N.M.H. Riedmann

Methods of Data Compression

Abstract

This paper tries to give the reader a quick overview of some of the most frequent methods of data compression. It puts a focus on statistical methods while only noting a few character based methods where they are applicable as a combined method. As such a short overview can not discuss its topics in full detail its content is mainly based on the overviews given in Data Compression by Gilbert Held and Introduction to Data Compression by Guy E. Blelloch. For detailed information a reader might want to consult the individual publications on the different methods, like Lempel and Ziv's publications in IEEE Transactions on Information Theory.

1 Introduction

Starting to move from theoretical construct to real world use in the late 1950's and early 1960's, data compression is not only a way of using less resources in storing data, but also widely useful in data transfer. The possibility of improving the speed of data transfer is the reason for its early use in terminal-mainframe communication and its continued use as inbuilt functions in routers and similar devices. [1, 2]

There a three notable scopes of data compression which offer methods of different effectiveness. The most simple way of data compression is logical compression which is simply about reducing the size of data by intelligently choosing input and method of storage. I.e. using 130604 instead of 2013-June-03.

Character-based compression solely operates on the given data and uses utilises certain occurrences in the data to compress it.

Statistical compression utilises the knowledge about the occurrence of individual characters or patterns in the data in order to compress it efficiently. Of course modern efficient compression methods often utilise combinations of several different methods to achieve higher compression rates.

Furthermore Lossless and Lossy compression methods exist. All methods listed below are lossless methods, which are thus usable for compressing text. As Blelloch strains lossy compression which is mainly found in compressing pictures, video and music is not to be understood as resulting in missing parts but as simplifying the data. (i.e. reducing the resolution in areas) Thus the idea of lossy text compression would not result in missing words but in a simplified output text.

2 Statistical compression

More complicated but also more efficient than simpler compression methods, statistical compression relies on knowledge about the frequency of occurrence of certain parts of the data. (i.e. the letter 'E' is the most frequent in English text)

2.1 Information Theory

Like most authors do, I will start this section with a quick overview of some theoretical concepts. [1, 2] Of particular importance is the concept of Entropy , which is used in Information Theory to describe the average number of bits needed to represent each symbol in an alphabet. Entropy H can be defined as:

$$H_{avg} = -\sum_{i=1}^{n} P_i \log_2 P_i = \sum_{i=1}^{n} P_i \log_2 \frac{1}{P_i} \tag{1}$$

Where n is the total number of states or messages, i is the current message and P_i is the probability of a certain message. Based on the calculation of entropy in (1) it is possible to calculate the Redundancy R of information. For a system capable of transmitting on n levels, the unit of information equals $log_2 n$ [1] thus the redundancy becomes:

$$R = log_2 n - H_{avg} \tag{2}$$

Blelloch also explains the term of selfinformation of a message as defined by Shannon as: [1]

$$i(s) = log_2 \frac{1}{P(s)} \tag{3}$$

Note that Blelloch uses a different notation than the previous one, which is based on the one used by Held. Here s is a state or message out of a set of states S previously denoted by i as a state out of a number of states n. $P(s)$ is the probability of a certain message, previously denoted as P_i.

Selfinformation of a message is the number of bits of information in the message. It can be understood as the number of bits a message should be encoded in. In this context entropy can be understood as the probability weighted average of self information of all messages in a Set.

2.2 Huffman Coding

Huffman coding is a widely used statistical encoding method that offers some special benefits. Huffman codes possess a prefix property that ensures that no short code sequence will be reused as a prefix of a longer code sequence. Viewed as a tree structure the code represents a decision tree in which each state is either an empty decision state or a terminal state. Each edge is weighted with either a binary one or zero and each one is only examined once on each individual path which does thus represent a unique bis combination. Huffman codes are created by listing characters in descending order of their probability of occurrence. Then pairs of characters are combined into nodes with the smallest frequency, until the last two paths are combined in a unity node of probability 1. Then each branch is assigned a 0 or 1 arbitrarily but constantly. If 0 is used on the upper branch once, it must be used on every upper branch. The average number of bits per character is equal to

$$\sum_i = 0^n l_i * P_i \tag{4}$$

Where i is a Huffman code in a set of codes of length n, l_i is the length of the code and P_i as before, is the probability of the coded character. This size approaches entropy.

If the coded set is know to the decoder, the code is instantly decodeable bit by bit. If there are more than one possible way of building a Huffman code (if there are several entries with the same probability) the code which results in the smallest Variance e.g. which has the smallest difference in code lengths is to be preferred.

[1]A binary representation can be viewed as as a system "transmitting" on two channels 0 and 1

2.2.1 Possible applications

Run Length encoding compresses data by replacing continuous runs of characters with one occurrence of the character and a counter of how often it occurs in concession. *[compression indicator][character][counter]* Two obvious constraints of standard Run Length encoding are that a run of characters has to be at least three - or five if no unassigned codes exist - characters long in order for the code to be efficient and that the counter will have to be encoded in the same way as the characters and will this will have a maximum value of 256. Longer runs will need to be encoded in several runs of 256 characters.

A suitable real world application of Run Length Coding consists of transforming the entire set of messages into the form of *[character][count]* and than Huffman coding the characters and count values. For the count values a suitable frequency estimate is to choose $count = 1$ as the value with the highest probability.

Move-To-Front encoding is based on changing the position of each character from a character set in a previously fixed alphabet. On encounter each character is moved to the front of the alphabet, with all other characters moved backwards. After this is done the sequence of characters is encoded as integer values based on their position in the alphabet. This number sequence is then again coded in Huffman code.

2.3 Shannon-Fano Coding

Shannon-Fano coding works similarly to Huffman coding. Symbols are entered in descending order of probability then the list is divided into 2 equally probable parts until each resulting sub list after a split is also split fittingly. Now each digit in one group will be assigned a 1 and the other on will be assigned a 0. Afterwards a suffix bit will be added to each member of each two-member subset to distinguish them.

Like in Huffman coding the necessary *bits per entry* near entropy. Held notes that Shannon-Fano coding will be more efficient with sets with bigger variance, while Huffman codes will be more efficient for smaller variances.

2.4 Arithmetic Coding

In Arithmetic coding each possible sequence of entries is represented on a line from 0 to 1. A sequence of entries is assigned to an interval of size $\prod_{i=1}^{n} P_i$ starting with 1 and narrowing by P_i each message.

A sequence is then coded with the smallest binary representation of a number in the interval, which is itself represented as an interval. (i.e $.010 = [.010\overline{0}, .010\overline{1})$)

2.5 Residual coding

Residual coding is useful for message sequences whose order is meaningful (i.e. closeness in the sequence implies similarity of the content.) An encoder using residual coding guesses the next entry based on the context [2] and then stores the difference between the guess and the real value, called the residual. Then decoding the decoder makes the guesses based on the same context but directly adds the transmitted residual to produce the correct value.

2.6 Lempel-Ziv Algorithms

In general the Lempel-Ziv algorithm [1, 3, 4], when given a position in a file - in the following called 'cursor' - looks backwards for a set amount of characters until

[2] JPEGLS - lossless JPEG compression - utilises residual coding using the pixel to the left, above, to the left and above and to the right and above as context

the longest match for a string following the cursor is found. A code referring to the match is output and the cursor progresses past the match. Released in two papers, two variants of the algorithm, generally referred to as LZ 77 and LZ 78 were presented by Lempel and Ziv in 1977 and 1978. LZ 77, which is used in gzip, ZIP and V.42bis (a standard modem protocol) only looks backwards in a window of set size and outputs a triple of (position of occurs in the window, length of the match, next character after match). LZ 78, which is used in Unix compress and gif functions by placing new strings i in a dictionary initially filled with strings of length=1, if a prefix with $length = l(i) - 1$ exists in the dictionary. Each value in the dictionary has a distinctive code. The decoder then builds the same dictionary in the same way the encoder did an gets the same results. If the dictionary gets too big something has to be done. Implementations handle this differently. I.e. GIF throws away the entire dictionary if it reaches a certain size, while Unix compress deletes the dictionary if it deems it ineffective.

3 Conclusion

The field of data compression methods is vast and quickly growing ever since its beginnings. Even this very short overview can hardly fit on four pages although it does neither discuss methods in real detail nor looks into simpler methods at all. Furthermore while some relatively old methods like the Huffman Coding are still used, they are mostly used in variants and combinations. Recent conference proceedings like those of the *IEEE Data Compression Conference* focus heavily on compression methods for specialised purposes and the efficient combination of methods. The current state of the field is thus difficult to grasp and an overview of methods can not be more than a look at the tip of the iceberg.

References

[1] Guy E. Blelloch. Introduction to data compression. Book draft,Department of Computer Science, Carnegie Mellon University, 2013.

[2] Thomas R.Marshall Gilber Held. *Data Compression.* John Wiley and Sons Ltd., 1991.

[3] Abraham Lempel Jacob Ziv. A universal algorithm for sequential data compression. *IEEE Transactions On Information Theory*, 23(3):337–343, 1977.

[4] Abraham Lempel Jacob Ziv. Compression of individual sequences via variable-rate coding. *IEEE Transactions On Information Theory*, 24(5):530–536, 1978.