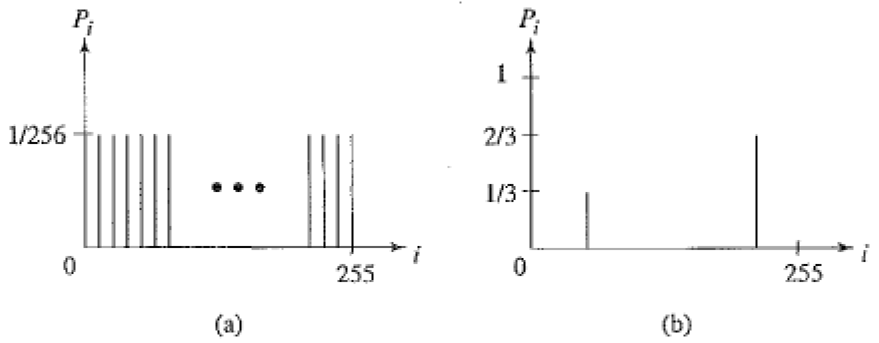

FIGURE 7.2: Histograms for two gray-level images.

Figure 7.2(a) shows the histogram of an image with *Ill1iform* distribution of gray-level intensities, - that is, *Vi Pi* = 1/256. Hence, the entropy of this image is

$$\eta = \sum_{i=0}^{255} \frac{1}{256} \cdot \log_2 256 = 8$$

As can be seen in Eq. (7.3), the entropy 17 is a weighted sum of tenns log2 ~i ; hence it represents the *average* amount of information contained per symbol in the source S. For a memory less source2 S, the entropy 1) represents the minimum average number of bits required to represent each symbol in S. In other words, it specifies the lower bound for the average number of bits to code each symbol in S.
If we use *I* to denote the average length (measured in bits) of the code words produced by the encoder, the Shannon Coding Theorem states that the entropy is the *best* we can do (under certain conditions):
Coding schemes aim to get as close as possible to this theoretical lower bound. It is interesting to observe that in the above uniform-distribution example we found that 1) = 8 - the minimum average number of bits to represent each gray-level intensity is at least 8. No compression is possible for this image! In the context of imaging, this will correspond to the "worst case," where neighboring pixel values have no similarity. Figure 7.2(b) shows the histogram of another image, in which 1/3 of the pixels are rather dark and 2/3 of them are rather bright. The entropy of this image is

$$\begin{aligned} \eta &= \frac{1}{3} \cdot \log_2 3 + \frac{2}{3} \cdot \log_2 \frac{3}{2} \\ &= 0.33 \times 1.59 + 0.67 \times 0.59 = 0.52 + 0.40 = 0.92 \end{aligned}$$

In general, the entropy is greater when the probability distribution is flat and smaller when it is more peaked.

RUN-LENGTH CODING

Instead of assuming a memory less source, *run-length coding* (RLC) exploits memory present in the information sour<;e. It is one of the simplest fonns of data compression. The basic idea is that if the information source we wish to compress has the property that symbols tend to form continuous groups, instead of coding each symbol in the group individually, we can code one such symbol and the length of the group.
As an example, consider a bilevel image (one with only I-bit black and white pixels)with monotone regions. This information source can be efficiently coded using run-length coding. In fact, since there are only two symbols, we do not even need to code any symbol at the start of eachrun. Instead, we can assume that the starting run is always of a particular color (either black or white) and simply code the length of each run.
The above description is the one-dimensional run-length coding algorithm. A twodimensionalvariant of it is usually used to code bilevel images. This algorithm uses thecoded run information in the previous row of the image to code the run in the CUiTent row.A full description of this algorithm can be found in [5].

VARIABLE·LENGTH CODING (VLC)

Since the entropy indicates the information content in an information source S, it leads to a family of coding methods commonly known as *entropy coding* methods. As described earlier, *variable-length coding* (VLC) is one of the best-known such methods. Here, we will study the Shannon~Fano algorithm, Huffman coding, and adaptive Huffman coding.

Shannon-Fano Algorithm

The Shannon-Fano algorithm was independently developed by Shannon at Bell Labs andRobert Fana at MIT [6]. To illustrate the algorithm, let's suppose the symbols to be codedare the characters in the word HELLO. The frequency count of the symbols is

Symbol	H	E	L	O
Count	1	1	2	1

The encoding steps of the Shannon-Fano algorithm can be presented in the following *top-down* manner:
1. Sort the symbols according to t~e frequency count of their occurrences.
2. Recursively divides the symbols into two parts, each with approximately the same number of counts, until an parts contain only one symbol.
A natural way of implementing the above procedure is to build a binary tree. As a convention, let's assign bit 0 to its left branches and 1 to the right branches.

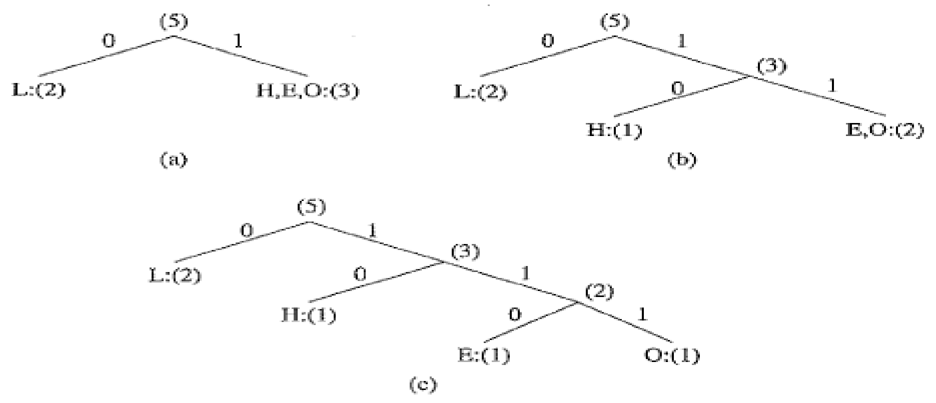

FIGURE 7.3: Coding tree for HELLO by the Shannon–Fano algorithm.

Initially, the symbols are sorted as LHEO. As Figure 7.3 shows, the first division yields two parts: (a) L with a count of2, denoted as L:(2); and (b) H, E and 0 with a total count of 3, denoted as H,E,0:(3). The second division yields H:(l) and E,O:(2). The last division is E:(1) and 0:(1).
Table 7.1 summarizes the result, showing each symbol, its frequency count, information content (log2 .;;), resulting codeword, and the number ofbits needed to encode each symbol in the word HELLO. The total number of bits used is shown at the bottom. To revisit the previous discussion on entropy, in this case,

$$\begin{aligned}\eta &= p_L \cdot \log_2 \frac{1}{p_L} + p_H \cdot \log_2 \frac{1}{p_H} + p_E \cdot \log_2 \frac{1}{p_E} + p_O \cdot \log_2 \frac{1}{p_O} \\ &= 0.4 \times 1.32 + 0.2 \times 2.32 + 0.2 \times 2.32 + 0.2 \times 2.32 = 1.92\end{aligned}$$

TABLE 7.1: One result of performing the Shannon–Fano algorithm on HELLO.

Symbol	Count	$\log_2 \frac{1}{p_i}$	Code	Number of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
TOTAL number of bits:				10

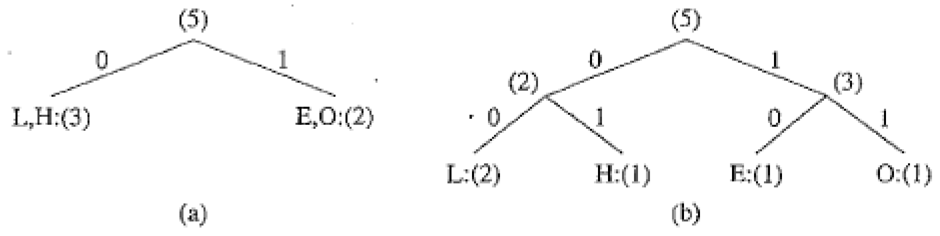

FIGURE 7.4: Another coding tree for HELLO by the Shannon–Fano algorithm.

This suggests that the minimum average number of bits to code each character in the word HELLO would be at least 1.92. In this example,. the Shannon-Fano algorithm uses an average of 10/5 = 2 bits to code each symbol, which is fairly close to the lower bound of1.92. Apparently, the result is satisfactory.
It should be pointed out that the outcome of the Shannon-Fano algorithm is not necessarily unique. For instance, at the first division in the above example, it would be equally valid to divide into the two parts L,H:(3) and E,O:(2). This would result in the coding in Figure 7.4. Table 7.2 shows the code words are different now. Also, these two sets of code words may behave differently when errors are present. Coincidentally, the total number of bits required to encode the world HELLO remains at 10.
The Shannon~Fano algorithm delivers satisfactory coding results for data compression, but it was soon outperformed and overtaken by the Huffman coding method.

Huffman Coding

First presented by David A. Huffman in a 1952 paper [7], this method attracted an overwhelming amount of research and has been adopted in many important and/or commercial applications, such as fax machines, JPEG, and MPEG.
In contradistinction to Shannon-Fano, which is top-down, the encoding steps of the Huffman algorithm are described in the following *bottom-up* manner. Let's use the same example word, HELLO. A similar binary coding tree will be used as above, in which the left branches are coded 0 and right branches 1. A simple list data structure is also used.

TABLE 7.2: Another result of performing the Shannon–Fano algorithm on HELLO.

Symbol	Count	$\log_2 \frac{1}{p_i}$	Code	Number of bits used
L	2	1.32	00	4
H	1	2.32	01	2
E	1	2.32	10	2
O	1	2.32	11	2
TOTAL number of bits:				10

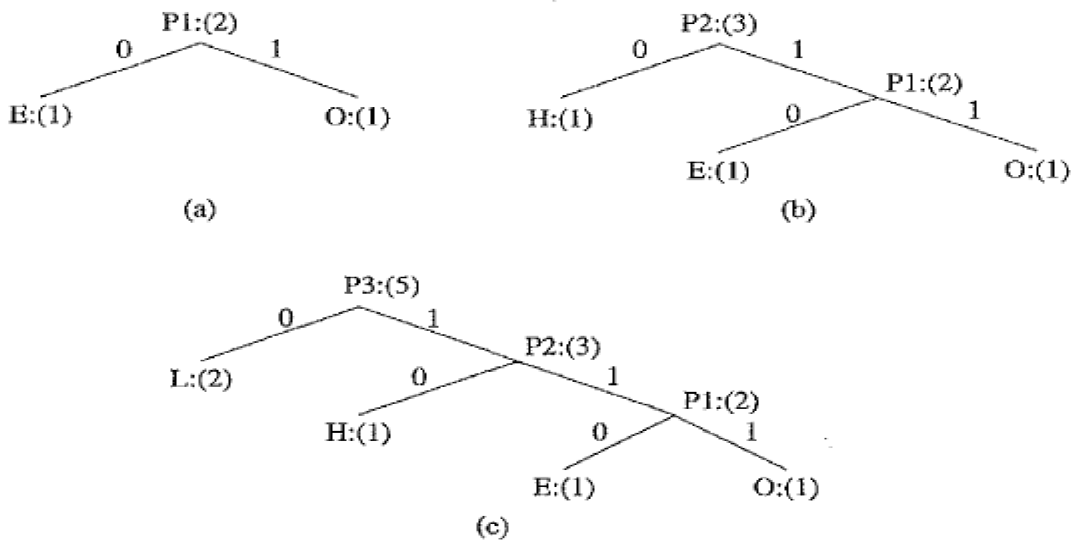

FIGURE 7.5: Coding tree for HELLO using the Huffman algorithm.

ALGORITHM 7.1 HUFFMAN CODING
1. Initialization; put all symbols on the list sorted according to their frequency counts.
2. Repeat until the list has only one symbol left.
(a) From the list, pick two symbols with the lowest frequency counts. Form a Huffman subtree that has these two symbols as child nodes and create a parent node for them.
(b) Assign the sum of the children's frequency counts to the parent and insert it into the list, such that the order is maintained.
(c) Delete the children from the list.
3. Assign a codeword for each leaf based on the path from the root.

In the above figure, new symbols PI, Pl, P3 are created to refer to the parent nodes in the Huffman coding tree. The contents in the list are illustrated below:
After initialization: LHEO

After iteration (a): LPI H

After iteration (b): LP2

After iteration (c): P3