Generalized Compression Algorithms in Discrete Mathematics

Jeffrey Sorrentino

University of Arizona

Generalized Compression Algorithms in Discrete Mathematics

Abstract

All forms of digital communication have some basis in discrete math. Our modems fax machines and cell phones all use compression algorithms that are deeply rooted in discrete math, not to mention the need for compression over network transmission and conservation of disk space. While this paper's main focus is on compression techniques for network and disk conservation, in most cases similar techniques can be used in compression of most data.

I will start by discussing Statistical Encoding and the Noiseless Coding Problem. Statistical Encoding is the process of finding repetitive sequences within a stream, be it a series of ASCII characters or a binary file. Statistical Encoding algorithms will map a set of keys to words or bit patterns with high probability equivalent to smaller keys. Noiseless Coding Problem shows up while making substitutions for words of length n and replacing them with a much smaller key. There are many approaches to doing this, some much more efficient then others. The Noiseless Coding Problem is such that when keys are inserted into a stream, if the keys are not chosen correctly one might have unaligned or undecipherable bit patterns in the compressed data. When uncompressing such data unwanted character representations might appear, thus the data has become noisy and distorted.

Introduction

The need for compression arises from the limitations of space and bandwidth current and future systems have. Although the capacity for larger datasets and higher bandwidth is ever growing, so is the need for it. Be it a network throughput problem or a necessity to conserve disk space, data compression is embedded within most communication transmissions and incorporated into a large portion of today's standard computer applications. The question arises, how do we compress data effectively without producing overwhelming overhead and undecipherable uncompressed data. Lossless Compression and The Noiseless Coding Problem address the later part of this question.

The implications of probability and set theory, which are based in discrete math, are quite evident within the context of compression, but for now I will postpone any direct references. However, deeper investigation of algorithms and the use of different alphabets, be it ASCII or binary, will revealed the roots of compression in discrete math.

Beginning Compression

Null Suppression and Bit Mapping are some of the earliest and simplest forms of compression. Null Suppression is a pretty straightforward idea, as is sounds you are compressing the data by suppressing null, zeros or some form of blank space or null character depending on the implementation. This technique takes advantage of the fact that most files can contain a large number of blanks and zeros. This method is applicable to data sets containing fixed sized units, such as words or bytes.

Example:

> Assume we are dealing with data set containing a large number of 0 in which implementing Null Suppression is worthwhile and that each item in the collection is a fixed sized.
>
> Original: | data | 0 | 0 | 0 | 0 | 0 | 0 | data | 0 | 0 | 0 | data | 0 | 0 | 0 | 0 | 0 |
>
> Compressed: | 1000000100010000 | data | data | data |
>
> **[1]**

To implement the bit-map we appended the front of the collection with a key. We drop all of the zeros and turned on 1's corresponding to the data in the addresses to follow and 0's where the groups of zeros were within the collection. By suppressing the zeros using the Bit Map we when from size = 16 to size = 4.

Null Suppression can also be implemented using the Run Length technique, where a special character is inserted along with an integer to replace a run of nulls.
Example:

Original Data: A10000X02500000N00000COST

Compressed Data: A1#4X025#5N#5COST

[1]

The choice of characters to use for replacement is quite trivial, any character which is not commonly used, is acceptable. If however while replacing nulls in a collection the chosen character is found in the stream, just replace it with a double instance of it.
Example:

Original Data: A1#400000000034

Compressed Data: A1##4#934

This same basic technique is also used to do Pattern Substitution. Pattern Substitution looks for repeating patterns in files and replaces them with the key character and a representative number. To do this however one must also start to compute probabilities to truly be effective. This is considered to be a type of Statistical Compression and thus needs the knowledge of the following functions.

Lossless Compression and The Noiseless Coding Problem

Lossless Compression or reversible compression is the concept in which the decompressed data approaches exactly matching the original data. There are Lossless and there are Lossy-Distortion algorithms. In some settings there is an acceptable amount of data loss within certain function to produce faster and lighter compression. As is the case for encoding and compressing some image files. By taking advantage of the limitations of the human eye, image compression can eliminate certain pixel patterns, which will not be noticeable, but in turn will reduce the size of the image file.

Although there are statistical coding algorithms like Huffman Codes and Shannon-Fano that approach Noiseless Coding it seems they are still subject to possible data loss.
The Counting Argument states that any compression method is limited. The compression method cannot losslessly compress all files of size N since some of the N files are random.

Proof: (by contradiction on $N > 0$)

Assume that an algorithm exists that can compress losslessly all files of size N bits. There are 2^N files of size N bits, and compressing them with our algorithm must produce 2^N files of sizes < N. There are 2^{N-1} files of sizes $N - 1$, 2^{N-2} files of size $N - 2$ and so on down to $2^{N-N} = 1$ file of size $N - N = 0$. This is equivalent to.

$$2^{N-1} + 2^{N-2} + \ldots + 1 = 2^N - 1$$

Instead of 2^N, so there must be at least two size-N files compressed to the same smaller size file which means the algorithm is lossy. **[7]**

Although the Counting Argument is somewhat discouraging, it is only an issue when multiple files are being compressed. Algorithms are considered optimal when they approach the Noiseless Coding.

The Noiseless Coding Problem is described as follows: A random variable takes on the values $\{ x_1, \ldots, x_m \}$ with probabilities $\{ p_1, \ldots, p_m \}$ respectively. Code words $\{ w_1, \ldots, w_m \}$ of lengths $\{ n_1, \ldots, n_m \}$ respectively, are assigned to the symbols $\{ x_1, \ldots, x_m \}$ **[1]** The Code words are combinations of characters taken from a code alphabet usually in our case $\sum = \{0,1\}$. The problem itself is to construct a uniquely decipherable code make up of the code words, which minimizes the average code word length.

Statistical Methods

As described in the Noiseless Coding Problem the basic idea is that any file, e.g. text and binary files have repeating letters, strings, and sub strings. Because of this a Pattern Substitution will clean up a file quite nicely. By generating the probabilities of all possible characters in the file and even some frequently repeated strings-sub strings you can do a substitution on the entire file with decipherable keys. Arranging the probabilities in descending order, assign the smallest key-code words to the highest probabilities.

The next problem is assuring decipherable keys. One of the safest ways to ensure decipherability is to implement instantaneous codes. Instantaneous codes assure a bit stream will not be misinterpreted by defining unique prefixes for each key. Take a file with say 4 characters.

Char	Key		Char	Key
x_1	0		x_1	0
x_2	010		x_2	100
x_3	01		x_3	101
x_4	10		x_4	11

The first column of the previous table is not uniquely decipherable, for example, if the string 01001001 was created there is know way to decipher even the first 3 bits whether they are x_1, x_2, or x_1. Where as the second column of unique prefixes in any sequence will not change the message.

There are two Statistical Coding Methods worth discussing in some detail. The first is the Shannon-Fano Code. This is the original implementation of the instantaneous codes and entropy coding. Entropy is defined as the average of the probabilities from a source X given by: $H(X) = \sum_i Pi \log Pi$ the log is usually taken to the base of 2 in which case the units of entropy or bits. **[5]**

As stated before, the Shannon-Fano algorithm is unambiguously, instantaneously decodable. Shannon-Fano code reaches an efficiency of 100% only when the source message probabilities are negative powers of 2, given by: $Efficiency = \frac{H}{l} * 100$ where l is the average length of the code word. **[6]** The coding of the algorithm is quite simple.

1.) As before, arrange the source message probabilities in descending order.
2.) Divide the message set into two subsets of equal or almost equal, total probability and assign a zero as the first code digit in one subset, and a one as the first code digit in the second.
3.) Continue this until each subset contains only one message.

[6]

This is usually accomplished using a simple recursive algorithm.

The second instantaneous code worth noting is the set of Huffman codes. Huffman codes are very similar to Shannon-Fano code and are also considered a minimum-redundancy code. The construction of the data set however is different. instead of splitting the set of probabilities, a tree is built in order to obtain a prefix exclusive set of keys. There are several ways to build Huffman trees, and each way creates a different Huffman code.

1.) To build a Huffman tree you must once again arrange the probabilities in descending order.
2.) Combine the two lowest probabilities and continue this procedure, always putting the higher probability branch on top until unity is reached.
3.) Assign a zero to the upper member of the pair and a one to the lower or vice versa.
4.) Traverse the path from each message probability to the unity point, recording the ones and zeros along the path.

5.) For each message write the one-zero message obtained from right to left. **[6]**

Current variations of the Huffman codes are still some of the most popular to date.

Although compression codes and algorithms do not use direct mathematical functions to compress the desired files and data, discrete math still plays an invaluable role in the compression of data. Set theory, probability, and function mapping are just a few the ways compression relies on discrete math.

Bibliography

[1] ARONSON, J.: (1977). Data Compression – A Comparison of Methods. NBS Special Publications 500-9, US Department of Commerce, National Bureau of Standards.

[2] DAVISSON L.D., GREY R.M.: (1976). Data Compression. Dowden, Hutchinson & Ross, Inc. Stroudsburg Pennsylvania.

[3] HUFFMAN, D.A.: A Method for the Construction of Minimum Redundancy Codes. Proc. IRE, 40(9), 1098-1101 (1952).

[4] HELD G.: (1991). Data Compression. John Wiley & Sons LTD. New York.

[5] GIBSON, D.G.: (1997). The Communications Handbook. CRC Press, IEE Press

[6] LYNCH, T.J.: (1985). Data Compression, Techniques and Applications. Lifetime Learning Publications, Belmont, California

[7] SALOMON**,** B.: (1998). Data Compression. Springer. New York