DATA COMPRESSION - INTRODUCTION

1. Definitions

Data Compression term is, I hope, well known. A possible general definition could be:

> ***Data Compression is the processing done to some data in order to reduce the dimension of their representation.***

The compression efficiency obtained by one specific method can be evaluated by the compression ratio. Considering the previous definition we have:

> ***The compression ratio is equal to the ratio between the dimension of data representation without compression and the dimension of data representation obtained with compression.***

Sometimes, for low compression ratios, the reversed ratio is considered, usually expressed in percent. Depending on the context, we have to decide which of the two approaches is used. In specific cases the compression ratio is redefined (detailed) corresponding to those situations.

Compression is done by changing the representation of the data, being a specific case of **coding** (therefore, the terms compression and coding will be used almost interchangeable, as well-known for different specific methods). Coding is done with regard to a specific data model, so that we are in a more general case of **modeling**, in search for suitable (optimum) model of the data.

2. Classifications

In the compression area there are a lot of methods, usually presented individually, without trying to integrate them. This leads us to the need of classifications, which can be done by different criteria.

The most important classification of compression methods is done regarding the **data reconstruction error**. Considering this, we have two major categories:

1. **Methods without loss – LOSSLESS** – where data are reconstructed perfectly, without any difference between original and reconstructed data. These methods apply generally when the lossless reconstruction is critical: compression of executable code, program sources, texts, in general data of numerical nature.
2. **Methods with loss – LOSSY –** where data are reconstructed within the limit of some errors considered acceptable. The acceptance of some errors leads us to a spectacular growth of the compression ratio compared to the previous case. These methods are applied in general to data of

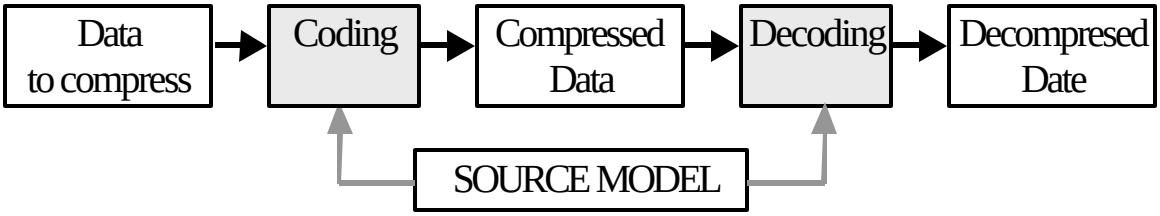

Figure 1.1 General compression scheme for static modeling

analogical nature (which represent signals that originally where analogical) to be used by a human. In that category fall the audio and video signals where, in the processing chain, a quantizer exists anyway, so that some errors are inherent.

A general method is not always by default lossless or lossy. By example, the predictive techniques can go in both categories, depending on the data they apply on (numerical or analogical nature) and by the existence in the processing chain of a quantizer.

Depending on the way the data **source model changes in time** the compression methods can be classified in:

1. Static methods - where the model is fix, does not change in time, and it is build apriori, based on some messages considered typical. In that category falls the compression used in fax transmissions. Static methods are useful when the data stream statistics is known and it remains unchanged, the resulting methods being dedicated to a specific context. Because of their static approach they are well-suited for hardware implementations. The general compression (coding) scheme for static methods is presented in Figure 1.1.

2. Semistatic Methods - where the model is built before coding (compression) based on the data that will be coded and is kept unchanged during coding. The advantage consists in the adaptation of the method to the data stream (methods being more general applicable). The disadvantages are the need to **pass twice** through the data and the need to **transmit the model** to the decoder, fact

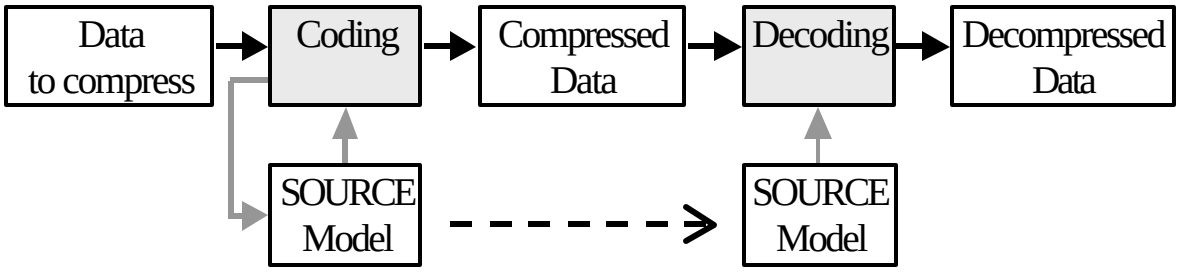

Figure 1.2 General compression scheme for semistatic modeling

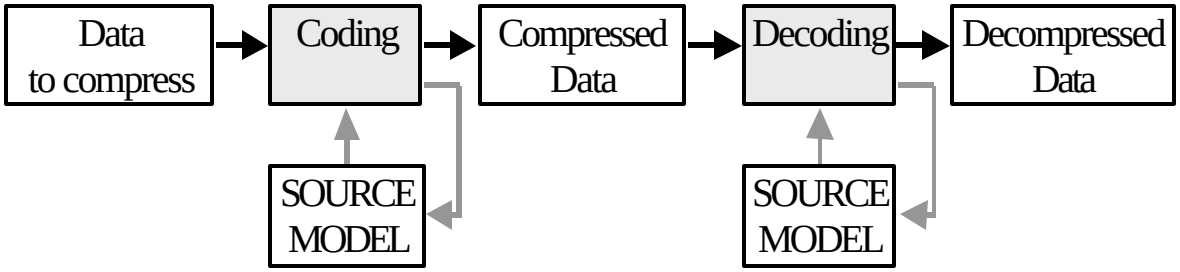

Figure 1.3 General compression scheme for adaptive modeling

that overloads the compressed stream and worsens the performances. In that category we have the method based on Huffman coding (well-known as "static" Huffman – unfortunately a misleading name but often used). The general compression (coding) scheme for this case is given in Figure 1.2.

3. Adaptive Methods (dynamic) - where compression starts at one state of the model, the same at compression and decompression. The initial state of the model can be obtained as in any of the previous cases. Each symbol is coded/decoded using the **current** model. **After** coding/decoding the state of the model is **updated in the same way** at the coder and at the decoder. In this way we overpass the two disadvantages of the semistatic methods. The most well known adaptive methods are adaptive Huffman compression and dictionary methods. The general scheme for adaptive modeling is given in Figure 1.3.

Depending on the **information used to build the model** we can classify most of the lossless compression methods in:

1. Compression methods based on statistic modeling of discrete sources (statistic methods)

Statistic modeling of a discrete source of information consists of giving to each symbol of the source alphabet a probability of appearance and, after that, a code with a smaller number of bits to the symbols with a greater probability of appearance. In that category we have the Shannon-Fano coding, Huffman coding (the semistatic version is known as optimal for the symbol by symbol coding) and arithmetic coding (which codes sequences of symbols and not individual symbols.

2. Compression methods based on linguistic modeling of discrete sources (dictionary methods)

These methods are based on the fact that the symbols generated by the source have some restrains regarding the combinations they can appear. Because of that restrains not all the possible combinations appear with the same probability. The idea is to build a dictionary from words of the source language. After putting a word in the dictionary, if the word appears again it is replaced by some information regarding its position inside the dictionary. In this case the dictionary itself represents the model of the source. LZ77, LZ78, LZW fall in this category.

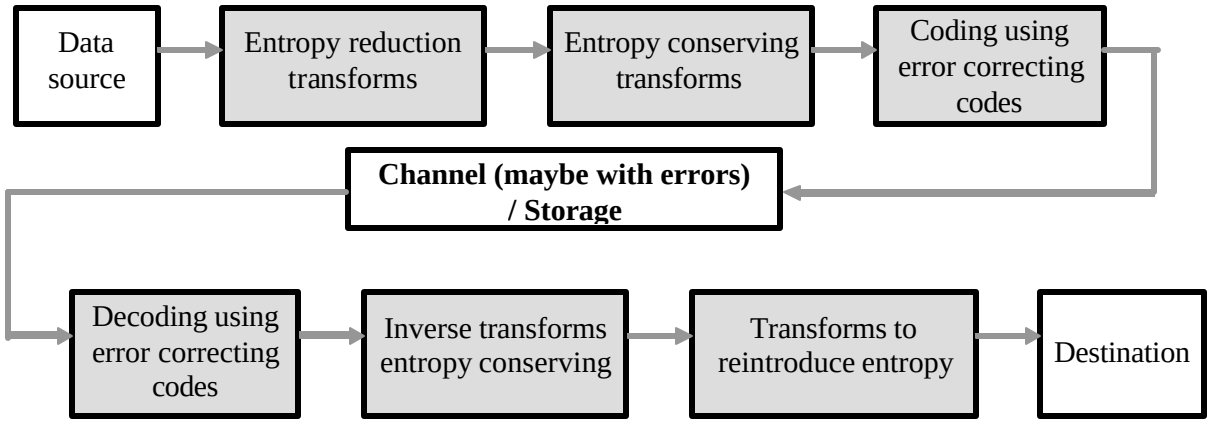

Figure 1.4 General compression-decompression model

3. General compression-decompression model

The most general model of a data compression system is given in Figure 1.4. The first transform is the one which reduces the entropy in order to eliminate the parameters of the messages that are not important to the final user (the lossy part). The next transform preserves entropy but reduces the redundancy in order to improve transmission or storage efficiency (the lossless part). In the case when transmission is done on a noisy channel some redundancy can be reintroduced by using error correcting codes. Because by these codes we do not achieve compression but the contrary we do not present them at all.

4. Some examples

In the following we present some simple compression examples, somehow empirical. These are not used in practice by themselves, in that simple form, but usual as steps in more complex schemes. We present them in order to exemplify the concepts of static/semistatic/adaptive and statistic/dictionary coding respectively. By analyzing the results we notice that none of the methods is useful in all situations, for some (unhappy chosen) streams we do not get compression at all, but instead we can have a growth of the representation.

4.1. Run Length Encoding - RLE

We present in the following a RLE version as implemented in the BinHex 4.0 format. This format uses, apart from the RLE coding, a standard comment, an imposed header and a 7 bits coding of the resulted binary stream. In the following we present only the RLE step, the others can be found in the BinHex 4.0 documentation.

The basic idea of RLE is to code the length of the repetitive sequences (called "runs"). We present an input data stream and the RLE encoded output data stream.

input stream								output stream RLE							
00	11	22	33	44	55	66	77	00	11	22	33	44	55	66	77
11	22	22	22	22	22	22	33	11	22	90	06	33			
11	22	90	33	44				11	22	90	00	33	44		

By analyzing the example we notice:

- Symbols that are not part of a "run" are not processed at all.
- The "runs" are replaced by a triple containing the repeated symbol, the 90_H marker and the length of the "run".
- In case of the 90_H marker appearance in the input stream it has to be replaced by a specific sequence (using the fact that the runs do not have a 0 length).

Even if in practice we can use a different coding approach, we have to solve 2 problems: coding of the "run" and the appearance of the marker.

We notice that the method becomes useful only when runs of length 4 or greater exist (shorter runs must be left unchanged) and the efficiency is reduced by the existence of the marker in the data stream (when the stream is in fact expanded).

As presented, the method is based on a **static** modeling of the data stream. If we want a **semistatic** modeling, we can make an analysis of the data stream in advance and choose as marker a symbol that appears less often in the stream. Certainly, the chosen marker must be sent to the decoder (for example, as the first byte in the compressed stream). If we want an **adaptive** modeling, we can make a continuous analysis of the data stream already transmitted and change the marker on-the-fly.

4.2. Simple statistic coding

To exemplify that method we propose to code the next data stream:

ABABACAD

The simplest coding is the coding on 2 bits for each symbol (corresponding to a 4 symbols alphabet) as:

A – 00, B – 01, C – 10, D – 11

So, the coded sequence becomes:

0001000100100011

having 16 bits.

Statistic coding consists in making the statistic of symbol occurrence and using of shorter codes for

high frequency symbols. So, we evaluate the frequencies of appearance:

A – 4, B – 2, C – 1, D – 1

and consider codes accordingly:

A – 0, B – 10, C – 110, D – 111

The encoded stream is now:

01001001100111

having only 14 bits, achieving some compression compared to the previous case.

The method falls in the semistatic category, at the first pass we build the model (codes) and only at the second pass we do the coding. For the practical implementation of the method the codes allocated to each symbol have to be transmitted to the decoder (not considered in the previous example), so that the gain achieved could be undermined. We haven't specified yet how to build such codes starting from frequencies (by example using the Shannon-Fano or the Huffman methods) but we keep in mind the idea of taking into account the data stream statistics (preferably unbalanced).

4.3. Dictionary based coding and MTF

For this example we consider the sequence ABCDDDCCBBAA which we code based on a dictionary. We build a dictionary with all the symbols (therefore containing ABCD) and as coding of a symbol we consider its index inside the dictionary. The static case dictionary is presented in the first columns of the next table:

Static dictionary		
Data stream	Dictionary	Index (code)
A	ABCD	0
B	ABCD	1
C	ABCD	2
D	ABCD	3
D	ABCD	3
D	ABCD	3
C	ABCD	2
C	ABCD	2
B	ABCD	1
B	ABCD	1
A	ABCD	0
A	ABCD	0

Adaptive dictionary MTF			
Data stream	Current dictionary	Index (code)	Updated dictionary
A	ABCD	0	ABCD
B	ABCD	1	BACD
C	BACD	2	CBAD
D	CBAD	3	DCBA
D	DCBA	0	DCBA
D	DCBA	0	DCBA
C	DCBA	1	CDBA
C	CDBA	0	CDBA
B	CDBA	2	BCDA
B	BCDA	0	BCDA
A	BCDA	3	ABCD
A	ABCD	0	ABCD

After coding we get the 012333221100 sequences. Obviously, for now we have no gain at all.

In the following we consider the „**M**ove **T**o **F**ront" (MTF) method for updating the dictionary. So, after coding of a symbol based on the dictionary, the symbol is "moved to front" inside the dictionary.

In this way we try to optimize the representation of symbols repeating themselves at short time intervals, their index having in that case lower values. The method falls in the adaptive modeling case based on a dictionary, and, as all the adaptive methods, consists in:

- Coding / decoding based on the current model (current dictionary)
- Updating of the model (dictionary)

In the last columns of the previous table we present the case of applying the MTF method on the data considered. At each step the current dictionary is the one obtained by updating that from the previous step. The resulted stream is 012300102030.

If we build the histogram of appearance for the generated indexes, we notice the unbalancing of the distribution in order to promote small indexes.

Without MTF	With MTF
0 ***	0 ******
1 ***	1 **
2 ***	2 **
3 ***	3 **

This fact allows us to apply to the obtained data stream a statistic method (as the simple one presented above), which will lead us to better results if the stream is unbalanced. We notice that the method is valuable as an intermediate step in a longer chain of processing (idea generally valid in data compression). Obviously, if the assumption regarding the repetition of the symbol is not justified, the MTF effect can be insignificant or even negative. The MTF method can be used also in the case where dictionaries contain strings of symbols (the case used in practice) not only individual symbols as presented previously for simplicity. Furthermore, we can take in consideration dictionary updates by moving only k positions to front (not to the first position) and moving to front only when a dictionary entry is repeatedly used (not at each use).