Lossless compression of nanopore sequencing raw signals ⋆

Rafael Castelli, Tomás González, Rodrigo Torrado, Álvaro Martín[0000−0001−8601−1242], and Guillermo Dufort y Álvarez*[0000−0001−6125−5603]

Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, 11300, Montevideo, Uruguay
* To whom correspondence should be addressed: gdufort@fing.edu.uy

Abstract. Nanopore sequencing has emerged as a crucial component in the arsenal of genomic technologies, with advances from Oxford Nanopore Technologies (ONT) progressively reducing the costs of DNA sequencing. An ONT nanopore sequencer operates by guiding DNA fragments through a nanopore, partially blocking a flow of electrical current, which is sampled over time. This variation in current is registered as a *raw signal*, and it allows for the translation of electrical signals into a DNA sequence, a process known as *basecalling*. As the available algorithms for basecalling continually evolve, it is preferable to retain the raw signal data for future re-analysis.
However, the volumes of raw data are massive, being nearly ten times larger than the size of data after basecalling in FASTQ format. Therefore, efficient lossless compression algorithms for raw signals are needed to reduce storage and transmission costs.
While recent research has focused on studying nanopore FASTQ data, a thorough study of the methods used in practice for the compression of raw data, such as the state-of-the-art compression algorithm VBZ, is still missing in the scientific literature.
In this sense, in this work, we aim to elucidate the mechanisms behind the efficiency of VBZ and introduce a set of variations that further improve its compression performance. Our findings indicate that we can enhance the performance of VBZ by an average of 2.42%, with gains increasing to 3.02% for the latest nanopore flowcells (10.x), using comparable computational resources.

Keywords: Nanopore sequencing · Data compression · Nanopore raw signals · DNA sequencing.

1 Introduction

Nanopore sequencing has emerged as a crucial component in the arsenal of genomic technologies, as advances from Oxford Nanopore Technologies (ONT)

⋆ Supported by Agencia Nacional de Investigación e Innovación (FMV_3_2022_1_172797).

have progressively reduced the costs of sequencing extensive eukaryotic genomes and transcriptomes [13]. Despite these advancements, the community utilizing nanopore sequencing still faces significant challenges due to the sheer volume of data and the computational hurdles involved.

An ONT nanopore sequencer operates by guiding DNA fragments through a *nanopore*, i.e, a protein embedded in a membrane via a nanometre-sized channel. As a DNA strand traverses the *capture region* of the nanopore, it partially blocks a flow of electrical current running through the pore, which is sampled over time. Under the assumption that each different combination of DNA nucleotides present in the capture region produces a uniquely identifiable perturbation of the electrical current, this variation in current allows for the translation of electrical signals into a DNA sequence, a process known as *basecalling*.

As the available algorithms for basecalling, identification of DNA/RNA modifications, and other signal-level analyses, are continually evolving [19,11,20,6], it is preferable to retain ONT raw signal data for future re-analysis, as releases of new basecallers can enable improvements in basecalling accuracy, modified basescalling, reference-guided SNP calling, or polishing of data.[1] However, the volume of raw signal data produced by nanopore sequencing is typically very large; for instance, a human genome at 30× coverage results in about 1 TB of raw data with current compressed formats [8]. This is nearly ten times larger than the size of data after basecalling in FASTQ format. Managing such large datasets incurs high storage and transmission costs, which calls for efficient lossless compression algorithms for raw signals. However, although lossless and lossy compression of FASTQ nanopore sequencing data has been investigated recently [3,4,5,18,14,12], raw data compression has received less attention. In [1], the authors examine the effects of lossy compression on raw nanopore sequencing data, particularly its impact on basecalling. They demonstrate that, within the scope of their study, the selected tools maintain basecalling accuracy despite the loss of information. However, as we previously noted, the creation of novel basecalling algorithms might lead to significantly different outcomes, and thus lossy compression has not been adopted as a standard practice. In terms of lossless compression, various techniques are currently used in practice, but a thorough and systematic study of these methods is still missing in the scientific literature.

The first specific format developed for storing raw nanopore sequencing data was FAST5 [8], a derivative of the universal *HDF5* format.[2] Compression of FAST5 files typically involved standard tools like *gzip*.[3] Innovations such as Picopore [9] enhanced this approach by applying other ad-hoc techniques to suppress redundant data from the FAST5 format, in addition to compressing the raw signals with gzip at its highest compression level mode (at expense of speed). Nevertheless, this method continued to employ a generic compression algorithm that is not specifically designed for nanopore raw data. To improve raw signal compression, ONT introduced VBZ, a compression algorithm that

[1] https://help.nanoporetech.com/en/articles/8676645-should-i-store-pod5-files
[2] https://www.hdfgroup.org/solutions/hdf5
[3] https://www.gnu.org/software/gzip

combines the *StreamVByte* integer encoding[4] with the LZ77-based [21] ZSTD compressor.[5] VBZ was also initially developed as an HDF5 plugin. More recently, two novel file formats for raw nanopore data emerged: SLOW5 [8], engineered for efficient parallelization and acceleration of nanopore data analysis; and POD5 from ONT,[6] now the default format for raw data storage. Both formats make use of an updated version of the VBZ algorithm; we refer to the original and updated versions of VBZ as VBZ_0 and VBZ_1, respectively, and we describe them in detail in Section 2.

Although the VBZ compression algorithm is being adopted for raw nanopore signal compression in practical applications, the specifics of its operation and the statistical model assumptions have not been detailed in public research. One of the contributions of this paper is shining light on the design principles of VBZ and its underlying statistical model. Moreover, in Section 2 we propose a collection of new lossless compression algorithms for nanopore raw signals, which evolve from VBZ and further improve its compression performance. We assess the performance of these compressors in comparison to each other and against VBZ_0 and VBZ_1, utilizing publicly accessible raw nanopore sequencing data from various organisms and nanopore models (flowcell types). Our evaluation, which we report in Section 3, covers both compression and computational efficiency. The findings indicate that our methods can enhance the performance of VBZ by an average of 2.42%, with gains increasing to 3.02% for the most recent nanopore flowcells (10.x). Moreover, these improvements are attainable using computational resources comparable to VBZ in speed and memory requirements.

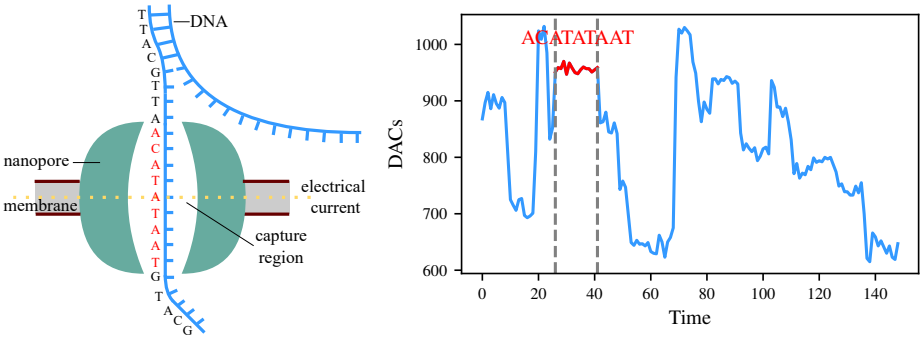
We summarize this paper in Section 4, where we present some concluding remarks and we outline directions for future work.

2 Methods

A nanopore sequencing raw signal is comprised of a sequence of so-called *data acquisition values* (DACs), $x_1, x_2, \ldots, x_n$. As a DNA strand traverses the capture region of a pore, it perturbs an electrical current, which is sampled over time (see Figure 1(a)). The sequencing device quantizes these measurements into the digital values x_i, which are represented as 16-bit signed integers. Figure 1(b) illustrates such a signal. During the time that a specific *kmer* (a sequence of k consecutive bases) occupies the capture region, the sampled DACs typically lie close around a kmer-specific mean value. In our example, the kmer ACATATAAT that is in the capture region in Figure 1(a) gives rise to the portion of sequence that is colored red in Figure 1(b). As the DNA strand moves on through the pore, different kmers successively occupy the pore, each producing a sequence of DACs with a different characteristic mean value. Generally, this results in a stepped-shape signal, as in Figure 1(b), where each roughly flat region corresponds to a specific kmer interposed in the capture region.

[4] https://github.com/lemire/streamvbyte
[5] https://github.com/facebook/zstd
[6] https://github.com/nanoporetech/pod5-file-format

Fig. 1: (a) A representation of a DNA molecule passing through a nanopore and perturbing the electrical current. (b) An example of a fragment of a nanopore raw signal. The part of the signal corresponding to the perturbation produced by the kmer ACATATAAT is colored in red.



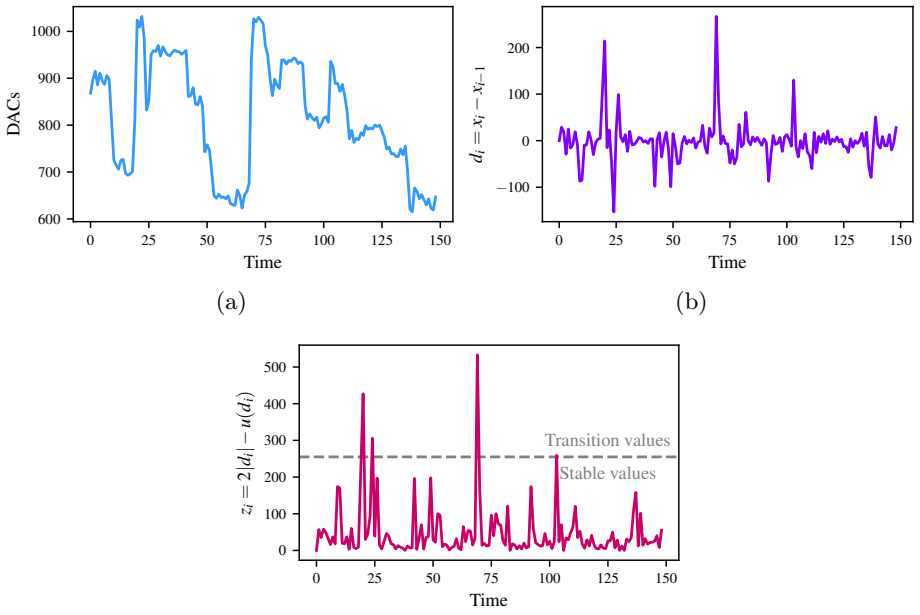
(a) (b)

We refer to a subsequence $x_i, x_{i+1}, ..., x_{i+m}$ where all samples originate from the same kmer as a *stable region*; in our example, the stable region generated by kmer ACATATAAT is colored red in Figure 1(b). A usually adopted statistical model for nanopore raw signals posits that all values x_i in a stable region are sampled independently with the same probability distribution, which, according to ONT, follows a normal discrete distribution [2], whose mean depends on the kmer occupying the pore.

Given that each stable region has a characteristic mean, the differences between consecutive samples tend to be centered around zero, regardless of the specific kmer present in the capture region. Let the *difference sequence*, $d_1, d_2, \ldots, d_n$, represent the differences between consecutive DACs, i.e., $d_i = x_i - x_{i-1}$ for $i > 1$, and $d_1 = x_1$. Notice that the sequence $d_1, d_2, \ldots, d_n$ univocally determines $x_1, x_2, \ldots, x_n$; VBZ and all our proposed algorithms encode a representation of the difference sequence, from which the decoder recovers the original DAC sequence.

Figures 2(a) and 2(b) provide an illustrative example of a DAC sequence, and the corresponding difference sequence, respectively. The difference sequence exhibits areas of consistently low absolute values, indicative of stable regions, interspersed with abrupt spikes generated by transitions from a stable region to the next. In general, as suggested in the figure, low absolute values (referred to as *stable*) occur more frequently than large absolute values (referred to as *transitional*), which is exploited in VBZ by using shorter codes (i.e., using fewer bits) for the former than for the latter. To this end, each difference value d_i is mapped to a non-negative integer $z_i = 2|d_i| - u(d_i)$, where $u(d_i) = 1$ for negative d_i $u(d_i) = 0$ otherwise. This mapping enumerates non-negative integers in increasing order of absolute value, translating $0, -1, 1, -2, 2, \ldots$ into

Fig. 2: (a) A section of a raw nanopore sequencing signal. (b) Differential encoding of the raw signal. (c) Rice mapping of the differential sequence.



$0, 1, 2, 3, 4, \ldots$. This enumeration is known as Rice mapping [16] in the context of Golomb coding [10] for data compression and it is also popularly called *zig-zag* encoding. Since the mapping is clearly reversible, from each z_i a decoder can recover d_i and, as explained before, from $d_1, \ldots, d_n$ it can reconstruct the original DAC sequence. Figure 2(c) shows the Rice mapping applied to the difference sequence of Figure 2(b).

The central aspects of the VBZ compression algorithm are listed below; it is essentially comprised of a *StreamVByte*-like algorithm, which applies different encodings to stable and transitional values, combined with the ZSTD algorithm.

1. Each value z_i is coded as a pair (k_i, q_i), where k_i is a *key* that determines the number ℓ_i of bits in a fixed length code for z_i, and q_i is a string of ℓ_i bits (a code word) that encodes z_i with such fixed length code. In VBZ, q_i is simply the representation of z_i as an unsigned integer of ℓ_i bits, but we will use other codes in some of our proposed algorithms.
2. All keys k_i are coded with a fixed length code and stored in a specific buffer, called *key buffer* and denoted B_k, in order of occurrence.
3. The code words q_i are stored in a specific *data buffer*, denoted B_d, separate from the key buffer, in order of occurrence.
4. The concatenation of the buffers B_k and B_d is compressed with the ZSTD algorithm.

Given a VBZ compressed bit stream, a decoder recovers the DAC sequence going through the following steps:

1. Apply a ZSTD decoder to recover the buffers B_k and B_d.
2. Iterate through the keys in B_k and for each k_i determine the number ℓ_i of bits to be read from the data buffer to obtain q_i, which represents z_i.
3. From z_i obtain d_i by inverting the Rice mapping.
4. From the difference sequence $d_1, \ldots, d_n$ reconstruct the original DAC sequence $x_1, \ldots, x_n$.

VBZ_0 and VBZ_1 differ in the definition of the keys, and the definition of the code for z_i corresponding to each possible key value. In VBZ_0, these definitions are as follows.

- for $z_i = 0$, define $k_i = 0$ and $\ell_i = 0$ (q_i is empty in this case).
- for $0 < z_i \leq 15$, define $k_i = 1$ and $\ell_i = 4$.
- for $15 < z_i \leq 255$, define $k_i = 2$ and $\ell_i = 8$.
- for $255 < z_i$, define $k_i = 3$ and $\ell_i = 16$.

Thus, VBZ_0 defines 4 different key values, which are coded using 2 bits per key in the buffer B_k. In contrast, VBZ_1 utilizes only two key values, each coded with a single bit in B_k. The definitions in this case are as follows.

- for $z_i \leq 255$, define $k_i = 0$ and $\ell_i = 8$.
- for $255 < z_i$, define $k_i = 1$ and $\ell_i = 16$.

In Figure 2(c), values z_i beneath the dotted line are assigned a key $k_i = 0$, and values above the dotted line are assigned a key $k_i = 1$ in algorithm VBZ_1.

Recall that, in general, we expect low values of z_i to occur more frequently than large values. The way in which VBZ exploits this is two fold: on one hand, small values of z_i correspond to small values of ℓ_i, so that fewer bits are written to B_d; on the other hand, a high frequency of small values of z_i yields a highly repetitive content of the key buffer, which makes it highly compressible with ZSTD. Consider VBZ_1 for example, and suppose that 90% of the values z_i are smaller than 256. Then all these values are encoded in B_d using 8 bits (rather than 16 as in the original representation of DACs) and 90% of the bits in B_k are 0.

Notwithstanding the foregoing strengths, we also notice some improvement opportunities:

1. **Mixed content in the input to ZSTD**.
   The buffers B_k and B_d contain data of different nature, presumably with completely different statistics. Since ZSTD compresses by learning statistical regularities in the data, compressing B_k together with B_d may harm compression performance.
2. **Mixed content in the data buffer**.
   Similarly, the buffer B_d contains interspersing values from both stable and transitional regions, which again respond to different statistical models.

3. **Byte alignment break**.
   If some of the code word length ℓ_i are not multiple of 8, as in VBZ_0, then the values stored in B_d may lose byte alignment, potentially causing some values q_i to split across two consecutive bytes. This fragmentation can significantly harm the performance of the ZSTD compressor, which performs pattern matching on the data on a 1-byte words basis.
4. **Redundancy in the coding space for z_i given k_i**.
   We notice that the coding space assigned to q_i is inherently redundant for some values of k_i, in the sense that some code words are associated to values of z_i that would receive a different key (and thus are never actually used). For example, in VBZ_0, for $k_i = 2$ we use a fixed length code for q_i of length $\ell_i = 8$, which is able to encode all values of z_i in the range $0 \leq z_i \leq 255$. However, the values in the range $0 \leq z_i \leq 15$ receive a different key value and, thus, these 16 code words are reserved but never used for $k_i = 3$.
5. **Substitution of ZSTD**.
   ZSTD is an effective and computationally efficient general purpose data compressor. In general, however, it is possible to improve the compression performance designing a specific purpose data compressor (possibly sacrifying computational efficiency).

Considering these improvement opportunities, we propose a series of new compression algorithms, which are adapted from VBZ.

1. **Compressor C1**:
   Compressor C1 is defined as VBZ_1 except that ZSTD is run separately for the key and data buffers. Thus, instead of compressing the concatenation of B_k and B_d, we concatenate the result of compressing B_k with the result of compressing B_d.
   This proposal stems from item 1 in the preceding list of improvement opportunities.
2. **Compressor C2**:
   Compressor C2 is defined as VBZ_1 but the data buffer B_d is replaced by two separate buffers, B_d^H and B_d^L. B_d^H stores the 8 most significant bits of all code words q_i that are 16 bits long, i.e., those with $k_i = 1$. B_d^L stores the 8 less significant bits of all code words (for both $k_i = 0$ and $k_i = 1$).
   Each of the three buffers, B_k, B_d^H, and B_d^L, is encoded separately with ZSTD.
   This proposal stems from items 1 and 2 in the preceding list of improvement opportunities, applied to VBZ_1.
3. **Compressor C3**:
   Compressor C3 is a refinement of C2. It is defined as VBZ_1 but the data buffer B_d is replaced by three separate buffers, B_d^H, B_d^{L0}, and B_d^{L1}. As in C2, B_d^H stores the 8 most significant bits of all 16-bits-long code words q_i (those with $k_i = 1$). However, since stable and transition values obey to different statistical models, in C3 we do not mix in the same buffer data values that were assigned different keys. Specifically, B_d^{L0} and B_d^{L1} contain the 8 less significant bits of all code words q_i with $k_i = 0$ and $k_i = 1$, respectively.
   Each of the four buffers, B_k, B_d^H, B_d^{L0}, and B_d^{L1}, is encoded separately with ZSTD.

4. **Compressor C4**:
   Compressor C4 is defined as VBZ_0 but the data buffer B_d is replaced by four separate buffers, B_d^1, B_d^2, B_d^{3H}, and B_d^{3L}. B_d^1 and B_d^2 contain all code words q_i for $k_i = 1$ and $k_i = 2$, respectively.
   B_d^{3H} and B_d^{3L} contain the 8 most significant bits and 8 less significant bits, respectively, of all code words q_i with $k_i = 3$.
   Each of the five buffers, B_k, B_d^1, B_d^2, B_d^{3H}, and B_d^{3L}, is encoded separately with ZSTD.
   This proposal stems from items 1 and 2 in the preceding list of improvement opportunities, applied to VBZ_0. In addition, since each buffer contains code words that are all the same length (4 bits in the case of B_d^1 and 8 bits for the others), item 3 is also addressed by this compressor.
5. **Compressor C5**:
   Compressor C5 is defined as C4 but, in addition, C5 redefines the coding space assigned to q_i for each possible values of k_i. This redefinition addresses item 4 in our list of improvement opportunities.
   Specifically, the definition of the keys and the code for z_i corresponding to each possible key value is as follows.
   - for $z_i = 0$, define $k_i = 0$ and $\ell_i = 0$ (q_i is empty in this case).
   - for $0 < z_i \leq 16$, define $k_i = 1$ and $\ell_i = 4$. We define q_i as the representation of $z_i - 1$ as an unsigned integer of 4 bits.
   - for $16 < z_i \leq 272$, define $k_i = 2$ and $\ell_i = 8$. We define q_i as the representation of $z_i - 17$ as an unsigned integer of 8 bits.
   - for $272 < z_i$, define $k_i = 3$ and $\ell_i = 16$. We define q_i as the representation of $z_i - 273$ as an unsigned integer of 16 bits.
6. **Compressor C6**:
   Compressor C6 is defined C3, but employs adaptive arithmetic coding [17] to compress each data buffer. This serves as a benchmark to measure the compression loss when using a fast LZ77 compression scheme compared to a slower, but more effective, entropy coding scheme (item 5 in our list of improvement opportunities).

All the proposed compression algorithms were implemented and integrated to a fork of the POD5 library, and are available online.[7]

3 Experimental Results

To assess the performance of the different compression methods, we evaluate the tools on several publicly available datasets, described in Table 1.

The selected datasets cover two types of organisms, human and fly, and different flowcell types, including pore versions 9.4.1, 10.3.1, and 10.4.1. Further instructions on how to download the datasets are available online.[8]

[7] https://github.com/tomas-gr/pod5_nanoraw_comp
[8] https://github.com/tomas-gr/pod5_nanoraw_comp

Table 1: Raw nanopore sequencing datasets used for evaluation. All the selected datasets are a part of the *Oxford Nanopore Open Data project* [15]. The datasets can be accessed using the AWS dataset names in the rightmost column.

Name	Organism	Author	# Files	Flowcell type	Size (GB)	AWS dataset name
DS 1	Fly	Stanford University	14	R10.4.1	20.1	melanogaster_bkim_2023.01
DS 2	Human	ONT	14	R10.3.1	27.2	gm24385_2020.09
DS 3	Human	ONT	15	R9.4.1	36.8	gm24385_2020.09
DS 4	Human	ONT	15	R10.4.1	27.4	giab_lsk114_2022.12

To measure the performance of a compressor on a dataset, we compress each file of the dataset separately and calculate the *compression ratio* in bits per sample (bps), CR, defined as $CR = C/S$, where S is the total number of samples of all the files in the dataset, and C is the total size in bytes of the compressed files. Notice that smaller values of CR correspond to better compression performance. To compare compression ratios, we define the *percentage relative difference (PRD)*, $\frac{CR_2 - CR_1}{CR_1} \times 100$, between the compression ratios CR_1 and CR_2, with respect to CR_1. All experiments were conducted in a Rocky Linux v8.8 server with an Intel(R) Core(TM) i9-10940X CPU, 3.30GHz (1.20GHz-4.80GHz), 256GB of RAM, and 8TB of NVMe storage.

The compression ratios obtained by VBZ_1, VBZ_0, and the proposed compressors, on each dataset, are shown in Table 2. The table also shows the percentage relative difference between the compression ratios obtained by VBZ_1 and the proposed methods, using VBZ_1 as the reference (thus, negative values indicate an advantage for the proposed methods). Finally, the last two columns of the table show the simple averages of the results.

Table 2: Comparison of compression ratios obtained on all the datasets for all compression algorithms, and percentage relative difference (PRD) with respect to VBZ_1.

Compressor	DS 1		DS 2		DS 3		DS 4		Averages	
	CR	PRD	CR	PRD	CR	PRD	CR	PRD	CR	PRD
VBZ_1	7.016		7.107		5.267		7.235		6.656	
VBZ_0	7.899	12.59	7.875	10.81	5.962	13.21	8.064	11.46	7.450	12.02
C1	6.991	-0.35	7.077	-0.42	5.269	0.04	7.197	-0.53	6.633	-0.31
C2	6.924	-1.30	6.990	-1.65	5.271	0.07	7.103	-1.82	6.572	-1.17
C3	6.919	-1.37	6.979	-1.80	5.272	0.10	7.094	-1.95	6.566	-1.25
C4	6.864	-2.16	6.909	-2.79	**5.221**	**-0.87**	7.039	-2.70	6.508	-2.13
C5	**6.831**	**-2.63**	**6.872**	**-3.30**	5.234	-0.61	**7.008**	**-3.13**	**6.487**	**-2.42**
C6	6.796	-3.13	6.847	-3.66	5.218	-0.93	6.968	-3.69	6.457	-2.85

The results show that all proposed compressors consistently outperform both VBZ algorithms in terms of compression ratio (CR), on average. Specifically, the best results on each dataset are achieved by C6. However, C6 uses adaptive arithmetic coding for the compression of the data buffers, which ultimately translates into a significantly reduced computational efficiency compared to the others, as shown in Table 3.

Between the methods that use fast ZSTD compression, the best performance is achieved by C5, which exhibits an average PRD of -2.42% compared to VBZ_1. It is worth noting that VBZ_1 improves VBZ_0 in every tested dataset.

In dataset DS 3 (using flowcell type 9.4.1), the compression ratios of the evaluated algorithms, and the advantage of the proposed methods with respect to VBZ_1, are notably smaller than in the more recent 10.x datasets. However, given the natural progression toward newer technologies, these older nanopore types will likely see reduced utilization in the future. In this sense, it is important to note that for the newer flowcell types (10.x), C5 achieves even better CR results when compared to VBZ_1, with an average PRD of 3.02%.

In terms of computational efficiency, Table 3 shows the average encoding and decoding speeds, and the maximum memory usage registered during both processes, for each compressor on each dataset. The table also shows the results of running an optimized version of VBZ_1 that uses fast SIMD instructions [7], which we report as VBZ_{1SSE} (this is the default version of VBZ in the POD5 library). Note that this version of VBZ_1 can only be run on SIMD capable x86 CPUs.

Table 3: Encoding and decoding speeds in MB/s for all the compressors on all the datasets. Best results (without considering VBZ_{1SSE}), for each dataset, both for encoding and decoding, are bold-faced. Additionally, column *mem* shows the maximum memory usage (in GB) registered during the encoding and decoding processes, for all the compressors, on all the files of each dataset.

Compressor	DS 1			DS 2			DS 3			DS 4			Averages	
	enc	dec	mem	enc	dec	mem	enc	dec	mem	enc	dec	mem	enc	dec
VBZ_{1SSE}	1006	549	2.1	995	550	2.2	1305	615	2.2	985	568	2.5	1073	570
VBZ_1	530	**368**	2.0	529	**364**	2.3	659	**346**	2.2	517	**371**	2.5	559	**362**
VBZ_0	275	133	2.2	270	127	2.4	311	115	2.3	278	138	2.7	284	128
C1	539	281	2.0	533	281	2.2	670	256	2.2	523	286	2.5	566	276
C2	**678**	339	2.0	**670**	338	2.2	909	335	2.1	**653**	342	2.5	**728**	338
C3	658	275	2.0	654	273	2.2	**943**	260	2.1	633	279	2.5	722	272
C4	274	151	2.0	272	146	2.2	309	126	2.1	276	158	2.5	283	145
C5	256	142	2.0	254	137	2.2	292	125	2.1	259	148	2.5	265	138
C6	21	8	2.0	21	8	2.2	31	9	2.2	21	8	2.5	24	8

The results show that VBZ_{1SSE} is the fastest tool for both encoding and decoding across all datasets. However, among compressors that do not utilize SIMD instructions, C2 stands out as the fastest encoder, averaging a speed of

728 MB/s; only 0.68x slower than VBZ_{1SSE}. In terms of decoding speed, VBZ_1 leads with 362 MB/s, closely followed by C2 at 338 MB/s.

Additionally, we observe that C5, which is the algorithm that achieves the best CR performance between algorithms using fast ZSTD compression, is slower than VBZ_1 in both encoding and decoding speeds, with factors of 0.47x and 0.38x, respectively. In future work this could be addressed by focusing on optimizing the algorithms through the integration of SIMD instructions, to narrow the gap in compression and decompression speed.

Regarding memory usage, the compressors demonstrate comparable performance. Across all datasets, the greatest observed difference in memory usage between any two compressors is a marginal 0.2 GB.

4 Conclusions and future work

The VBZ algorithm is widely accepted in the bioinformatics community as the state-of-the-art tool for lossless compression of nanopore sequencing raw signals, and it is the default method used by the standard raw data storage format POD5. However, the algorithm has not been previously studied in the literature.

In this work, we presented a detailed analysis of the VBZ compression algorithm. VBZ employs differential encoding alongside StreamVByte to create an encoded signal, which can be then efficiently compressed using the ZSTD algorithm. The design of this encoding takes advantage of the sequential progression of the raw signal through stable regions that are characterized by similar statistical behaviour.

More importantly, we have detected potential improvements to the VBZ algorithm, and proposed a series modifications that enhance its compression capabilities without significantly impacting computational efficiency.

Future research directions include refining the performance of these modifications through the application of SIMD instructions within the algorithms, thereby narrowing the performance disparity between the proposed modifications and the current implementation of VBZ. Additionally, we consider improving compression performance by integrating a predictor for transitions between stable regions of the signal into the compression process. Despite the inherent randomness of the transitions between stable regions, we hypothesize that the duration of these regions exhibits sufficient consistency to predict transitions, and that this information could potentially be used to improve compression performance.

References

1. Chandak, S., Tatwawadi, K., Sridhar, S., Weissman, T.: Impact of lossy compression of nanopore raw signal data on basecalling and consensus accuracy. Bioinformatics **36**(22-23), 5313–5321 (Apr 2021). https://doi.org/10.1093/bioinformatics/btaa1017

2. David, M., Dursi, L.J., Yao, D., Boutros, P.C., Simpson, J.T.: Nanocall: an open source basecaller for Oxford Nanopore sequencing data. Bioinformatics **33**(1), 49–55 (Jan 2017). https://doi.org/10.1093/bioinformatics/btw569
3. Dufort Y Álvarez, G., Seroussi, G., Smircich, P., Sotelo, J., Ochoa, I., Martín, Á.: Compression of Nanopore FASTQ Files. In: Rojas, I., Valenzuela, O., Rojas, F., Ortuño, F. (eds.) Bioinformatics and Biomedical Engineering, vol. 11465, pp. 36–47. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-17938-0_4, series Title: Lecture Notes in Computer Science
4. Dufort Y Álvarez, G., Seroussi, G., Smircich, P., Sotelo, J., Ochoa, I., Martín, Á.: ENANO: Encoder for NANOpore FASTQ files. Bioinformatics **36**(16), 4506–4507 (Aug 2020). https://doi.org/10.1093/bioinformatics/btaa551
5. Dufort Y Álvarez, G., Seroussi, G., Smircich, P., Sotelo-Silveira, J., Ochoa, I., Martín, Á.: RENANO: a REference-based compressor for NANOpore FASTQ files. Bioinformatics **37**(24), 4862–4864 (Dec 2021). https://doi.org/10.1093/bioinformatics/btab437
6. Ferguson, S., McLay, T., Andrew, R.L., Bruhl, J.J., Schwessinger, B., Borevitz, J., Jones, A.: Species-specific basecallers improve actual accuracy of nanopore sequencing in plants. Plant Methods **18**(1), 137 (Dec 2022). https://doi.org/10.1186/s13007-022-00971-2
7. Flynn, M.J.: Some computer organizations and their effectiveness. IEEE Transactions on Computers **C-21**(9), 948–960 (1972). https://doi.org/10.1109/TC.1972.5009071
8. Gamaarachchi, H., Samarakoon, H., Jenner, S.P., Ferguson, J.M., Amos, T.G., Hammond, J.M., Saadat, H., Smith, M.A., Parameswaran, S., Deveson, I.W.: Fast nanopore sequencing data analysis with SLOW5. Nature Biotechnology **40**(7), 1026–1029 (Jul 2022). https://doi.org/10.1038/s41587-021-01147-4
9. Gigante, S.: Picopore: A tool for reducing the storage size of oxford nanopore technologies datasets without loss of functionality. F1000Research **6**, 227 (2017). https://doi.org/10.12688/f1000research.11022.3
10. Golomb, S.: Run-length encodings (corresp.). IEEE Transactions on Information Theory **12**(3), 399–401 (Jul 1966). https://doi.org/10.1109/TIT.1966.1053907
11. Hendra, C., Pratanwanich, P.N., Wan, Y.K., Goh, W.S.S., Thiery, A., Göke, J.: Detection of m6A from direct RNA sequencing using a multiple instance learning framework. Nature Methods **19**(12), 1590–1598 (Dec 2022). https://doi.org/10.1038/s41592-022-01666-1
12. Kokot, M., Gudyś, A., Li, H., Deorowicz, S.: CoLoRd: compressing long reads. Nature Methods **19**(4), 441–444 (Apr 2022). https://doi.org/10.1038/s41592-022-01432-3
13. Marx, V.: Method of the year: long-read sequencing. Nature Methods **20**(1), 6–11 (Jan 2023). https://doi.org/10.1038/s41592-022-01730-w
14. Meng, Q., Chandak, S., Zhu, Y., Weissman, T.: Reference-free lossless compression of nanopore sequencing reads using an approximate assembly approach. Scientific Reports **13**(1), 2082 (Feb 2023). https://doi.org/10.1038/s41598-023-29267-8
15. Oxford Nanopore Technologies: Oxford nanopore open data. https://labs.epi2me.io/dataindex/ (2022), accessed: 2024-04-03
16. Rice, R.F.: Some practical universal noiseless coding techniques—parts i–iii. Tech. Reps. JPL-79-22, JPL-83-17, and JPL-91-3, Jet Propulsion Lab., Pasadena, CA (Mar, Mar, Nov 1991), originally published in Mar. 1979, Mar. 1983, and Nov. 1991

17. Rissanen, J.: Generalized Kraft inequality and arithmetic coding. IBM Journal of Research and Development **20**(3), 198–203 (1976). https://doi.org/10.1147/rd.203.0198
18. Rivara-Espasandín, M., Balestrazzi, L., Dufort Y Álvarez, G., Ochoa, I., Seroussi, G., Smircich, P., Sotelo-Silveira, J., Martín, Á.: Nanopore quality score resolution can be reduced with little effect on downstream analysis. Bioinformatics Advances **2**(1), vbac054 (Jan 2022). https://doi.org/10.1093/bioadv/vbac054
19. Simpson, J.T., Workman, R.E., Zuzarte, P.C., David, M., Dursi, L.J., Timp, W.: Detecting DNA cytosine methylation using nanopore sequencing. Nature Methods **14**(4), 407–410 (Apr 2017). https://doi.org/10.1038/nmeth.4184
20. Wick, R.R., Judd, L.M., Holt, K.E.: Performance of neural network basecalling tools for Oxford Nanopore sequencing. Genome Biology **20**(1), 129 (Dec 2019). https://doi.org/10.1186/s13059-019-1727-y
21. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23**(3), 337–343 (1977). https://doi.org/10.1109/TIT.1977.1055714