2016 Data Compression Conference

Predictive Coding of Aligned Next-Generation Sequencing Data

Jan Voges+, Marco Munderloh, Jörn Ostermann

Institut für Informationsverarbeitung (TNT)
Leibniz Universität Hannover
Appelstr. 9A
30167 Hannover, Germany
{voges,munderl,office}@tnt.uni-hannover.de

+To whom correspondence should be addressed.

***Abstract**:* Due to novel high-throughput next-generation sequencing technologies, the sequencing of huge amounts of genetic information has become affordable. On account of this flood of data, IT costs have become a major obstacle compared to sequencing costs. High-performance compression of genomic data is required to reduce the storage size and transmission costs. The high coverage inherent in next-generation sequencing technologies produces highly redundant data. This paper describes a compression algorithm for aligned sequence reads. The proposed algorithm combines alignment information to implicitly assemble local parts of the donor genome in order to compress the sequence reads. In contrast to other algorithms, the proposed compressor does not need a reference to encode sequence reads. Compression is performed on-the-fly using solely a sliding window (i.e. a permanently updated short-time memory) as context for the prediction of sequence reads. The algorithm yields compression results on par or better than the state-of-the-art, compressing the data down to 1.9% of the original size at speeds of up to 60 MB/s and with a minute memory consumption of only several kilobytes, fitting in today's level 1 CPU caches.

1. Introduction

Next-generation sequencing (NGS) machines do not read out an entire genomic template. Instead, they produce short random fragments of nucleotide sequences known as sequence reads. A quality score, expressing the confidence that a particular nucleotide has actually been found at a specific location, is associated with each nucleotide in a sequence read. This raw sequencing data generated by NGS machines is commonly stored in FASTQ files [1]. Recently, several tools for FASTQ compression have been developed, which – without performing any kind of internal alignment or assembly – are approaching the Kolmogorov complexity [1]–[4].

FASTQ data can be aligned with tools such as BWA [5], SAMtools [6] or Bowtie2 [7]. Subsequently, the data is stored in Sequence Alignment/Map (SAM) format files which, compared to FASTQ files, contain the alignment/mapping information as well as additional meta data [6], [8]. Mapped sequencing data represented in SAM files contains more redundancy, as multiple sequence reads are typically mapped to the same location on the donor genome. The average amount of reads mapping to the same location is referred to as coverage.

We propose a novel nucleotide sequence compression algorithm called tsc that exploits the redundancy introduced by the coverage as well as redundant information within sequence reads. In contrast to other tools such as Scramble [9] or DeeZ [10], our algorithm does not rely on a reference. Compression is solely performed on the aligned sequence reads. Furthermore, the algorithm only requires partially sorted reads. The prediction context for sequence reads is derived by tracking previously encountered reads in a sliding window (i.e. a short-time memory). In order to encode a new sequence read, the best matching read is selected from the sliding window according to a predefined policy. Before new reads are pushed to the sliding window, they are preprocessed using the alignment information; this can be interpreted as a local assembly of the donor genome. This self-referential sliding window-technique ensures a minute memory footprint and prevents redundant encoding of allelic regions. A major feature of the algorithm is that aligned sequence reads are coded in independent blocks. Random access (RA) to the compressed data is thereby ensured.

The algorithm was furthermore designed to conform to requirements for genome compression issued by ISO/IEC JTC 1 SC 29/WG 11, also known as Moving Picture Experts Group (MPEG) [11].

2. Compression of Aligned Nucleotide Sequences

To introduce this topic, we will briefly examine the SAM file format [8]. Each sequence read combined with mapping/alignment information is stored in a single line of a SAM file. This is referred to as a SAM record in the scope of this paper to avoid ambiguities. A SAM record consists of 11 mandatory fields. Among these are the mapping positions (pos), the so-called CIGAR strings ($cigar$), and the actual nucleotide sequences (seq). The CIGAR strings contain information about inserts and/or deletions (indels). This information is generated during the alignment process. The proposed algorithm jointly encodes these SAM fields, producing a new representation called tsc record.

2.1. Related Work

In the scope of this paper, three tools have been selected for the evaluation and comparison of the compression performance of the proposed algorithm: Scramble [9] (implementing the CRAM file format [12]), DeeZ [10], and Quip [4].

Table 1 – Selected compression tools

Tool	Version	Reference-based	RA
tsc	1.0	N	Y
Quip (normal mode)	1.1.8	N	N
Quip (de-novo assembly mode)	1.1.8	N	N
DeeZ	1.1	Y	Y
Scramble (CRAM 3.0)	1.14.6	Y	Y

Scramble performs a reference-based compression of sequence reads. DeeZ performs a reference-based local assembly internally to obtain the consensus for a given position. Sequence reads are then encoded with reference to the obtained consensus. The third tool, Quip, mainly focuses on FASTQ compression, but also supports SAM file compression. Quip was used in two modes. The first mode make use of high-order Markov chains for the prediction of nucleotides followed by an arithmetic coder. The second mode is employing a de-novo assembly algorithm.

Table 1 shows an overview of the mentioned tools. Tsc is the only tool which does not rely on a reference and at the same time ensures RA to the compressed data.

2.2. Proposed Algorithm

The proposed algorithm solely requires SAM files which contain regions that are sorted by their mapping positions. Figure 1 visualizes a snapshot of subsequent nucleotide sequences from the data sample DH10B (see Table 2). Figure 2 shows the same sequences which now have been shifted according to their mapping positions.

The algorithm is based on the unwinding of the sequence reads by using the mapping positions (as shown in Figure 2) and the additional alignment information provided by the CIGAR strings. The general mechanism is shown in Listing 1. The sample SAM records in rows 1-3 are taken from an example in the SAM file format specification [8]. First, we discuss the SAM record in line 1. The nucleotide sequence has been positioned at mapping position 7. The CIGAR string (`8M2I4M1D3M`) indicates that the first eight bases match to the reference which has been used for alignment (`8M`). According to the CIGAR string, the next two bases have been inserted into the read (`2I`), and so on.

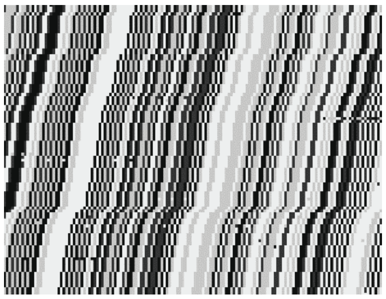
Figure 1 – Sorted reads from data sample DH10B. Different gray-scale values represent the different nucleotides.

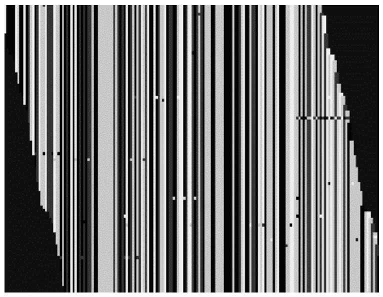
Figure 2 – The same nucleotide sequences, shifted to their mapping positions

Line 5 in Listing 1 shows the positions of the reference genome used for alignment. Finally, line 6 shows the expanded nucleotide sequence from line 1. The sequence has been shifted to its mapping position. The insertions have been skipped from the expanded sequence. The deletion is represented by a question mark.

The process of expanding the nucleotide sequence yields a modified CIGAR string, which we call $stogy$, and an expanded nucleotide sequence, called exs. For the SAM record from line 1, $stogy$ and exs are shown in line 7. Additionally, an "i", indicating the first record in a block (i.e. an I-record), and the mapping position have been prepended to line 7 in Listing 1. The tilde is used to indicate the end of the new sequence read representation. Analogous to a SAM record, we call this joint representation a tsc record.

After encoding the first read, the mapping position pos and the expanded read exs are pushed to a circular buffer of size N_w which implements the sliding window. The following SAM records are also expanded and pushed to the circular buffer, again obtaining $stogy$ and exs. Subsequently, the best matching read (pos_{ref} and exs_{ref}) is selected from the sliding window. The position offset $pos_{off} = pos - pos_{ref}$, as well as the expanded reads exs and exs_{ref} are then used to compute the modifications mod from exs to exs_{ref} and/or the trailing sequence part $trail$ from exs with respect to exs_{ref}. In the next step, the tsc records are composed using pos_{off}, $stogy$, mod, and $trail$, as shown in lines 10 and 13 in Listing 1.

```
7   8M2I4M1D3M TTAGATAAAGGATACTG
9   3S6M1P1I9M AAAAGATAAGGATAACTGG
9   5S6M       GCCTAAGCTAA

3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23
                T   T   A   G   A   T   A   A   G   A   T   A   ?   C   T   G
 POS      STOGY                 EXS
i7:8M2IAG4M1D3M:TTAGATAAGATA?CTG~

            A   A   A   A   G   A   T   A   A   G   A   T   A   A   C   T   G   G
POSOFF   STOGY      TRAIL
2:3SAAA6M1P1IG9M:G~

    G   C   C   T   A   A   G   C   T   A   A
POSOFF STOGY   MOD
0:5SGCCTA6M:2C~
```

Listing 1 – Exemplary operating principle of the proposed algorithm

In addition to the first record in a block and to a normally predicted record, there might be incomplete SAM records. This yields in total three different types of tsc records (square brackets indicate optional parts):

1. I-records: $ipos{:}\,stogy{:}\,exs\sim$
2. If pos, $cigar$, or seq is missing: $m[pos{:}\,][cigar{:}\,][seq]\sim$
3. (Normally) predicted record: $pos_{off}{:}\,stogy[{:}\,mod][{:}\,trail]\sim$

Tsc records of type 2 are not pushed to the sliding window. Furthermore, new I-records are also produced if the position offset pos_{off} is larger than the length of the expanded reference sequence exs_{ref}. Upon constructing a new I-record, the sliding window is cleared and a new block might be started. New I-records are also inserted every N_b records to ensure block-wise RA to the compressed data.

The ability to add new I-records ensures that the algorithm also works on only partially sorted input data.

The sliding window is tracking N_w previous encountered sequence reads. In order to select the best matching read from the sliding window, a metric called neo is computed for each tsc record and also pushed to the circular buffer. Let $len(arg)$ denote the length of the argument arg. With this notation, the parameter α can be used to adjust the selection of reference reads from the sliding window, as shown in Equation 1.

$$neo = (1-\alpha)\cdot pos_{off} + \alpha\cdot\big(len(stogy) + len(mod)\big) \text{ with } 0 \le \alpha \le 1 \qquad \textbf{(1)}$$

The best matching record is then the one with neo_{ref} as shown in Equation 2.

$$neo_{ref} = \min_{0\le w<N_w}\big(neo - neo(w)\big) \qquad \textbf{(2)}$$

The I-records have a neo value calculated according to Equation 3.

$$neo = \alpha\cdot len(stogy) \qquad \textbf{(3)}$$

The tsc records as a new sequence read representation are ultimately passed to a dictionary-based codec in order to exploit the redundancy along the tsc record stream. Specifically, error-free and properly aligned sequence reads produce a tsc record stream which is highly suitable for dictionary-based compression because of repetitions in the data stream. In this paper, zlib [13], [14] has been used. It is based on the DEFLATE algorithm [15], which is in turn a combination of the LZ77 algorithm [16] and Huffman coding [17].

3. Evaluation and Results

The following sections describe the selection of test data, the test conditions, and compression results. In addition to the overall compression results, we evaluated the block-wise CRs of the proposed algorithm to verify its operating principle.

3.1. Test Data

MPEG issued a reference collection of NGS data which was compiled according to specific criteria to ensure a wide variety of challenges for existing and future compression tools [18]. Therefore, the algorithms are evaluated using SAM/BAM files from the MPEG database. The selected data is shown in Table 2.

Table 2 – Selected test data from the MPEG database

ID	BAM size (GB)	Record count	Coverage
cancermixed/HCCmix	123.0	942,395,158	26x
DH10B	1.4	13,175,679	448x
LID8465/K562	13.0	246,476,391	16x
ERR317482WGS/9827#49	6.1	56,463,236	2.3x
ERP002490/NA12878	106.0	1,286,194,550	26x

3.2. Implementation

The proposed compression algorithm was implemented in a software also called tsc, which has been developed at Institut für Informationsverarbeitung. Tsc is a highly modular software: after parsing a SAM file and splitting it into the twelve distinct streams (one stream for each SAM field), each stream or a combination of streams can be passed to several so-called sub-block codecs. One of these sub-block codecs is dedicated to implementing the proposed algorithm. It jointly encodes the mapping positions, the CIGAR strings, and the nucleotide sequences. Zlib version 1.2.8 is used to obtain the binary data.

In the scope of this paper, the parameters for the proposed algorithm were empirically derived by finding a good trade-off between compression ratio and runtime. The sliding window size of the nucleotide sequence compressor was set to $N_w = 10$, and the selection of context reads from it was done using Equations 1-3 with $\alpha = 0.5$. Figure 3 shows the average tsc CR for the first 100,000 reads from all data samples in Table 2 over the block size N_b. As the CR converges toward larger block sizes, a block size of $N_b = 10{,}000$ records has been chosen for the experiments. Since the size of the sliding window is limited to $N_w = 10$ context reads, the tsc algorithm requires only tiny amounts of memory. If we, for example, assume an average read length of 200 base pairs, then the sliding window consumes about $10 \cdot (8\,B + 4\,B + 200\,B) \approx 2\,kB$ (8 bytes are used for neo, and 4 bytes are used for pos). This amount of data easily fits into level 1 caches of modern CPUs. In contrast, for the compression of the data samples from Table 2, the memory consumption of Scramble ranges from about 0.2 GB up to 1 GB. Quip uses about 1 GB, and DeeZ uses roughly 1.4-3.6 GB.

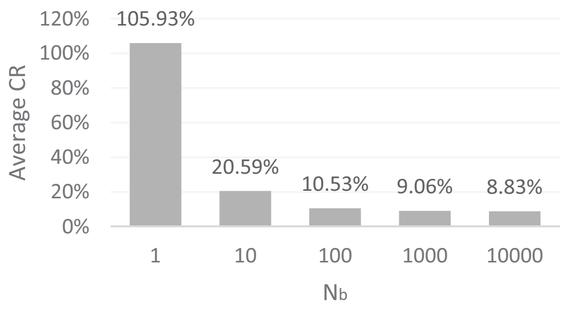

Figure 3 – Average tsc CR versus block size

3.3. Compression Results

In this section, the compression ratio (CR) is defined as $CR = \frac{s_{comp}}{s_{uncomp}}$, where s_{comp} and s_{uncomp} denote the compressed and uncompressed size, respectively.

Figure 4 shows the CRs obtained using tsc, Scramble, DeeZ, and Quip (in both the statistical and the de-novo assembly mode) on the data from Table 2. It is worth mentioning that the tools operate with slightly different input and output signals. The uncompressed sizes for each data sample were computed by accumulating the sizes of the SAM fields RNAME, POS, CIGAR, and SEQ. The compressed sizes were computed by accumulating those parts of the individual compressed bit streams that are associated with one of the mentioned SAM fields.

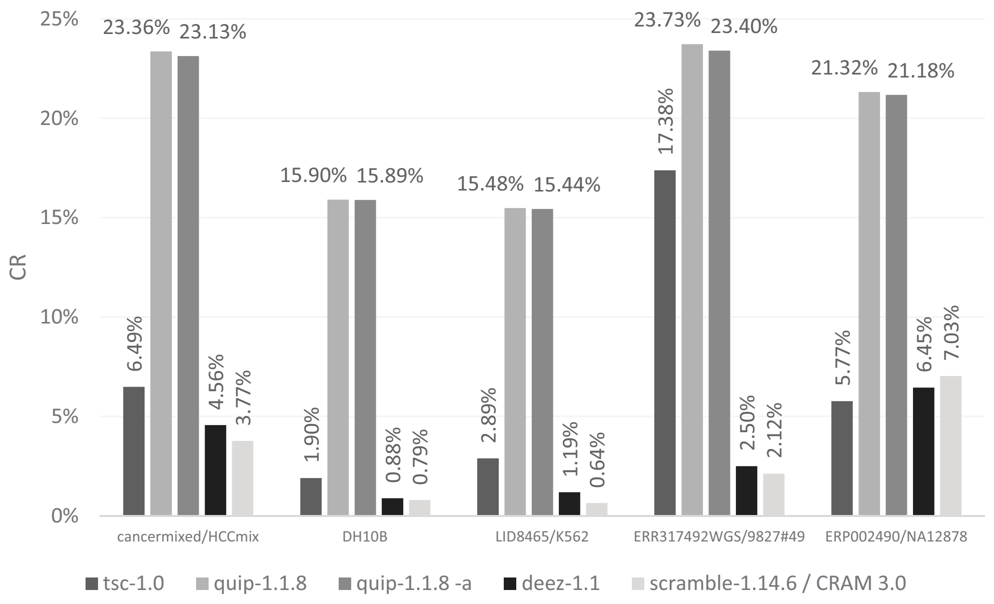

Figure 4 – Comparison of the sequence read compression results

Tsc outperforms all competing tools for the data sample ERP002490/NA12878, as shown in Figure 4. Furthermore, tsc performs better than Quip on all data samples and is only outperformed by the reference-based tools. Data sample ERR317492WGS/9827#49 has a very low coverage of only 2.3x as shown in Table 2. Hence, the proposed algorithm does not perform well compared to the reference-based compressors. However, on average, tsc is able to compress the data down to 6.88% of the original size.

The compression speed of the tsc software was measured on an Intel® Core^{TM} i7-3770K CPU with 8 cores @ 3.50 GHz and 32 GB RAM. Tsc achieves compression speeds ranging from about 10 MB/s for LID8465/K562 up to 60 MB/s for DH10B. The decompression is roughly twice as fast. This indicates that the tsc compression and decompression speeds are only bounded by the employed dictionary-based codec. A table with file pointer offsets to the compressed blocks is stored in the tsc file header. Therefore, RA speed is only bounded by the speed of file seek operations – if the desired block has been found, it can be decompressed at full speed.

3.4. Block-wise Evaluation of Compression Ratios

To evaluate the block-wise compression performance of the proposed algorithm, several values have been monitored for each block. Among these are:

- The mean length of the $stogy$ string per block
- The mean length of the mod string per block
- The compression ratio per block

Each indel coded in the $stogy$ string and each modification (stored in mod) increase the size of a tsc record. Therefore, additional indels or modifications should impose an increase in CR. Nevertheless, due to the self-referential algorithm, mutations and/or allelic regions in the donor genome should not increase the CR. Thereby, low-error alignments should be ideally suited for compression with the proposed algorithm. To show this behavior, we define the average number of edits per block as shown in Equation 4.

$$edits = \frac{1}{2N_b} \cdot \sum_{b=0}^{N_b-1} len\big(stogy(b)\big) + \frac{1}{2N_b} \cdot \sum_{b=0}^{N_b-1} len\big(mod(b)\big) \tag{4}$$

The distribution of block-wise CRs over the average number of edits per block for the data samples from Table 2 is shown in Figure 5 to Figure 8. Furthermore, the Pearson correlation coefficient r and the Spearman correlation coefficient ρ were added to the figures. The Spearman correlation coefficient is less sensitive to outliers than the Pearson correlation coefficient and is thus more suitable for the data. With ρ ranging from 0.20 up to 0.99, it is clearly shown that the obtainable CRs are correlated with the errors introduced by the sequencing technology and poor alignments. Emerging sequencing technologies yielding fewer errors or alignment tools finding better alignments should produce data that can be compressed at much lower bit rates using the proposed algorithm.

4. Conclusion

It has been shown that the tsc algorithm for the compression of aligned nucleotide sequences is able to reduce the storage size of the combined mapping positions, CIGAR strings, and nucleotide sequences down to 1.9% for the dataset DH10B. On average, the algorithm is able to compress the data to about 6.88% of the original size and is only outperformed by the reference-based tools. Furthermore, tsc performs better than Quip on all data samples and outperforms all competing tools on the dataset ERP002490/NA12878.

In addition to its competitive compression ratios, a major advantage of the algorithm is its minute memory consumption and its low complexity. In contrast to the other tools which consume up to several gigabytes of memory, the context data used by the proposed algorithm solely consumes several kilobytes. This amount of data easily fits into level 1 caches of modern CPUs. Furthermore, the context for the prediction of aligned sequence reads is only derived from previously encountered reads. This naturally prevents the redundant encoding of allelic regions in the donor genome. Moreover, no reference is required for compression respectively decompression.

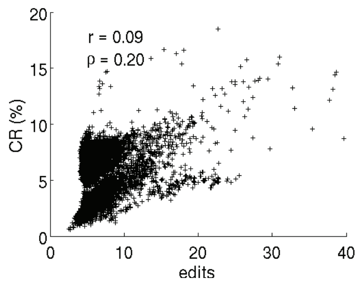

Figure 5 – cancermixed/HCCmix

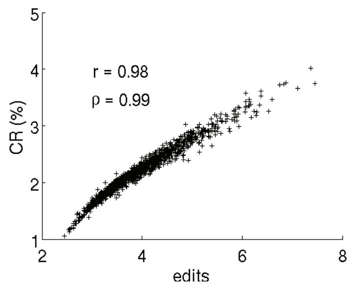

Figure 6 – DH10B

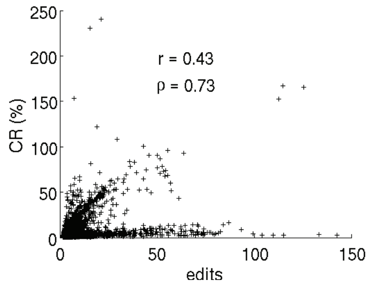

Figure 7 – LID8465/K562

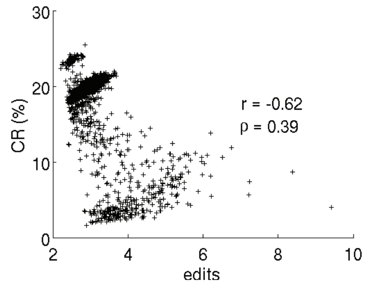

Figure 8 – ERR317492WGS/9827#49

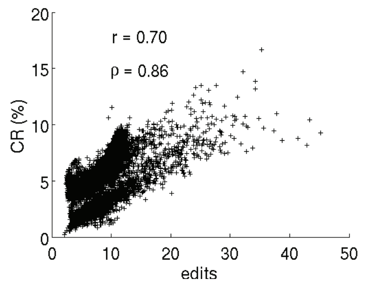

Figure 9 – ERP002490/NA12878

In addition, the algorithm is suitable for only partially sorted input data. Competitive compression ratios can be obtained, while at the same time fast and fine-grained random access to the compressed data is ensured. Due to these features and a novel compressed record structure, computation on the compressed data is encouraged.

Furthermore, it has been shown that the obtained compression ratios depend on the error rates of the employed sequencing technologies and alignment tools. Low-error data produced by emerging sequencing technologies will have an immediate positive impact on the compression ratios obtained by the proposed algorithm.

References

[1] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, “The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants,” *Nucleic Acids Res.*, vol. 38, no. 6, pp. 1767–1771, 2010.

[2] J. K. Bonfield and M. V. Mahoney, “Compression of FASTQ and SAM Format Sequencing Data,” *PLoS One*, vol. 8, no. 3, p. e59190, 2013.

[3] F. Campagne, K. C. Dorff, N. Chambwe, J. T. Robinson, and J. P. Mesirov, “Compression of structured high-throughput sequencing data,” *PLoS One*, vol. 8, no. 11, p. e79871, Jan. 2013.

[4] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze, “Compression of next-generation sequencing reads aided by highly efficient de novo assembly,” *Nucleic Acids Res.*, vol. 40, no. 22, pp. e171–e171, 2012.

[5] H. Li and R. Durbin, “Fast and accurate short read alignment with Burrows-Wheeler transform.,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–60, 2009.

[6] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, “The Sequence Alignment/Map format and SAMtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[7] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with Bowtie 2.,” *Nat. Methods*, vol. 9, no. 4, pp. 357–9, 2012.

[8] The SAM/BAM Format Specification Working Group, “Sequence Alignment/Map Format Specification,” 2014.

[9] J. K. Bonfield, “The Scramble conversion tool.,” *Bioinformatics*, vol. 30, no. 19, pp. 2818–9, Oct. 2014.

[10] F. Hach, I. Numanagić, and S. C. Sahinalp, “DeeZ: reference-based compression by local assembly.,” *Nat. Methods*, vol. 11, no. 11, pp. 1082–4, Nov. 2014.

[11] C. Alberti, M. Mattavelli, L. Chiariglione, I. Xenarios, N. Guex, H. Stockinger, T. Schuepbach, P. Kahlem, C. Iseli, D. Zerzion, D. Kuznetsov, Y. Thoma, E. Petraglio, C. Sahinalp, I. Numanagić, J. Bonfield, V. Zalunin, J. Delgado, W. Allasia, S. Cheng, and J. Voges, “Requirements on Genome Compression and Storage.” ISO/IEC JTC 1/SC 29/WG 11 (MPEG), Document number N15345, Warsaw, 2015.

[12] M. Hsi-Yang Fritz, R. Leinonen, G. Cochrane, and E. Birney, “Efficient storage of high throughput DNA sequencing data using reference-based compression,” *Genome Res.*, vol. 21, no. 5, pp. 734–740, Jan. 2011.

[13] J.-L. Gailly and P. Deutsch, “ZLIB Compressed Data Format Specification version 3.3.” Network Working Group, 1996.

[14] P. Deutsch, “GZIP file format specification version 4.3.” Network Working Group, 1996.

[15] P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3.” Network Working Group, 1996.

[16] J. Ziv and a. Lempel, “A universal algorithm for sequential data compression,” *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[17] D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proc. IRE*, vol. 40, no. 9, pp. 1098–1001, 1952.

[18] C. Alberti, M. Mattavelli, L. Chiariglione, I. Xenarios, N. Guex, H. Stockinger, T. Schuepbach, P. Kahlem, C. Iseli, D. Zerzion, D. Kuznetsov, Y. Thoma, E. Petraglio, C. Sahinalp, I. Numanagić, and J. Delgado, “Database for Evaluation of Genome Compression and Storage.” ISO/IEC JTC 1/SC 29/WG 11 (MPEG), Document number N15092, Geneva, 2015.