A REVIEW OF LOSSLESS DATA COMPRESSION ALGORITHMS

REVIEW ARTICLE

Aditya Meharia
School of Computer Engineering
Kalinga Institute of Industrial Technology
Bhubaneswar, India 751024
adityameharia14@gmail.com

Junaid H. Rahim
School of Computer Engineering
Kalinga Institute of Industrial Technology
Bhubaneswar, India 751024
junaidrahim5a@gmail.com

May 7, 2020

ABSTRACT

This work aims to provide an introduction to the domain of Data Compression in Information Theory and a comprehensive review of the existing literature in the field of algorithms for Lossless Data Compression. We identify and discuss the potential opportunities, barriers and the future scope of the field. We also review data compression methods used for text, image, video and audio data.

***K*eywords** Data Compression · Algorithms · Lossless Data Compression · Information Theory

Submitted to **Dr. S.K.S. Parashar**

1 Introduction

A rapid growth in modern communication technology led an explosion in the amount of data we transmit and store. Large files consume significant resources for transmission as well as storage. Due to this exponential increase in the size of the data we transmit, researchers developed algorithms that can be used to compress the data to save storage space as well as transmission time. Data Compression is a process by which we encode the input data into a representation that occupies fewer bits than the original input. This encoded representation is transmitted and decoded back to the original form at the destination. Data compression algorithms are broadly classified into two classes viz **Lossless Compression** and **Lossy Compression** algorithms. We will be only covering Lossless Data Compression algorithms in this review article.

1.1 History of Data Compression

The field of data compression gained huge significance in the 1970s after the surge in the usage of the internet. The need to reduce transmission time pushed computer scientists to find new ways to compress information. Although, the very earliest form of compression was Morse Code, invented in 1838, in Morse code the letters 'e' and 't' from the english language were given shorter codes as they have a high probability of occurrence.

Later with the advent of mainframe computers, Calude Shanon and Robert Fano invented Shanon-Fano coding in 1949[1], the algorithm assigns shorter codes to symbols with high probability resulting in a shorter way to represent the data. In 1952, David Huffman, one of the students of Robert Fano at MIT studying information theory took the option to write a term paper when given a choice between taking a final exam or writing a paper. Huffman was interested in finding the most efficient way to assign prefix codes to a set of symbols, after months of work Huffman published Huffman Coding in his paper "A Method for the Construction of Minimum-Redundancy Codes"[2], Huffman coding was an improvement over Shanon-Fano coding in terms of efficiency as it assured the assignment of the shortest possible codes to the given symbols. The early implementations of Shanon-Fano coding and Huffman coding were done using hardcoded codes, later in the 1970s, software compression was implemented and Huffman Codes were dynamically generated depending on the input data.

In 1977, Abraham Lempel and Jacob Ziv published[3] their groundbreaking LZ77 algorithm and later the LZ78 algorithm, these algorithms used a dictionary to compress data. The popular UNIX operating system used a compression utility based on LZW which was a slight modification of the LZ78 algorithm. Later the UNIX community adopted the DEFLATE based gzip and Burrows-Wheeler transform based bzip2 formats mostly due to their open source nature[4]. It was a beneficial decision in the long run as gzip and bzip2 have consistently given higher compression ratios compared to the LZW format.

In 1989, Phil Katz released the PKZIP format, later in 1993 Katz updated the format and named it PKZIP 2.0, he based it on the DEFLATE algorithm, the .zip format used so extensively in today's day is based on the PKZIP 2.0 format. ZIP and other DEFLATE based formats were extremely popular till the mid 1990s when new and improved formats began to emerge. In 1993, Eugene Roshal released his WinRAR utility which uses the proprietary RAR format. The RAR format is one of the most used formats to compress data and share it via the internet. In 1999, UNIX adopted the 7-zip or the .7z format, this was the first one capable enough to challenge the dominance of the .zip and .rar formats as .7z was not limited to just one compression algorithm, but could instead choose any of bzip2, LZMA, LAMA2 and PPMd algorithms among others.

1.2 Overview of Lossless Data Compression Techniques

Lossless Compression algorithms are a class of algorithms that can reproduce the original content from the encoded respresentation without any loss of information, the data before compression and after decompression is exactly the same. Lossless compression is used in a variety of fields where it is important that the original and decompressed information be the same. The GNU tool gzip uses lossless algorithms for the ZIP file format.

Lossless compression algorithms usually have a two step procedure.

1. A statistical model of the input data is generated. This usually assigns a probability of occurrence to pieces of input data. For example, if the input data is piece of text, then the model would be the probabilities of occurance of each alphabet
2. A coding system uses this model to map the data in a way that the pieces with high probability of occurrence are assigned a shorter code than those with a low probability of occurance

The probabalistic model is usually generated in two ways, a static way and an adaptive/dynamic way. In the static approach, the data is analysed and the probability model is generated before starting the encoding procedure, this is a modular and simple approach but doesn't perform well for heterogeneous data since the approach forces the use of a single model for the all the data. In the dynamic method, the model is updated while compressing the data. The encoder and decoder start with a trivial model in the initial state, thus performs poorly on initial data, but as the model adapts to the data, the performance improves. Most efficient compression techniques usually employ an adaptive model. There are various ways to achieve losseless compression namely Run Length Encoding (RLE), Lossless predictive coding (LPC), Entropy coding and Arithmetic coding etc.[5][6]

2 Prefix Codes and Entropy

A prefix code is a "code" system in which the codes given to each character is not the prefix of the code given to any other character i.e. it follows the prefix property [7]. For example $\{2, 42, 12\}$ is an example of a prefix system whereas $\{2, 42, 12, 21\}$ is not because '2' is the prefix of '21'.

Prefix codes are also known as prefix-free codes, prefix condition codes and instantaneous codes. They are uniquely decodable codes that is no two codes will have the same value on decoding. Prefix codes can be both fixed length and of variable length. It does not require between words to seperate them. Variable length preifx codes have been used extensibly in Huffman and Shannon coding and are still used in modern compression algorithms along with arithmetic coding.

Entropy denotes the randomness of the data that you are passing as input to the compression algorithm. That means the more random the text i.e. higher entropy is, the lesser you can compress it. It represents an absolute limit on the best possible lossless compression of any communication: treating messages to be encoded as a sequence of independent and identically distributed random variables. Shannon's source coding theorem shows that, the average length of the shortest possible representation to encode the messages in a given alphabet is expressed as follows

Given a random variable X, with possible outcomes x_i, each with probability $P_X(x_i)$, the entropy $H(X)$ of X, where b is the base of the logarithm is as follows:

$$H(X) = -\sum_{i=1}^{n} P_X(x_i) \log_b P_X(x_i) = \sum_{i=1}^{n} P_X(x_i) I_X(x_i) = E[I_X]$$

3 Shanon Coding

It is named after its creator **Claude Shanon**, the technique was used to prove Shanon's noiseless coding theorem in his 1948 article "A Mathematical Theory of Communication"[1]. Even though being suboptimal, the method was a first of its kind. This method is credited to have given rise to the entire field of Information Theory. Some of the most efficient compression algorithms today are usually an exntension of shanon's method

Shanon Coding is a method to generate prefix codes for a given piece of data. It is done using the occurrence probabilities of the pieces of data. First the probabilities p_i are arranged in descending order, then each piece is assigned a code which is the first l_i digits of binary representation of the cumulative probability till that piece of data.

Given that the probability of occurrence is p_i, the cumulative probability is expressed as

$$\sum_{k=0}^{i-1} p_k$$

where $l_i = \lceil \log_2 p_i \rceil$

It is a suboptimal algorithm, it does not give the lowest possible code word length.

The following is an example of assigning prefix codes to compress the string "lossless data compression"

i	a_i	p_i	$p_c = \sum_{k=0}^{i-1} p_k$	Binary Representation	$l_i = \lceil \log_2 p_i \rceil$	code
0	s	0.24	0.00	0.00000000...	2	00
1	o	0.12	0.24	0.00111101...	3	001
2	e	0.08	0.36	0.01011100...	3	010
3	<space>	0.08	0.44	0.01110000...	3	011
4	a	0.08	0.52	0.10000101...	3	100
5	l	0.08	0.60	0.10011001...	3	100
6	i	0.04	0.68	0.10101110...	4	1010
7	d	0.04	0.72	0.10111000...	4	1011
8	t	0.04	0.76	0.11000010...	4	1100
9	c	0.04	0.80	0.11001100...	4	1100
10	m	0.04	0.84	0.11010111...	4	1101
11	r	0.04	0.88	0.11100001...	4	1110
12	p	0.04	0.92	0.11101011...	4	1110
13	n	0.04	0.96	0.11110101...	4	1111

4 Huffman Coding

Named after its creator David A. Huffman. Although C.E. Shannon [1] and R.M.Fano [8] developed ensemble coding procedures to prove that the average number of binary digits required per message approaches from above the average amount of information per message,it was not optimum. Kraft [9] had derived a coding method which gives an average code length as close as possible to the ideal when the ensemble contains a finite number of members. However Huffman was able to derive a definte procedure for this. The output of the Huffman table given us a prefix-variable code table which consists of the source symbol and the encoded symbol.

Like Shannon coding, Huffman algorithm also tries to minimize the entropy by assisgning shortest codes to the charcters which occur most frequently. The algorithm works by creating a binary tree (using a min heap) which can have either leaf nodes or internal nodes. The leaf node contains the weight and the symbol whereas the internal nodes consists of weight and links to the two child nodes. The bit '0' is used to reprsent the left child of an internal nodes whereas the bit '1' is used to represent the right child.

Steps to build a Huffman Tree:-

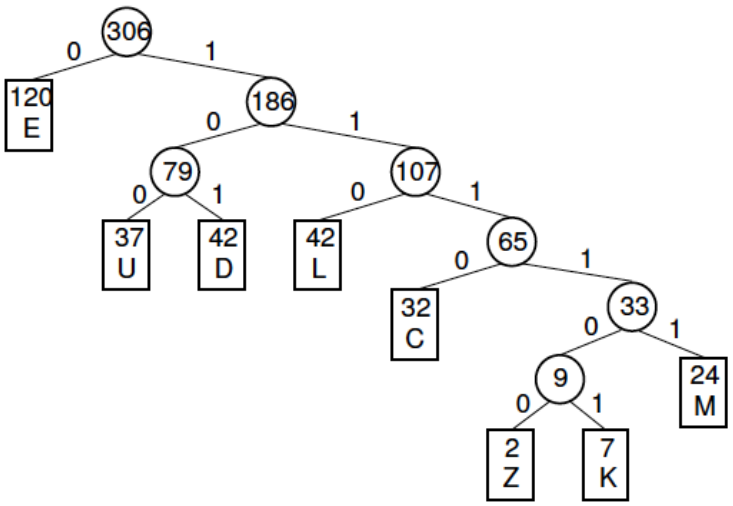
1. The process begins by traversing the input string and finding all the unique charcters along with their frequencies.
2. Then create a leaf node for each symbol and their frequencies(weight) and add build a min heap of all the leaf nodes.
3. Take the two nodes with the minimum weight and create a new internal node with the weight equal to the sum of the frequencies of the 2 nodes.Make the first extracted child as the left node and the second extracted child as the right node and add it to the min heap.
4. Repeat the steps until the heap contains only one node.

Letter freqency table

Letter	Z	K	M	C	U	D	L	E
Frequency	2	7	24	32	37	42	42	120

Huffman Code

Letter	Freq	Code	Bits
E	120	0	1
D	42	101	3
L	42	110	3
U	37	100	3
C	32	1110	4
M	24	11111	5
K	7	111101	6
Z	2	111100	6



Decoding the Huffman tree is very simple, traverse the tree node by node as each bit is read from the input stream (reaching a leaf node necessarily terminates the search for that particular byte value).

For example, decoding an encoded string can be done by looking at the bits in the coded string from left to right until a letter decoded. 10100101 $\Rightarrow$ DEED

5 Lempel-Ziv(lz) Compression Methods

Also known as the LZ family,it initially consisted of LZ-77 and LZ-78 which was published by Abraham Lempel and Jacob Ziv in 1977[3] and 1978[10].Also known as LZ-1 and LZ-2 respectively,both are theoretically dictionary coders.These algorithms formed the basis for a lot of variations like LZW,LZSS,LZMA and also for a lot compression schemes like DEFLATE which has been discussed later in this article.

5.1 LZ-77 Algorithm

LZ77 algorithms achieve compression by replacing repeated occurrences of data with references to a copy of that data existing earlier within the uncompressed data stream. A match is encoded by a pair of numbers called a length-distance pair, which is like the statement "each of the next length characters is equal to the characters exactly distance characters behind it in the uncompressed stream".

To spot matches, the encoder must keep track of some amount of the recent data, like the last 2 kB, 4 kB, or 32 kB. The structure during which this data is held is named a sliding window , which is why LZ77 is usually called sliding-window compression. The encoder must keep this data to find matches, and therefore the decoder must keep this data to interpret the matches the encoder refers to. The larger the sliding window is, the longer back the encoder may look for creating references.

While encoding, for the search pointer to continue finding matched pairs after the end of the search window, all characters from the primary match at offset D and forward to the end of the search window must have matched input, and these are the (previously seen) characters that comprise one run unit of length LR, which must equal D. When search pointer proceeds past the search window and forward, as long as the run pattern repeats within the input, the search and input pointers are going to be in sync and match characters until the run pattern is interrupted. Then L characters had been matched in total, L > D, and therefore the code is [D, L, c].

Upon decoding [D, L, c], again, D = LR. When the primary LR characters are read to the output, this corresponds to one run unit appended to the output buffer. At this moment , the read pointer might be thought of as only needing to return int(L/LR) + (1 if L mod LR != 0) times to the beginning of that single buffered run unit, read LR characters (or maybe fewer on the last return), and repeat until all of the L characters are read. But mirroring the encoding process, since the pattern is repetitive, the read pointer need only trail in sync with the write pointer by a fixed distance which is equal to the run length LR until L characters are copied to output in total.

Algorithm 1 LZ-77

```
while input not empty do
    prefix : longest prefix of input that begins in window
    if prefix exists then
        j : distance to start of prefix
        k : length of prefix
        a : char following prefix in input
        if j := 0 then
            k := 0
            a := first char of input
        end if
    end if
    output(j, k, a)
    s := pop k + 1 char from front of input
    discard k+1 char from front of window
    append s to back of window
end while
```

5.2 LZ-78 Algorithm

The LZ78 is also a dictionary-based compression algorithm that maintains a dictionary. The encoded output consists of two elements: an index pertaining to the longest matching lexical entry and therefore the first non-matching symbol. The algorithm also adds the index and symbol pair to the dictionary. When the symbol is not yet found in the dictionary, the codeword has the index value 0 and it's added to the dictionary as well . With this method, the algorithm constructs the dictionary. LZ78 algorithm has the power to capture patterns and hold them indefinitely but it also features a serious

Symbol	Probability	Range
a	0.2	[0, 0.2)
e	0.3	[0.2, 0.5)
i	0.2	[0.5, 0.7)
o	0.1	[0.7, 0.8)
u	0.1	[0.8, 0.9)
!	0.1	[0.9, 1.0)

drawback. The dictionary keeps growing forever without bound. There are a lot of methods to limit dictionary size. the easiest one is to stop adding entries and continue like a static dictionary coder or to throw the dictionary away and start from scratch after a certain number of entries has been reached[11]

Algorithm 2 LZ-78

```
q:NIL;
while there is input do
    j : next symbol from input
    if q_j exists in the dictionary then
        q : q_j
    else
        output(index(q), j)
        Add q_j to the dictionary
        q : NIL
    end if
end while
```

6 Arithmetic Coding

Arithmetic coding [12] is a data compression technique that encodes data (the data string) by creating a code string which represents a fractional value on the number line between 0 and 1. The coding algorithm is symbolwise recursive; i.e., it operates upon and encodes (decodes) one data symbol per iteration or recursion. On each recursion, the algorithm successively partitions an interval of the number line between 0 and 1, and retains one of the partitions as the new interval. Thus, the algorithm successively deals with smaller intervals, and the code string, viewed as a magnitude, lies in each of the nested intervals. The data string is recovered by using magnitude comparisons on the code string to recreate how the encoder must have successively partitioned and retained each nested subinterval. Arithmetic coding differs considerably from the more familiar compression coding techniques, such as prefix (Huffman) codes. Also, it should not be confused with error control coding, whose object is to detect and correct errors in computer operations.[13] [14]

Algorithm 3 Arithmetic Coding

```
count source units
interval I := new interval 0..1
divide I according to rate of units
readSymbol(X)

while X != EOF do
    new I := subinterval I matching X
    divide I acording to rate of units
    readSymbol(X)
end while
output(best number from I)
```

For example, suppose the alphabet is (a, e, i, O, u, !), and a fixed model is used with probabilities shown

We have to transmit the message 'eaii!'. Initially, both encoder and decoder know that the range is [0, 1). After seeing the first symbol, e, the encoder narrows it to [0.2, 0.5), the range the model allocates to this symbol. The second symbol, a, will narrow this new range to the first one-fifth of it, since a has been allocated [0, 0.2). This produces [O.2, 0.26), since the previous range was 0.3 units long and one-fifth of that is 0.06. The next symbol, i, is allocated [0.7, 0.8),which when applied to [0.2, 0.26) gives the smaller range [0.242, 0.248). Proceeding in this way, the en-coded message builds up as follows:

Initially [0, 1)

After seeing

e [0.2, 0.5)
a [0.2, 0.26)
i [0.242, 0.248)
i [0.2462, 0.2468)
! [0.24674, 0.2468)

Suppose all the decoder knows about the message is the final range, [0.24674, 0.2468) .It can -immediately deduce that the first character was e since the range lies entirely within the space the model of Table allocates for e. Now it can simulate the operation of the encoder:

Initially [0, 1)
After seeing e [0.2, 0.5)

This makes it clear that the second character is a,since this will produce the range

After seeing a [0.2, 0.26),which entirely encloses the given range [0.24674, 0.2468). Proceeding like this, the decoder can identify the whole message. However, the decoder will face the problem of detecting the end of the message, to determine when to stop decoding. After all, the single number 0.0 could represent any of a, aa, aaa, aaaa, ... To resolve the ambiguity, we ensure that each message ends with a special EOF symbol known to both encoder and decoder. For the alphabet of Table I, "!" will be used to terminate messages When the decoder sees this symbol,it stops decoding.

Relative to the fixed model of Table, the entropy of the five-symbol message 'eaii!' is

$$-\log 0.3 - \log 0.2 - \log 0.1 - \log 0.1 - \log 0.1 = -\log 0.00006 = 4.22$$

7 DEFLATE

In computing, Deflate may be a lossless data compression file format that uses both LZ77 and Huffman coding. It was designed by Phil Katz, for version 2 of his PKZIP archiving tool. Deflate was later specified in RFC 1951 [15] The deflate compressor is given an excellent deal of flexibility to compress the info . The programmer must design smart algorithms to form the proper choices, but the compressor does have choices about the way to compress data.

There are three modes of compression that the compressor has available:

1. Not compressed at all. This is an intelligent choice for, say, data that's already been compressed. Data stored during this mode will expand slightly, but not as much as it would if it was already compressed and any other compression methods was tried upon it.

2. Compression, first with LZ77 then with Huffman coding. The trees used to compress during this mode are defined by the Deflate specification itself and no extra space must be taken to store those trees.

3. Compression, first with LZ77 then with Huffman coding with trees that the compressor creates and stores along side the data.

The data is choppy in "blocks," and every block uses a single mode of compression. If the compressor wants to change from non-compressed storage to compression with the trees defined by the specification, or to compression with specified Huffman trees, or to compression with a different pair of Huffman trees, the present block must be ended and another one begun.

Current Research Work

The current research work in Data Compression has shifted more towards the application part, where data compression is being applied in all sorts of fields like Deep Learning, Networking, Image Compression, Video and Audio Compression etc. We have discussed few papers which represent the research done in various domains to get the most out of data compression and make computing systems more efficient.

8 Tweet Classification By Data Compression [16]

The above mentioned paper proposes a compression based method for classification of tweets. They used the DEFLATE algorithm to compress the tweet and then evaluate and classify the given tweet according to its compressibility. The proposed method achieved higher accuracy when compared to the state of the art learning methods.

8.1 Proposed Method

The main problem was to classify the tweet according to the sentiment of the tweet, i.e classifying them into "positive" and "negative" classes. The algorithm was fed query strings and then slow it learns the difference by creating two tweet models M_n and M_p. Finally the algorithm calculates a classification score $f(x)$ which is then used to finally classify using a fixed threshold value.

8.2 Classification

There are 2 steps involved in the classification procedure. The first step calculated the compressibility of the tweet and the second step calculates the classification score.

The Compressibility scores are calculated as follows [16]

$$C_p = Z(M_p.x) - Z(M_p)$$
$$C_n = Z(M_n.x) - Z(M_n)$$

Where C_p and C_n are the compressibility scores. $M_p.x$ implies that x is appended to M_p and $Z(k)$ is the compressed size of the input k

And finally the classification score is calculated as follows

$$f(x) = \frac{C_p(x) + \gamma}{C_n(x) + \gamma}$$

where γ is a smoothing parameter. Also known as Laplace Smoothing.

8.3 Related Work

Data compression has recently been used for data mining. Benedetto et al. used gzip for language recognition, authorship attribution, and language classification [17]. Cilibrasi and Vit´anyi presented a method for clustering by compression, which is based on the normalized compression distance (NCD) [18], and reported the evidence of successful application in areas as diverse as genomics, virology, music, astronomy, etc [19]. Bratko et al. investigated an approach to spam filtering based on statistical data compression models, and their empirical evaluations indicated that their approach outperforms currently established spam filters based on machine learning algorithms [20]. Marton et al. reported the effectiveness and behavior of different compression-based text classification methods on English text [21].

On the other hand, machine learning approaches have been frequently applied to Twitter mining. Irani et al. devised machine learning methods such as the decision tree for identifying the tweets in which spammers misused trending topics by using tweet text and the associated web pages with the tweet [22]. Sriram et al. devised a Naive Bayes classifier for identifying tweet types such as news, events, opinions, deals, and private messages based on the author information and tweet texts [23]. Sakaki et al. devised a support vector machine for finding tweets related to an event (e.g., earthquake) [24]. To their knowledge, there was no application of data compression to Twitter mining.

8.4 Results

According to the results mentioned in the article, CTC performed statistically better than the state of the art methods. This method also applies to multi lingual tweets which gives it an edge over some of the other machine learning approaches.

This paper was an interesting application of the DEFLATE algorithm to classify tweets.

9 Energy Aware Lossless Data Compression [25]

This paper reports a rather different viewpoint on the whole compression domain. The energy required to send a bit via wireless transmission is close to 1000 times higher than the energy required for a regular 32 bit operation on a computer [26]. The article reports the fact that compressing and decompressing information before and after transmission is more energy expensive compared to the case without any compression. The paper suggests solutions to this problem and they are able to achive an energy reduction of close to 51%.

9.1 Lossless Data Compression of low-bandwidth devices

Compression algorithms are versatile. They can be applied at a lot of points in the hardware-software spectrum. When compression is applied in hardware, the benefits and costs propagate to all aspects of the system. Compression in software may have a more dramatic effect, but for better or worse, its effects will be less global. An all-purpose header compression scheme (not confined to TCP/IP or any particular protocol) appears in [27]. TCP/IP payloads can be compressed as well with IPComp [28], but this can be wasted effort if data has already been compressed at the application layer.

LBFS or Low Bandwidth File System leverages the similiarities between the data stored on the client and the server, and only exchanges data blocks that differ [29]. Compression is applied before the data is transmitted. Rsync [30] is a protocol for efficient file transfer which preceded LBFS. A protocol-independent scheme for text compression, NCTCSys, is presented in [31]

Along with remote proxy servers which may cache or reformat data for mobile clients, splitting the proxy between client and server has been proposed to implement certain types of network traffic reduction for HTTP transactions [32] [33]. Because the delay required for manipulating data can be small in comparison with the latency of the wireless link, bandwidth can be saved with little effect on user experience. Alternatively, compression can be built into servers and clients as in the mod gzip module available for the Apache webserver and HTTP 1.1 compliant browsers [34]. Delta encoding, the transmission of only parts of documents which differ between client and server, can also be used to compress network traffic [35] [36] [37] [38]

9.2 Methodologies and Summary

The tests were done on a Compaq Personal Server codenamed "Skiff".[39]

The energy required for compression and decompression are almost directly proportional to time required for executing. The energy required is observed to be more for aggressively compressed data due to the large number of memory references. Even though the energy requirement is proportional to the execution time, using the fastest algorithm for compression and decompression doesn't reduce the energy footprint. They were able to conclude the fact that reducing energy is not as simple as picking the fastest compression algorithm. To facilitate such dynamic energy adjustment, they were working on EProf: a portable, realtime, energy profiler which plugs into the PC-Card socket of a portable device [40]

The energy requirements of the CPU and Network change drastically. They are extremely difficult to predict over a period of time. Thus software developers have to be aware of their hardware constraints and optimize accordingly to reduce energy footprint.

10 Lossless Compression of Already Compressed Textures [41]

The following cited paper investigates and proposes a lossless compression algorithm for rendering textures in graphics

Compressing textures allow the computer to render more textures due to smaller memory consumption, as the memory access. Compared to image compression methods like JPEG however, textures codecs are typically much less efficient, which is a problem when downloading the texture over a network or reading it from disk. Therefore, in this paper we

investigate lossless compression of already compressed textures. By predicting compression parameters in the image domain instead of in the parameter domain, a more efficient representation is obtained compared to using general compression such as ZIP or LZMA.

11 On the Compressive Power of Boolean Threshold Autoencoders [42]

An autoencoder is a layered neural network whose structure can be viewed as consisting of an encoder, which compresses an input vector of dimension D to a vector of low dimension d, and a decoder which transforms the low-dimensional vector back to the original input vector (or one that is very similar). In this paper they explore the compressive power of autoencoders that are Boolean threshold networks by studying the numbers of nodes and layers that are required to ensure that each vector in a given set of distinct input binary vectors is transformed back to its original[43]. It is shown that that for any set of n distinct vectors there exists a seven-layer autoencoder with the smallest possible middle layer, (i.e., its size is logarithmic in n), but that there is a set of n vectors for which there is no three-layer autoencoder with a middle layer of the same size. In addition we present a kind of trade-off: if a considerably larger middle layer is permissible then a five-layer autoencoder does exist. The results obtained suggest that it is the decoding that constitutes the bottleneck of autoencoding. For example, there always is a three-layer Boolean threshold encoder that compresses n vectors into a dimension that is reduced to twice the logarithm of n.[44] [45] [46]

12 Texture compression using low-frequency signal modulation [47]

A new lossy texture compression technique is presented in this paper that is suited to implementation on low-cost, low-bandwidth devices as well as more powerful rendering systems. It uses a representation that is based on the blending of two (or more) 'low frequency' signals using a high frequency but low precision modulation signal. Continuity of the low frequency signals helps to avoid block artifacts. Decompression costs are kept low through use of fixed-rate encoding and by eliminating indirect data access.

13 Conclusion and Future Prospects

In the presented review article we had an overview and simple examples of some of the most established methods in the domain of Lossless Data Compression. We also had a look at some of the modern research going on in the field. Almost every software system in function today has a component of compression and decompression to reduce the load and improve efficiency.

When large groups of similar files are stored, such as a group of sequential videos, they can be combined into archive files. Archive files don't have to be compressed, but they are often several gigabytes in size, so they are usually compressed to save storage space and transmission time. Most compression programs also have the ability to archive files into one large file. Most operating systems will compress their backup files, which are very large archive files, since these files just take up space until the need for them arises.

As we are moving towards a more data driven age, the amount of which users are creating data is exploding. In such scenarios, data compression stands at a very crucial juncture. Active research in this field will yield astonishing speed improvements in the current computing landscape.

References

[1] Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[2] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[3] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[4] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. 1994.

[5] PM Parekar and SS Thakare. Lossless data compression algorithm–a review. *International Journal of Computer Science & Information Technologies*, 5(1), 2014.

[6] P Yellamma and Narasimham Challa. Performance analysis of different data compression techniques on text file. *International Journal of Engineering Research & Technology (IJERT)*, 1(8):1–6, 2012.

[7] Jean Berstel and Dominique Perrin. *Theory of codes*. Academic Press, 1985.

[8] Robert M Fano. *The transmission of information*. Massachusetts Institute of Technology, Research Laboratory of Electronics . . . , 1949.

[9] Leon Gordon Kraft. *A device for quantizing, grouping, and coding amplitude-modulated pulses*. PhD thesis, Massachusetts Institute of Technology, 1949.

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

[11] Christina Zeeh. The lempel ziv algorithm. In *URL: http://w3studi. informatik. uni-stuttgart. de/˜ zeehca/Seminar/LempelZivReport. pdf [accessed November 3, 2003]*, 2003.

[12] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[13] Jorma Rissanen and Glen G Langdon. Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162, 1979.

[14] Glen G Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2):135–149, 1984.

[15] Deflate. *https://tools.ietf.org/html/rfc1951*.

[16] Kyosuke Nishida, Ryohei Banno, Ko Fujimura, and Takahide Hoshide. Tweet classification by data compression. In *Proceedings of the 2011 International Workshop on DETecting and Exploiting Cultural DiversiTy on the Social Web*, DETECT '11, page 29–34, New York, NY, USA, 2011. Association for Computing Machinery.

[17] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language trees and zipping. *Physical Review Letters*, 88(4):048702, 2002.

[18] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul MB Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004.

[19] Rudi Cilibrasi and Paul MB Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[20] Andrej Bratko, Gordon V Cormack, Bogdan Filipič, Thomas R Lynam, and Blaž Zupan. Spam filtering using statistical data compression models. *Journal of machine learning research*, 7(Dec):2673–2698, 2006.

[21] Yuval Marton, Ning Wu, and Lisa Hellerstein. On compression-based text classification. In *European Conference on Information Retrieval*, pages 300–314. Springer, 2005.

[22] Danesh Irani, Steve Webb, Calton Pu, and Kang Li. Study of trend-stuffing on twitter through text classification. In *Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2010.

[23] Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, and Murat Demirbas. Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842, 2010.

[24] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860, 2010.

[25] Kenneth C. Barr and Krste Asanović. Energy-aware lossless data compression. *ACM Trans. Comput. Syst.*, 24(3):250–291, August 2006.

[26] Application Note. Writing efficient c for arm. *ARM Ltd, Note*, 34:12–13, 1998.

[27] Jeremy Lilley, Jason Yang, Hari Balakrishnan, and Srinivasan Seshan. A unified header compression framework for low-bandwidth links. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 131–142, 2000.

[28] A Shacham, B Monsour, R Pereira, and M Thomas. Rfc3173: Ip payload compression protocol (ipcomp), 2001.

[29] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 174–187, 2001.

[30] A Tridgell. Efficient algorithms for sorting and synchronization—thesis, 2000.

[31] Nitin Motgi and Amar Mukherjee. Network conscious text compression system (nctcsys). In *Proceedings International Conference on Information Technology: Coding and Computing*, pages 440–446. IEEE, 2001.

[32] Barron C Housel and David B Lindquist. Webexpress: A system for optimizing web browsing in a wireless environment. In *Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 108–116, 1996.

[33] Ronny Krashinsky. Efficient web browsing for mobile clients using http compression. *See: http://www. cag. lcs. mit. edu/˜ ronny/classes/httpcomp. pdf*, 2003.

[34] Hyperspace communications, inc. mod gzip. *http://www.ehyperspace.com /htmlonly/products/mod gzip.html.*

[35] James J Hunt, Kiem-Phong Vo, and Walter F Tichy. An empirical study of delta algorithms. In *International Workshop on Software Configuration Management*, pages 49–66. Springer, 1996.

[36] Jeffrey Mogul. A trace-based analysis of duplicate suppression in http. 1999.

[37] Jeffrey C Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 181–194, 1997.

[38] Jonathan R Santos and David Wetherall. Increasing effective link bandwidth by supressing replicated data. In *USENIX Annual Technical Conference*, number 98, 1998.

[39] J. Hicks et al. Compaq personal server project. *http://crl.research.compaq.com/projects/personalserver/default.htm*, 1999.

[40] Kelly Koskelin, Kenneth Barr, and Krste Asanovic. Eprof: An energy profiler for the ipaq. In *2nd Annual Student Oxygen Workshop*, 2002.

[41] Jacob Strom and Per Wennersten. Lossless compression of already compressed textures. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 177–182, 2011.

[42] Avraham A Melkman, Sini Guo, Wai-Ki Ching, Pengyu Liu, and Tatsuya Akutsu. On the compressive power of boolean threshold autoencoders. *arXiv preprint arXiv:2004.09735*, 2020.

[43] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[44] James A Anderson, Edward Rosenfeld, and Andras Pellionisz. *Neurocomputing*, volume 2. MIT press, 1988.

[45] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[46] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[47] Simon Fenney. Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, page 84–91, Goslar, DEU, 2003. Eurographics Association.