Chapter 3

Compression

In Chapter 1 we examined the fundamental unit of information, the bit, and its various abstract representations: the mathematical bit, the classical bit, and the quantum bit.

A single bit is useful if exactly two answers to a question are possible. Examples include the result of a coin toss (heads or tails), the gender of a person (male or female), the verdict of a jury (guilty or not guilty), and the truth of an assertion (true or false). Most situations in life are more complicated. In Chapter 2 we considered some of the issues surrounding the representation of complex objects by arrays of bits. The mapping between the objects to be represented (the symbols) and the array of bits used for this purpose is known as a code.

In our never-ending quest for improvement, we are led to representations of single bits that are stronger, smaller, faster, and cheaper. Following this route leads to fundamental limitations imposed by quantum mechanics. We are also led to codes that are similarly stronger and smaller. In this chapter we will consider techniques of compression that can be used for generation of particularly efficient representations.

In Chapter 2 we looked at this general sort of a system, in which objects are encoded into bit strings, these are transported (in space and/or time) to a decoder, which then recreates the original objects. Typically the same code is used for a succession of objects one after another.

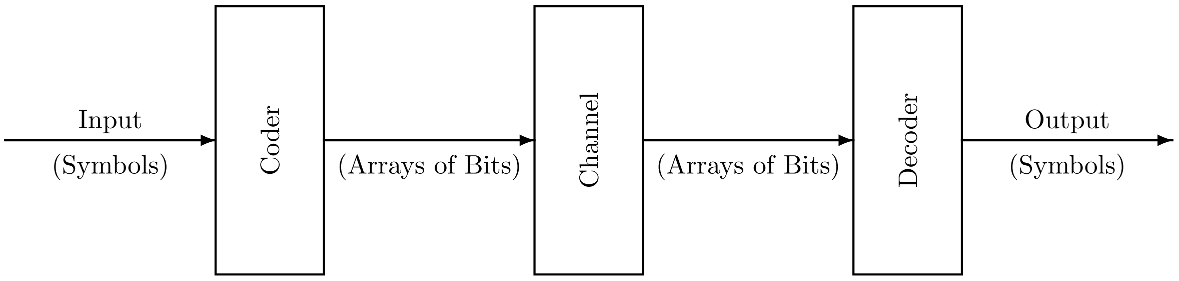

Figure 3.1: Generalized communication system

The role of data compression is to convert the string of bits representing a succession of symbols into a shorter string for more economical transmission, storage, or processing. The result is this system, with both a compressor and an expander. Ideally, the expander would exactly reverse the action of the compressor so that the coder and decoder could be unchanged.

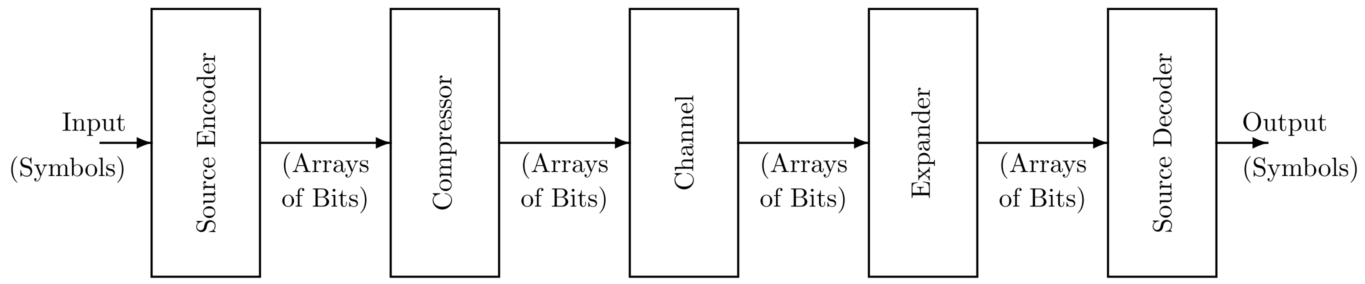

Figure 3.2: More elaborate communication system

This approach might seem surprising. Is there any reason to believe that the same information could be contained in a smaller number of bits? There are two types of compression, relying on different approaches:

- **Lossless** or **reversible** compression, which can only be done if the original code was inefficient, for example by not taking into account the fact that some symbols are more frequently used than others, or by having unused bit patterns.
- **Lossy** or **irreversible** compression, in which the original symbol, or its coded representation, cannot be reconstructed exactly, but instead the expander produces an approximation that is ”good enough”

[This version of Chapter 3 is a draft that is not yet complete. Two detail sections have been written so far. There is more to come.]

3.2 Detail: LZW Compression

There are several compression algorithms that use a “dictionary,” or code book, known to the coder and the decoder, which is generated during the coding and decoding processes. Many of these build on work reported in 1967 by Abraham Lempel and Jacob Ziv, and are known as “Lempel-Ziz” encoders. In essense, these coders replace repeated occurrences of a string by references to an earlier occurrence. The dictionary is merely the collection of these earlier occurrences.

One widely used LZ algorithm is the LZW algorithm described by Terry A. Welch.[1] This algorithm was originally designed to minimize the number of bits sent to and from disks, but it has been used in many contexts, including GIF compression programs for images.

The LZW compression algorithm is “reversible,” meaning that it does not lose any information – the decoder is able to reconstruct the original message exactly.

LZW Algorithm, Example 1

Encode and decode the text message

```
itty bitty bit bin
```

(this peculiar phrase was designed to have repeated strings so that the dictionary forms rapidly).

The initial set of dictionary entries is 8-bit character code with values 0-255, with ASCII as the first 128 characters, including specifically the following which appear in the string

32	space
98	b
105	i
110	n
116	t
121	y

Table 3.1: LZW Example 1 Dictionary

Dictionary entry 256 is reserved for the “clear dictionary” command, and 257 is reserved for “end of transmission.”

During encoding and decoding, new dictionary entries are created using all phrases present in the text that are not yet in the dictionary.

Encoding algorithm: Accumulate characters until the string does not match any dictionary entry. Then define this new string as a new entry, but send the entry corresponding to the string without the last character, which you use as the first character of the next string to match.

When this procedure is applied to the string in question, the first character is “i” and the string consisting of just that character is already in the dictionary. So the next character is added, and the accumulated string is “it” which is not in the dictionary. At this point the string which was in the dictionary, “i”, is sent and the string “it” is added to the dictionary, at the next available entry, which is 258. The accumulated string is reset to be just the last character, which was not sent, so it is “t”. The next character is added so the accumulated string is “tt” which is not in the dictionary. The process repeats.

For a while at the beginning the additional dictionary entries are all two-character strings, and there is a string transmitted for every new character encountered. However, the first time one of those two-character strings is repeated, it gets sent (using fewer bits than would be required for two characters) and a new three-character dictionary entry is defined. In this example it happens with the string “itt” (this message was designed to make this happen earlier than would be expected with normal text). Later in this example, one three-character string gets transmitted, and a four-character dictionary entry defined.

[1]Welch, T.A. ‘A Technique for High Performance Data Compression,” IEEE Computer, vol. 17, no. 6, pp. 8-19, 1984.

Decoding algorithm: Output character strings whose code is transmitted. For each code transmission, define a new dictionary entry as the previous string plus the first character of the string just received. Note that the coder and decoder create the dictionary on the fly; the dictionary therefore does not have to be explicitly transmitted, and the coder deals with the text in a single pass.

Encoding →				Transmission →		Decoding		
Input		New dictionary entry		9-bit characters transmitted		New dictionary entry		Output
105	i	–	–	256	(start)	–	–	–
116	t	258	it	105	i	–	–	i
116	t	259	tt	116	t	258	it	t
121	y	260	ty	116	t	259	tt	t
32	space	261	y-space	121	y	260	ty	y
98	b	262	space-b	32	space	261	y-space	space
105	i	263	bi	98	b	262	space-b	b
116	t	–	–	–	–	–	–	–
116	t	264	itt	258	it	263	bi	it
121	y	–	–	–	–	–	–	–
32	space	265	ty-space	260	ty	264	itt	ty
98	b		–	–	–	–	–	–
105	i	266	space-bi	262	space-b	265	ty-space	space-b
116	t	–	–	–		–	–	–
32	space	267	it-space	258	it	266	space-bi	it
98	b	–	–	–		–	–	–
105	i	–	–	–		–	–	–
110	n	268	space-bin	266	space-bi	267	it-space	space-bi
–	–	–	–	110	n	268	space-bin	n
–	–	–	–	257	(stop)	–	–	–

8-bit characters input

Table 3.2: LZW Example 1 Transmission Summary

Does this work, i.e., is the number of bits needed for transmission reduced? We sent 18 8-bit characters (144 bits) in 14 9-bit transmissions (126 bits), a savings of 12.5%, even for this very short example. For more normal text there is not much reduction for strings under 500 bytes. In practice, however, larger text files often compress by a factor of 2, and drawings by even more.

LZW Algorithm, Example 2

Encode and decode the text message

`itty bitty nitty grrritty bit bin`

(again, this peculiar phrase was designed to have repeated strings so that the dictionary forms rapidly; it also has a three-long sequence `rrr` which illustrates one of the aspects of this algorithm).

The initial set of dictionary entries include the characters in Table 3.3, which are found in the string.

Using the same algorithms as before, we obtain the following transaction:

In this example, 33 8-bit characters (264 bits) were sent in 22 9-bit transmissions (198 bits), a saving of 25% in bits. There was one four-character dictionary entry transmitted.

32	space	110	n
98	b	114	r
103	g	116	t
105	i	121	y
256	clear dictionary	257	end of transmission

Table 3.3: LZW Example 2 Dictionary

Encoding				$\longrightarrow$ Transmission		$\longrightarrow$ Decoding		
Input		New dictionary entry				New dictionary entry		Output
105	i	–	–	256	(start)	–	–	–
116	t	258	it	105	i	–	–	i
116	t	259	tt	116	t	258	it	t
121	y	260	ty	116	t	259	tt	t
32	space	261	y-space	121	y	260	ty	y
98	b	262	space-b	32	space	261	y-space	space
105	i	263	bi	98	b	262	space-b	b
116	t	–	–	–	–	–	–	–
116	t	264	itt	258		263	bi	it
121	y	–	–	–	–	–	–	–
32	space	265	ty-space	260	ty	264	itt	ty
110	n	266	space-n	32	space	265	ty-space	space
105	i	267	ni	110	n	266	space-n	n
116	t	–	–	–	–	–	–	–
116	t	–	–	–	–	–	–	–
121	y	268	itty	264	itt	267	ni	itt
32	space	–	–	–	–	–	–	–
103	g	269	y-space-g	261	y-space	268	itty	y-space
114	r	270	gr	103	g	269	y-space-g	g
114	r	271	rr	114	r	270	gr	r
114	r	–	–	–	–	–	–	–
105	i	272	rri	271	rr	271	rr	rr
116	t	–	–	–	–	–	–	–
116	t	–	–	–	–	–	–	–
121	y	–	–	–	–	–	–	–
32	space	273	itty-space	268	itty	272	rri	itty
98	b	–	–	–	–	–	–	–
105	i	274	space-bi	262	space-b	273	itty-space	space-b
116	t	–	–	–	–	–	–	–
32	space	275	it-space	258	it	274	space-bi	it
98	b	–	–	–	–	–	–	–
105	i	–	–	–	–	–	–	–
110	n	276	space-bin	274	space-bi	275	it-space	space-bi
–	–	–	–	110	n	276	space-bin	n
–	–	–	–	257	(stop)	–	–	–

Table 3.4: LZW Example 2 Transmission Summary