Literature Review

Sasha Jenner

May 2, 2022

1 Data Compression

1.1 Introduction

Data compression or source coding is the process of encoding information using fewer bits. There are two classes of data compression; lossless and lossy. Lossless data compression does not lose information in the process of encoding. Rather, it eliminates statistical redundancy. On the other hand, lossy data compression partially discards information such that the original data cannot be fully reconstructed. For this reason, lossy compression is also known as irreversible compression.

Data compression relies on the framework of information theory established fundamentally by Claude E. Shannon in "A Mathematical Theory of Communication" [1]. Information theory studies the transmission of digital information through communication systems. The communication system Shannon theorises is composed of five parts: the information source, transmitter, channel, receiver and destination. Depending on the information transmitted through the system, it is classified into three main categories; discrete, continuous and mixed.

In the discrete case, the information source can be represented by a stochastic process

$$\{X_t : t \in \mathbb{N}^+\}$$

with a discrete state space or alphabet of symbols S. Governed by this mathematical model, the information source produces a sequence of symbols from its alphabet which is received by the transmitter. If the probability of the transmitter observing the next symbol depends only on the current symbol and is conditionally independent of previous symbols, the process satisfies the Markov property. In this case, the information source can be represented by a Markov chain with a finite state space $S_1, S_2, \ldots, S_n$ and a

set of transition probabilities $p_i(j)$ defined as the probability that the system will transition from state S_i to S_j.

The entropy rate, H, of an information source is a measure of how much information it 'produces' per symbol on average. Or rather, how uncertain the transmitter is of the next symbol on average. For a discrete random variable X with possible symbols $s_1, s_2 \ldots, s_n$; the entropy is given by

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where p_i is the probability of observing the i-th symbol [1]. The units here are in bits. For an information source represented as a stochastic process, the entropy rate is given by

$$H(X) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n)$$

when the limit exists, where $H(X_1, X_2, \ldots, X_n)$ is the joint entropy of the first n members of the process [2]. The limit is known to exist for a stationary process where the joint probability distribution is invariant to shifts in time. For example, for a stationary Markov chain the entropy rate simplifies to

$$H(X) = -\sum_{i,j} \mu_i p_i(j) \log_2 p_i(j)$$

where μ is the stationary distribution of the chain.

A binary source code, C, of a random variable X is a mapping from its possible outcomes Ω to a set of finite length strings of symbols from $\{0, 1\}$ known as codewords. A code is described as a prefix code or instantaneous code when no codeword is a prefix of another. In such a case, any encoded string can be conveniently decoded without reference to future codewords. For uniquely decodable codes, the expected length of a codeword cannot be smaller than the entropy of X. That is,

$$\sum_i p_i l_i \geq H(X)$$

where $l_i = |C(s_i)|$ is the length of the i-th symbol's codeword. With equality possible if and only if the probability distribution is dyadic. That is, when $\forall i\ \exists n \in \mathbb{N}_0$ such that $p_i = 2^{-n}$. Similarly, the entropy rate also defines a lower limit on the expected number of bits per symbol for a source encoding of an information source represented as a stochastic process.

1.2 Entropy coding

Many lossless data compression methods have been proposed since Shannon's formative paper in 1948. Entropy coding is one such scheme that does not depend on the data's characteristics. It typically encodes each fixed-width symbol with a variable-length binary prefix code as described above. Fundamentally, it attempts to achieve a bit rate as close as possible to the entropy rate of the information source.

Huffman coding is a well-known entropy coding technique which minimises the expected length of prefix codes [3]. It solves the following optimisation problem:

$$\begin{aligned} &\underset{l_i}{\text{minimise}} \sum_i p_i l_i \\ &\text{subject to} \sum_i 2^{-l_i} \leq 1 \end{aligned}$$

where $l_i \in \mathbb{N}^+$. The inequality constraint is known as the Kraft-McMillan inequality and defines a necessary condition for the existence of a uniquely decodable binary code [4]. The Huffman coding algorithm is a greedy algorithm which constructs a binary tree from the bottom-up. Recursively, it takes the two symbols with the lowest probabilities and constructs a new node with these two as its children. This new node is considered to be a new symbol with the combined probability of its children. Once the tree is constructed, the edges to the left are labelled as 0 and those to the right as 1. A hash table mapping symbols to binary codewords is then constructed by traversing the tree from the root to each leaf node and concatenating the edge labels along the path taken. The time complexity for this algorithm is $O(n \log n)$ or $O(n)$ if the probabilities are already sorted, where n is the number of symbols [5].

However, perhaps a more appropriate measure of time complexity is the number of passes over an input string of length m. In this case, Huffman coding takes $O(m)$ time as it requires one pass if the probability distribution is already known. Otherwise, the probability distribution can be determined by a preliminary pass over the input. Or instead, an adaptive Huffman coding algorithm such as the Faller-Gallager-Knuth (FGK) or Vitter algorithm can be used [6, 7]. In the latter cases, the hash table or Huffman tree must also be encoded in the output.

Among all source codes when the probability distribution of the alphabet is known beforehand, Huffman coding is indeed optimal. However, once these restrictions are relaxed, other techniques prove to be more compressible. In particular, Huffman coding tends to be inefficient on small alphabets such

as the binary numbers $\{0, 1\}$ wherein no compression is possible without an alphabet extension [8]. Similarly, if the probability distribution is not dyadic, other non–source-coding techniques are more optimal.

Arithmetic coding is one such entropy coding technique which encodes the input string as a fraction in the range $[0, 1)$ [8]. The starting interval $[0, 1)$ is divided into sub-intervals corresponding to each symbol in the alphabet with the length of each sub-interval equal to its symbol's expected probability of occurrence. Each time a symbol is read from the input string its sub-interval is chosen and divided as before. Recursively, this process is repeated until the final symbol is read, at which point any fraction in the final interval can be returned. Arithmetic coding overcomes the weaknesses of Huffman coding described above and requires the same number of passes over the input. However, in practice it is more difficult to implement and face issues of computational accuracy. Range coding is an equivalent technique with a slightly different implementation [9].

1.3 Dictionary coding

Dictionary coding is another lossless compression scheme which searches in the input data for strings stored in a dictionary and substitutes matches with a reference to their location in the dictionary. There are two main classes of dictionary coders; ones that use a static predetermined dictionary and others which dynamically update their dictionary.

Byte-pair encoding falls under the second class and recursively replaces the most frequently occurring pair of adjacent bytes with a new byte not found in the input data [10]. Despite being so simple, Gage claims it is on par with the Lempel, Ziv and Welch method [11]. It is however quite slow at compression, requiring many passes of the data, but takes only one pass for decompression.

Lempel-Ziv coding is another dictionary coding technique which is widely used in practice and has many variants. LZ77 is the first such algorithm to be introduced in the literature and replaces repeated occurrences of substrings with references to their original location in the input [12]. It maintains a fixed-width sliding window represented as a circular buffer in order to keep track of the most recent data whilst keeping the algorithm's complexity under control. Each match is encoded by the substring's length and the distance to its original occurrence. LZ78 is very similar but instead constructs an explicit dictionary of substrings and replaces repeated occurrences with references to their location in the dictionary [13]. Both variants require one pass over the input string and have been proven to be asymptotically optimal as the length of the string grows to infinity.

1.4 Other encodings

There are many other lossless data compression schemes described in the literature. Run-length encoding is one such method which encodes substrings consisting of repeated consecutive symbols, known as *runs*, with their repeated symbol and length. First employed in 1967 for transmission of analogue television signals, run-length encoding proves beneficial when there are many runs, especially of long length [14]. Unfortunately, nanopore signal data does not contain many such runs and would likely encode poorly under this method.

Burrows-Wheeler transform is a data compression preprocessing step used to rearrange data in order to increase its number of runs [15]. It is easily reversible, such that the original untransformed data can be obtained from the Burrows-Wheeler transformed data. It has been used to great effect in bioinformatics to compress to basecalled genomic data in the form of FASTQ files [16].

Stream VByte is a specialised codec for compressing 32-bit unsigned integers [17]. It stores each integer using a variable number of bytes (1 to 4) depending on its size. For an integer in the interval $[2^{n-1}, 2^{n+7})$ only n bytes are required to represent it. The number of bytes used for each integer is stored in an array of control bytes which preface the actual data. 2-bit words are used to store the number of bytes used for each integer with 00, 01, 10 and 11 corresponding to 1, 2, 3 and 4 bytes respectively. This codec is currently incorporated in the state-of-the-art approach to compressing nanopore signal data known as *VBZ* [18]. The approach consists of

1. taking the integer differences between successive nanopore data points;
2. converting the signed differences to unsigned integers;
3. applying Stream VByte with 0 to 3 bytes (rather than the canonical 1 to 4); and
4. compressing using the Zstandard format [19].

This requires at least four passes over the input data depending on how many passes Zstandard performs.

2 Nanopore Signal Data

The primary interest of bioinformatics is to understand biological data. This is typically achieved through the development of software tools. Nanopore

sequencing is one approach employed in bioinformatics to determining the primary structure of biopolymers such as DNA and RNA. It involves recording the ionic current as a biopolymer passes through a pore of nanometer size with a voltage applied to the membrane. The resulting current signal is then used to determine the structure of the biopolymer. Oxford Nanopore is the leading producer of nanopore sequencing machines. The signal recorded by such machines is digitised and given as a sequence of 16-bit signed integers. This sequence, hereafter referred to as *nanopore signal data*, is the focus of this thesis.

There have been many studies investigating genomic data compression [20]. Few, however, have successfully focussed on losslessly compressing the signal data produced directly from nanopore sequencing. One tool called Picopore does not produce any novel insights, but rather, increases the level of gzip compression to the highest possible [21]. VBZ, described in section 1.4, is the current state-of-the-art encoding introduced by Oxford Nanopore in 2019.

Rather interestingly, there has been some recent research effects to investigate lossy compression of nanopore signal data. Chandak et. al. found that lossy time-series compressors (LFZip and SZ) tend to have a very small impact on the downstream analysis of nanopore data [22, 23]. In particular, after reducing the input size by 35-50%, basecalling and consensus accuracy is reduced by less than 0.2% and 0.002% respectively. However, they seem to overlook the prospect of a more efficient lossless compression technique for nanopore signal data:

> "obtaining further improvements in lossless compression (to VBZ) is challenging due to the inherently noisy nature of the current measurements" [22].

For this reason and perhaps due to the novel nature of nanopore sequencing, little has been further attempted in the literature to losslessly compressing nanopore signal data.

Fortunately, signal data similar in form to nanopore signal data is commonly recorded and extensively researched. Some examples include electrograms of the brain (EEG) and heart (ECG); seismic and radio waves; telemetry data from astronomy; and sonar signals. MCDRC is a deep learning approach to losslessly compressing sensor signal data based on a recurrent neural network architecture known as a multi-channel recurrent unit [24]. Its results are quite promising, outperforming existing techniques such as BSC, PAQ and CMIX.

A timeline of the major events in the data compression and nanopore sequencing literature is presented in Table 1.

Table 1: A timeline of the major events in the literature.

1948	Shannon publishes "A Mathematical Theory of Communication" [1]
1952	Huffman publishes a method for finding optimal prefix codes [3]
1967	Run-length encoding first described [14]
1976	Pasco and Rissanen develop arithmetic coding independently [8]
1977	Lempel and Ziv publish LZ77 [12]
1978	Lempel and Ziv improve upon LZ77 with LZ78 [13]
1979	Martin proposes range coding [9]
1984	Welch publishes LZW based on LZ78 [11]
1987	Vitter improves upon the adaptive Huffman coding algorithm FGK [7]
1994	Burrows and Wheeler publish the Burrows-Wheeler transform [15]; and Gage submits byte-pair encoding as a C article [10]
2014	Oxford Nanopore release the first portable nanopore sequencing device
2017	A tool for reducing nanopore data, Picopore, is released [21]
2018	Lemire publishes Stream VByte [17]
2019	VBZ is first released
2020	Lossy compression of nanopore signal data is investigated [22]
2021	MCDRC is published [24]

3 Outline of Research

3.1 Research Problem

The primary research problem is to develop an improved lossless encoding strategy for nanopore signal data. This requires some unpacking and can be sub-divided into the following smaller research problems.

3.1.1 Determine the salient features of nanopore signal data

The first problem is to determine the features and repeated patterns of nanopore signal data. Understanding the data is the most crucial step in developing a data-specific lossless encoder. Once the features have been recognised, they can be directly exploited by an encoding strategy in order to reduce the data's redundancy and bit rate. The intention is to encode each nanopore *read* separately in order to maintain random parallel access. With this in mind, there are potentially patterns between independent reads which should first be determined.

3.1.2 Apply and evaluate appropriate existing encoding strategies

The next problem involves experimenting with and evaluating existing lossless encoding strategies which are suitable for nanopore signal data. Most likely, there already exists a strategy in the literature which when applied to nanopore data would prove better than the state-of-the-art. This problem involves further investigation into the literature of integer and signal data compression. It also requires developing a iterative evaluation framework which can quickly determine whether an existing strategy implementation is worth further investigation. Furthermore, it may prove that an existing strategy forms the basis for an improved encoder when modified specifically with the salient features of nanopore signal data in mind.

3.1.3 Develop a new lossless algorithm (or modify an existing one) which is better than VBZ

This is the final problem and main contribution I intend to make. It is clear from the literature review that few lossless encoders specific to nanopore signal data have been considered. A new lossless algorithm which is better than the state-of-the-art known as VBZ (`https://github.com/nanoporetech/vbz_compression/`) would significantly alleviate the storage issues and/or long analysis times faced by the nanopore sequencing community.

3.2 Evaluation Process

A rigorous evaluation process which determines what constitutes a 'better' lossless encoding strategy must be outlined. There are many ways of evaluating an encoder. Fundamentally, these include the algorithm's

- data compression ratio;
- space and
- time complexity; and
- passes over the data.

The data compression ratio is a measure of how well the algorithm compressing the input data. It is simply defined as

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}.$$

The space complexity of the algorithm measures how much memory it requires. Traditionally, this is expressed using big O notation but a more

practical hard limit should be imposed. Considering the size of a nanopore read is usually around 256KB when stored in uncompressed binary, space is not usually a problem. In this case, streaming should not be a requirement on the algorithm and $O(n)$ is fine.

Similarly, the time complexity of an algorithm measures how many computational operations it performs. This is also often described in big O notation as a function of the size of the input data. A similar more practical measure is the number of passes over the data or simply the time taken since most encoders are $O(n)$ anyway. One pass is optimal for nanopore signal data since it contains some degree of randomness. However, several passes may not be much slower and needs to be properly investigated.

In addition, since the compression ratio and time taken is a trade-off for typical data compression techniques. A composite measure *compression-time ratio* is introduced as

$$\text{Compression-Time Ratio} = \frac{\text{Compression Ratio}}{\text{Time Taken}}.$$

In this case, if the compression ratio and time taken are multiplied by a constant factor, the compression-time ratio remains constant. For example, an improvement in the compression ratio by a factor of two can be compensated by up to a doubling of the time taken before the compression-time ratio decreases. This is not an ideal measure but is useful for the comparison of techniques especially as a function of some tunable parameter such as the compression level.

All of these metrics apart from the compression ratio may be different for compression and decompression and should be evaluated separately. In particular the following criterion is used to determine a 'better' encoding strategy.

1. The strategy must be lossless.
2. The best compression ratio is higher than that of VBZ on average across datasets.
3. The compression-time ratio is higher than that of VBZ on average across both datasets and compression levels for both compression and decompression.
4. $O(n)$ space complexity.

The datasets used should represent typical nanopore signal data from a range of Oxford Nanopore machines; biopolymers such as DNA and RNA;

and biological species. It should be large enough, at least several datasets, to conclude confidently the performance of the encoding strategies being tested.

References

[1] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[2] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley series in telecommunications. John Wiley & Sons, 1991.

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

[4] B. McMillan. Two inequalities implied by unique decipherability. *IRE Transactions on Information Theory*, 2(4):115–116, 1956.

[5] Jan Van Leeuwen. On the construction of huffman trees. January 1976.

[6] Donald E Knuth. Dynamic huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.

[7] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *Journal of the ACM*, October 1987.

[8] J. J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3):198–203, 1976.

[9] G. Nigel N. Martin. Range encoding: An algorithm for removing redundancy from a digitized message. In *Video & Data Recording Conference*, Southampton, UK, July 1979.

[10] Phillip Gage. A new algorithm for data compression. *The C User Journal*, 1994.

[11] Terry Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

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

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

[14] A.H. Robinson and C. Cherry. Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE*, 55(3):356–364, 1967.

[15] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Digital Equipment Corporation, 1994.

[16] A. J. Cox, M. J. Bauer, T. Jakobi, and G. Rosone. Large-scale compression of genomic sequence databases with the burrows-wheeler transform. *Bioinformatics*, 28(11):1415–1419, May 2012.

[17] Daniel Lemire, Nathan Kurz, and Christoph Rupp. Stream VByte : Faster byte-oriented integer compression. *Information Processing Letters*, 130:1–6, February 2018.

[18] Hasindu Gamaarachchi, Hiruna Samarakoon, Sasha P. Jenner, James M. Ferguson, Timothy G. Amos, Jillian M. Hammond, Hassaan Saadat, Martin A. Smith, Sri Parameswaran, and Ira W. Deveson. Fast nanopore sequencing data analysis with slow5. *Nature Biotechnology*, January 2022.

[19] Yann Collet and Murray Kucherawy. Zstandard Compression and the application/zstd Media Type. RFC 8478, October 2018.

[20] Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. Genomic data compression. *Annual Review of Biomedical Data Science*, 2(1):19–37, 2019.

[21] Scott Gigante. Picopore: A tool for reducing the storage size of oxford nanopore technologies datasets without loss of functionality. *F1000Research*, 6, 2017.

[22] Shubham Chandak, Kedar Tatwawadi, Srivatsan Sridhar, and Tsachy Weissman. Impact of lossy compression of nanopore raw signal data on basecalling and consensus accuracy. *Bioinformatics*, 36(22-23):5313–5321, 2020.

[23] Shubham Chandak, Kedar Tatwawadi, Chengtao Wen, Lingyun Wang, Juan Aparicio Ojea, and Tsachy Weissman. Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction. In

2020 Data Compression Conference Proceedings (DCC), volume 2020-March, pages 342–351, 2020.

[24] Qianhao Chen, Wenqi Wu, and Wei Luo. Lossless compression of sensor signals using an untrained multi-channel recurrent neural predictor. *Applied Sciences*, 11(21), 2021.