ADVANCED COMPRESSION TECHNIQUE FOR RANDOM DATA

Blaise M Crowly
Department of Information Technology, KMEA Engineering College, Aluva, Kerala India

ABSTRACT

This paper proposes a technique to compress any data irrespective of it's type. Compressing random data especially has always proved to be a difficult task. With very less patterns and logic within the data it quickly reaches a point where no more data can be represented within a given number of bits. The proposed technique will allow us to compress irrespective of it's pattern or logic within, and represent it. Further it will permit the technique to be again applied to the already compressed data without having any change in the possible compression ratio. While data can't be compressed further than a limit the technique will rely on representing the data as a position in any computationally easy number series that extends to infinity provided it have high enough deviation among it's digits. Only few markers that are linked to the position is saved, rather than representing the original data. The system then use these markers to guess the position and derive the data from it. The procedure is however, computationally intensive and as of now can raise questions of data corruption but with more computing power, efficient algorithm and proper data integrity checks it will be able to provide very high compression ratios in the future
.

KEYWORDS

Data compression, Data representation, Random data, Losless compression, Compression ratio

1. INTRODUCTION

Storage has always been a very scarce resource in computers, pushing more and more data into smaller spaces have become a necessary need today. The amount of data being produced today is very large and will only grow in the coming years. This makes the need for better compression methods vital for us. While many of the existing compression and efficient they soon hit a limit1 beyond which no more data can be effectively represented within a given amount of bits. This is worst in case of random data with very less or no patterns or logic within it.

The existing compression methods work primary by

1. Removal of unwanted / unimportant data.
2. Reducing redundant data..
3. Predicting / generating patterns.

These methods work perfectly as long as there are patterns within or there is expendable data. The result is that they are heavily dependent on what kind type of data they can effectively compress and different methods have to be applied depending on the type of data.

There is also a very clear difference in the effective compression ratios depending on the type of data, for example text files can be compressed in a lossless mode to much smaller size than multimedia can be. Hence we require a efficient universal compression technique that that will work on any form of data and can be used again on the compressed file to further compress it.

2. Objectives

To design a data representation method and required data compression and decompression technique for data so compressed such that it should.Should strictly provide lossless compression.

• Not use dictionaries.
• Works on any type of data.
• Is independent of the data density.
• Is independent of the data randomness.
• Not need to use demarcating bytes to separate compressed and non-compressed data.
• Provide high compression ratios.
• Can be applied repeatedly to a file without affecting the possible compression ratio.

3. Approach

The idea is to represent data as the prime number location between 0 and infinity within a computable series, the series does not have to be a normal number but have to be non repeating, have a large enough deviation between digits and should be computationally easy. Any set of data can be represented as a value in between 0 and infinity, for example a set of bytes {213, 110, 180, 177} can be condensed as 213110180177 and then be represented as the 213110180177th position on a number series or a computable number, who's n^{th} digit can be calculated in a computationally easy way, yet have high deviation and, extends to infinity. Few markers are then generated to to help us identify this position from the number series which are used later to guess the position value. These markers along with few some other information is stored.

This removes the need to actually store the full representation of the data hence allowing us to overcome the limit to which the data can be compressed. To make this representation possible and efficient we will make sure that the data that needs to be represented on the number series is a prime number and make it into one if it is not. This will limit to the number of positions the system need to guess contrary to when all values are possible. The decompression process will be longer than the compression process.

4. Compression

Consider a pool of bytes $P = \{p : 0 \leq b \leq 255\}$ such that there is no repetitive or logical pattern within it, a number $lim = \{2 \leq lim \leq size(p)\}$, a large integer number N initially set to 0, a empty data set C to hold the compressed data and a function Pat(n) that generates the n^{th} digit of a computable number or number series.

Now while $n < lim$ we combine one value of one byte each from P , say P_n , to N by doing $N = ((N \times 1000) + P_n)$ and increment n, hence if p_n is a double or single digit value we will have a 0 or 00 before the value respectively. N is first initialized 1 or any other single digit number and then the value is combined to it as per the formula above if p_0 is a single or double digit number, this can be called Head Padding. Hence we achieve a large integer number who's group of every 3 digits represents a value of the bytes in our pool in their original order.

$P =$

123	100	182	251	222

$N = 123100182251222$

Figure 1. A large integer is obtained by combining values of our bytes that we need to compress
.

$P =$

184	2	211	75	14

$N = 184002211075014$

Figure 2. N has 00 or 0 before the values if the new value to be combined is single or double digit respectively.

$P =$

17	222	210	75	144

$N = 1017222210075144$

Figure 3. N is initialized with 1 as Head Padding here as the first byte does not 3 digits.

Now N should represent a large number, eg: if lim was 10 we should have a 30 digit number assuming that the first byte contained a triple digit value (so that noHead padding was used). We check if the obtained number N is a prime number if it is not we append it with further 3 digits to make it a prime number, this can be called as Tail Padding.

$P =$

155	21	182	175

$N = 155021182175003$

Figure 3.Tail Padding. Here 003 is added to make N a prime number.

Once a prime number is available we derive two markers M_0 and M_1 using the Pat(n) function where n is N and N + 1 hence making M_0 and M_1 the N^{th} and $(N + 1)^{th}$ digits of the computable number or number series respectively. Now we can represent the data using the tuple V = {n, M_0 , M_1 , H, T } where H is a flag that is true if Head Padding was used T is a flag that is true if Tail

Padding was used. n, M_0 and M_1 being the number of bytes combined into N and the markers respectively.

5. Data Representation

To represent the converted information we shall store the markers, the number of bytes represented and the padding used if any. The only information that is requires to be stored to retrieve the information is the data within the tuple V = {n, M_0 , M_1 , H, T }

5.1. Two Byte Representation

The structure for 2 bit representation consist of 4 parts.

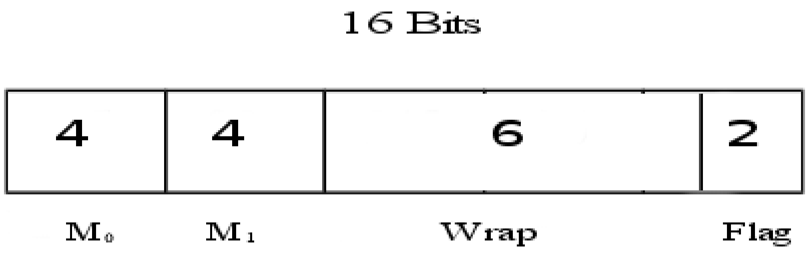

Figure 4.Two byte data representation

In this structure the first 8 bits are used to represent M_0 and M_1 which are the markers. The 6 bit Wrap can store between 0 and 65 which represent the number of bytes we combined to form N . In the standard representation it shall be equal to n in V, this can be expanded to larger numbers as discussed later. The last 2 flag bits are used to represent the kind of padding used. The flag is set to 00 if neither Head Padding nor Tail Padding was used in N , set to 01 if Head Padding alone was used, 10 if Tail Padding alone was used and 11 if both were used.

5.2. Three Byte Representation

The 3 byte representation contains two extra markers M_2 and j which will be used to improve data integrity and processing efficiency during de-compression.

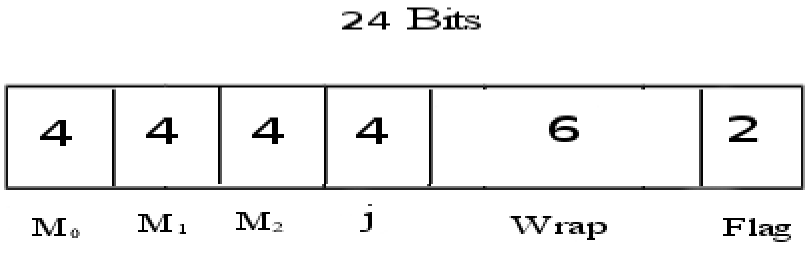

Figure 5.Three byte data representation.

The 3 byte representation is to be used when the Wrap represents a very large N. j is the first digit from the right in N .

5.3. Expansion Method

While the Wrap can effectively represent values between 0 and 65 it may not be always enough to effectively represent the values when we generate larger N values which can be more than the combination of 65 bytes. In such case the Wrap can represent number of data units, for example a base unit of l can be assumed for the entire file we are operating on and then if the Wrap value is r then the actual number of bytes combined to form N will be $l \times r$

6. DE-COMPRESSION

The de-compression process is more computationally intensive than the compression process as the logic involves guessing the prime number N from a range. The more efficiently the system is at deriving prime numbers from within a range the faster will be the de-compression process. The markers will be then used to check if the prime number obtained is the actual N that is required.

The Wrap value is used to calculate the length L of N , where $L = 3 \times r$ in general case or $L = 3 \times r \times l$ if the expansion method is used. If the flag bits are not 00, L is incremented by 1, 3 or 4 depending on the flag bits. 1 if flag bits are 01, 3 is flag bits are 10 and 4 if flag bits are 11.

Once the L is calculated the range between which N can occur can be derived as between 10L and 10L+1 . If the 3 byte representation is used the value of j can be used to further limit this to 1/10th of the total range. The possible combinations are much lesser than the entire range as the number.

For n in set $T = \{t : t$ is a prime number and $(10L \leq t < 10L+1)\}$, it is checked if P at(t) = M_0 . If P at(t) is found to be equal to M_0 , and if they are equal P at(t + 1) is checked with M_1 to confirm. In the 3 byte representation P at(t + 2) is also checked with M_2.

Once the number t is confirmed to be prime the flag bits are checked to identify the padding used. If Tail Padding is used the rightmost 3 digits of t are ignored and if Head Padding was used the leftmost digit will be ignored. Three digits at a time from the left side is taken from t and then stored into a byte as integer values from 0 to 255

The possible numbers that have to be checked can further be reduced using the property that, Ignoring the padding digits the 1st , 4th , 7th and every 3rd digit from then can only have 0, 1 or 2 as possible digits as the bytes values that were combined can only have values between 0 and 255. Call them the *Front-Digits*. Further if the Front-Digit is 2 the very next digit after a Front-Digit can only be among 0, 1, 2, 3, 4 or 5.

7. CONCLUSION

As we are producing more and more data each day we have a need to pack more of into smaller spaces for multiple reasons. The existing compression methods are extremely dependent on the type of data they are handling causing them to be very limited in their scope of use. The method suggested in this paper will allow data to be compressed irrespective of their type and can be repeatedly compressed if required.

While todays availability of computing power may not allow practical use of this technique as we develop faster and more precise computation it may be possible to apply this at a large scale even for everyday use. Combined with more computing power, better error detection technique designed specifically for this tech nique, a specially designed computable series and improvements in the data representation methods introduced here we may be able to effectively compress largeamounts of data effectively and repetitively to within a reasonable time in the future.

ACKNOWLEDGEMENTS

This paper was made possible with the help support and guidance from everyone including my parent, friends, teachers and fellow programmers and researchers. I thank everyone who tool time to discuss the concept with me and provide insight.

I would like to thank my institute, it’s staff and management for all the support and guidance in making this paper possible.

To the visionaries who traveled ahead of time and sacrificed the re time, money and life itself so that the world today can enjoy the technology and comforts we do today.

I thank Ms. Binny Verghese for her guidance and support to me when I was her student, to this day the best computer teacher I have had.
I also thank the open source Community for their timeless and selfless contribution to the world.

REFERENCES

[1] A Mathematical Theory of Communication. C. E. SHANNON The Bell System Technical Journal,Vol. 27, pp. 379423, 623656, July, October, 1948.
[2] A Concise Introduction to Data Compression. Salomon, David (2008). ISBN 9781848000728.
[3] An Improved Data Compression Method for General Data. Mahmud, Salauddin (2012) International Journal of Scientific & Engineering Research
[4] On Computable Numbers With an Application to the Entscheidungsproblem. Alan Turing (1936)
[5] The Generation of Random Numbers that are Probably Prime. Pierre Beauchemin and Gilles Brassard - Journal Of Cryptography (1988).

Authors

Blaise M Crowly A student, entrepreneur and independent researchers in information theory and cryptography. Currently doing his undregrad studies at KMEA Engineering College Kerala and CEO of Xincoz Labs a startups doing R&D in security and data communication.