A Novel Encoding Algorithm for Textual Data Compression

Anas Al-okaily[1,*] **and Abdelghani Tbakhi**[1]

[1]Department of Cell Therapy and Applied Genomics, King Hussein Cancer Center, Amman, 11941, Jordan
[*]anas.al-okaily@uconn.edu

ABSTRACT

Data compression is a fundamental problem in the fields of computer science, information theory, and coding theory. The need for compressing data is to reduce the size of the data so that the storage and the transmission of them become more efficient. Motivated from resolving the compression of DNA data, we introduce a novel encoding algorithm that works for any textual data including DNA data. Moreover, the design of this algorithm paves a novel approach so that researchers can build up on and resolve better the compression problem of DNA or textual data.

Introduction

The aim of compression process is to reduce the size of data as much as possible to save space and speed up transmission of data. There are two forms of compression processes: lossless and lossy. Lossless algorithms guarantee exact restoration of the original data, while lossy do not (due, for instance, to exclude unnecessary data such as data in video and audio where their loss will not be detected by users). The focus in this paper is the lossless compression.

Data can be in different formats such as text, numeric, images, audios, and videos. For textual data, which is the main focus in this paper, several encoding algorithms have been developed, namely and mainly, Huffman[1] (with several versions, namely, minimum variance Huffman, canonical Huffman, length-limited Huffman, non-binary Huffman, and adaptive Huffman), Faller-Gallager-Knuth[2] (an adaptive Huffman), Vitter[3] (an adaptive Huffman), Shannon[4], Shannon-Fano[5], Shannon-Fano-Elias[6], Lempel-Ziv[7] (abbreviated as LZ with several extensions such as Lempel-Ziv-Welch[8] (LZW), Lempel-Ziv-Stac[9] (LZS), Lempel-Ziv-Oberhumer[10] (LZO), Lempel-Ziv-Storer-Szymanski[11] (LZSS), and Lempel–Ziv-Ross-Williams[12] (LZRW)), Burrows-Wheeler[13] (BW), and Tunstall[14] algorithms. There are several other algorithms that are named based on the type of process involved in the algorithm such as arithmetic encoding[15], range encoding[16], Move-to-Front encoding,[17,18] (also referred as symbol ranking encoding), run-length-encoding[19], delta encoding, unary encoding, context tree weighting encoding[20], prediction by partial matching[21], context mixing[22], asymmetric numeral systems[23] (referred also as asymmetric binary coding, finite state entropy is a variant of this entropy), LIPT[24], and dynamic Markov encoding[25].

The main and common tools that implement one or more encoding algorithms and compress textual data are listed in Table 1. These tools are the ones that will be used to compare the results of the proposed algorithm with.

Compression algorithms can be classified based on the method type used in the algorithm. There are several method types used which are mainly: entropy, dictionary-based, predictive, and transform methods. A brief explanation of these methods is provided in the next two paragraphs but several and recent reviews and surveys that describe these methods are provided in several sources[26–30].

The aim of entropy encoding is to represent the most frequent characters or words with less bits and vice versa, instead of using the same number of bits to represent every character as is used in standard representations such as ASCII[31] encoding (8 bits for each character) and unicode encodings. Bits stands for binary digit and it is the basic unit of information in computing and digital communications with the value of 0 or 1. Examples of entropy encoding include Huffman[1] encoding and its family, Shannon[4], Shannon-Fano[5], Shannon-Fano-Elias[6], Tunstall[14], asymmetric numeral systems[23], range encoding[16], arithmetic encoding[15], and unary encoding. Predictive methods are based on employing the current decoded/decompressed data to predict the upcoming data. Context tree weighting encoding[20], prediction by partial matching[21], context mixing[22], and dynamic Markov encoding[25] are examples of algorithms that are based on prediction method.

The approach of dictionary encoding is to find the most repetitive patterns (characters or words) in the text and storing them in a dictionary then encode these patterns by its index in the dictionary. The dictionary can be built statically or dynamically. In static mode, same dictionary is used for all the compression process and is built ahead; this is suitable when a considerable prior knowledge about the text is available in advance. In the dynamic mode, the dictionary is built and updated during the compression process. This is achieved by employing a sliding window and then the dictionary is built from the previously encoded sequence within the sliding window. All the family of LZ[7] algorithm and byte-pair algorithm[32] apply this method. In transform methods, the text is converted to a representation that can be compressed more efficiently than the original representation. Examples mainly include move-to-front encoding, RLE encoding, delta encoding, LIPT, and BWT.

Textual data include genomics (DNA) data where there are several characteristics of this type of data. Firstly, the alphabet of DNA data consists of A, C, G, and T characters representing four nucleotides which are adenine, cytosine, guanine, and thymine; respectively. Secondly, DNA data contain repeats and palindromes. Thirdly, the size is very large. The human genome for instance consists of more than three billions of nucleotides. Moreover, sequencing genomic/DNA data is currently being performed in a daily basis around the globe and usually with high depth (30-100x). This generates in a daily basis a huge volume of data. Hence, many algorithms have been developed to handle the compression of this type of data. Compression algorithms involve two modes as introduced in[66], algorithms that utilize reference genome/source (vertical mode) and algorithms that are reference-free (horizontal mode). The focus in this paper is on reference-free mode. DNA data represented as genomes are mainly stored in FASTA format, many algorithms have been developed for this format, namely, BIND[67], Biocompress[68], Biocompress2[66], CBSTD[69], Cfact[70], CoGI[71], DELIMINATE[72], DNABIT[73], DNACompact[74], DNA-COMPACT[75], DNACompress[76], DNAC-SEB[77], DNAEcompress[78], DNAEnc3[79], DNAPack[80], DNASC[81], DNAX[82], GeCo[83], GeCo2[84], GENBIT[85], GenCompress[86], GeNML[87], HUFFBIT[88], improved RLE[89], JARVIS[90], MFCompress[91], modified HUFFBIT[92], NAF[93], NMLComp[94], NUHT[90], OBComp[95], OBRLDNAComp[96], POMA[97], Pufferfish[98], SBVRLDNACOMP[99], SeqCompress[100], UHT[101], XM[102], modified RLE[103], and other algorithms were proposed also in[104–109]. Several surveys and comparisons studies for DNA data are provided in the following sources[77,91,110–112].

DNA data involve mainly genomics data where the genome (set of tens of chromosomes) is stored mainly in FASTA format[113] (which is the main focus in this paper) and sequencing reads (millions/billions of reads where length of reads is in hundreds) which are stored with the quality of each base in mainly FASTQ format[114]. The number of bases that were sequenced since 1982 until June 2020 is 42,782,325,8901[115]. Many tools and algorithms have been developed to compress FASTQ format, namely, ALAPY[116], Assembltrie[117], BEETL[118], BdBG[119], DSRC[120], DSRC2[121], FaStore[122], Fastqz[123], Fqpack[124], FQSqueezer[125], Fqzcomp[123], GenCompress[126], GenoDedup[127], G-SQZ[128], GTZ[129], HARC[130], KIC[131], KungFQ[132], Leon[133], LFastqC[134], LFQC[135], LW-FQZip[136], MetaCRAM[137], Mince[138], Minicom[139] , Minimizers[140], Orcom[141], Quip[142], SCALCE[143], SeqDB[144], SlimGene[145], SOLiDzipper[146], SPRING[147], and in these papers[148]. Survey of this type of compression are published at[91,123].

Another common format used for storing/processing DNA data is SAM format[149]. Several algorithms have been developed to compress this format, namely, CRAM[150], DeeZ[151], Goby[152], HUGO[153], NGC[154], Quip[142], SAMcomp[155], SAMtools[149], SAMZIP[156], and Scramble[157]. Surveys of this type of compression are published at[91,123].

DNA data mostly contain characters with discrete uniform distribution (frequencies), hence, the compression process might not lead to high compression results as each character (base) will eventually assigned with almost same number of bits. If each character is represented by the same number of bits, then a naive and standard compression will be obtained which is not the aim of compression algorithms and not exploiting the repetitions contained in DNA data. As an example, instead of storing a sequence of twenty As followed by ten Ts, four Gs, then two As with 288 bits (36 bases $\times$ 8 bits) using ASCII encoding; instead and using Huffman encoding, the character A will be represented by one bit (0), T with 2 bits (10), G with three bits (110), and C with three bits (111). The cost then for storing the sequence will be 58 bits instead of 288 bits. Now, if the characters in the sequence are distributed as 12 for A, 9 for T, 8 for G, and 6 for C, then Huffman encoding will assign 2 bits for each character (A:00. C: 01, G:10, and T:11, which lead to store the sequence with 72 bits); this is reasonable encoding but yet is not exploiting efficiently or optimally the repetitions in DNA sequence.

Now, given the small alphabet size of DNA data, which is better for the compression process, the main challenge however is the uniform distribution of the frequencies of the four characters. Motivated from resolving the compression problem for DNA data and from this aforementioned challenge, we introduce a novel encoding algorithm that is lossless, reference-free, resolves this challenge, and is applicable not only to DNA data to any textual data. By implementation, the algorithm shows better compression results indeed. As it has been a tradition to name the encoding algorithms by names, and to exploit this fact to honoring two great professors, Last author and Pramod Srivastava (Professor at Department of Immunology, University of Connecticut School of Medicine) who highly and exceptionally advised the first author in his academic career, kindly refer to this algorithm by Okaily-Srivastava-Tbakhi (*OST*).

Methods

One of the challenges in compressing DNA data is the uniform distribution of characters (bases). However, two observations can be drawn from a careful look to the DNA data. Firstly, looking to the regional (local) content, it's clear that many sub-sequences (let's say sub-sequences of length 100 bases) contain non-uniform/skewed distributions. Secondly, sequences which are similar and better if they are compressed and encoded together are distant from each other with distance longer than the length of the sliding window (length usually is of kilobytes) that is used in algorithms such as LZW, context weighting tree, predictive partial

Table 1. Common general-purpose compression tools

Tool	Algorithms used	Version	Last update
bcm[33]	based on BW algorithm	v1.51	2020
blzpack[34]	based on brieflz library[35] where this library is a small and fast open source implementation of a LZ style compression algorithm	v1.3.0	2020
brotli[36]	Google tool which combines LZ, Huffman encoding, and 2nd order context modelling	v1.0.7	2018
bsc[37]	based on block sorting	v3.1.0	2016
bzip2[38]	uses the BW algorithm	v1.0.8	2019
cmix[39]	a program aimed to optimizing compression ratio at the cost of high CPU/memory usage	v18	2019
compress[40]	fast and simple compressor based on LZW algorithm	v4.2.4	2011
Freeze[41]	combines LZSS and Huffman encoding	v2.5.0	1993
gzip[42]	uses a combination of LZSS and Huffman coding	v1.10	2018
hook[43]	an implementation on dynamic Markov encoding	v0.8e	2007
Huffman-codec[44]	based on Huffman encoding	v1.0	2020
lizard[45]	contains four methods such as fastLZ4 + Huffman	v1.0	2019
lrzip[46]	an extension of rzip but replaces bzip2 by LZMA, LZO, or no second-stage	v0.616	2013
lz4[47]	belongs to the LZ77 family of byte-oriented compression schemes	v1.9.2	2019
lzb[48]	based on hexadecimal and base64 encoding/decoding	v1.0	2019
lzfse[49]	Apple tool that combines LZ and Finite State Entropy	v1.0	2017
lzip[50]	employs the Lempel–Ziv–Markov chain algorithm	v1.19	2017
lzop[51]	implements LZO algorithm	v1.04	2017
lzturbo[52]	compressor based on lZ77 algorithm	v1.2	2014
Nakamichi[53]	based on LZSS algorithm	v1.0	2020
pigz[54]	a parallel implementation of gzip	v2.4	2017
ppm[55]	an implementation of prediction by partial matching	v1.0	2020
qzip[56]	a gzip-like program which uses quicklz[56] compression library where quicklz library is based on LZRW algorithm	v0.2	2011
rans_static[57]	based on arithmetic encoding	v1.0	2016
rzip[58]	encodes large chunks of duplicated data then use bzip2	v1.0.6	2010
snzip[59]	uses Snappy[60] which is a Google library based on ideas from LZ77	v1.0.4	2016
srank[61]	an implementation of Move-to-Front encoding (symbol ranking)	v1.0	1997
xz[62]	a variant algorithm of LZ77 with huge dictionary sizes and special support for repeatedly used match distances where the output is then encoded with a range encoder	v5.2.2	2015
Zlib[63]	variation of LZ algorithm	v1.2.11	2017
zpipe[64]	is a ZLib implementation of gzip and deflate support	v1.0	2017
zstd[65]	Facebook tool that combines a LZ with a large search window and a fast entropy coding using both Finite State Entropy and Huffman encoding	v1.4.5	2020

matching, or dynamic Markov compression (even if these sequences are within the sliding window they are mostly distant from each other to a degree that makes encoding them costly and not efficient). These two observations motivated the design of a novel encoding algorithm.

Generally speaking, the algorithm is to scan the DNA data with non-overlapping windows, label the sequence within each window, concatenate it with the sequences in a bin correspondent to that label, and then output the label into an stream. Now, encode the labels of the bins based on, for instance, the number of the sequences in the bins; compress the stream of labels using the label codes. Then, compress the sequences in each bin using suitable compression algorithm depending on the content of the sequences in that bin. Finally, compress all the compression results (bins and stream of labels) together. The decompression process will be by firstly decompressing each bin and the stream of labels. Then, read the labels sequentially from the stream of labels, at each reading and using a counter for each bin, get the next sequence (of length same as the length of the non-overlapping window used during the compression process) from the bin of that label then increment the counter at that bin.

Compression algorithm. The steps of the algorithm for compressing a string S of length s and contains character from a fixed alphabet of length Σ, window length of w, sequence QL initialized to empty. The algorithm is as follows.

1. Scan S with non-overlapping windows of length w, where at each window:

   (a) Create label for the characters in the window so that this label classifies the sequence and where there is a bin correspondent to each label. The label can be set dynamically or designed in advance.

   (b) Concatenate the label string with QL, that stores the queue/order of the labels of the sequences, given that the string of each label must be unique.

   (c) Concatenate the sequence of the window with the sequences in the bin correspondent to that label.

   The time cost for this step is $O(\frac{sM}{w})$ where M is the cost for classifying each sequence of length w. If M is less than or equal to w, then the cost of this step is $O(s)$. Moreover, the value of w can be fixed or variable (variable so that the window is extended until the label of the sequence in the window matches one of the label of the bins). If w is variable, then it must be added to QL (after the string of the label for each window) and encoded using a universal code for integers such as Levenshtein coding[158], Elias coding[159] (delta, gamma, or omega), exponential-Golomb code[160], Fibonacci code[161], Stout code[162], or using a suitable encoding algorithm such as unary, binary, or Huffman encoding.

2. Once scanning S is finished, encode all collected labels using Huffman encoding for instance. The time cost of this step is $l log l$ where l is the number of labels that the user or the implementer is set.

3. Compress QL using the encoding schema from the previous step. The time cost of this step is $O(\frac{s}{w})$.

4. Compress the sequence in each bin with a compression algorithm suitable to the content of the sequences in the bin.

5. Compress/archive all resultant compressed files (bins and QL) together.

Decompression algorithm.

1. Extract all files of compressed bins and compressed QL.

2. Decompress all files (bins and QL).

3. Initialize a counter for each bin.

4. Read from QL the strings of the labels sequentially where at each read extract the sequence of length w starting from position equal to $counter$, increment $counter$ by the value of w, then output the extracted sequence to the output stream.

Clearly, the time and memory cost of the decompression process is linear.

Analysis of the algorithm. The main process in the algorithm is clearly data-binning process. Each sequence of length w (or of variable length) in the S must be labeled and binned into the bin correspondent to that label. Then, the sequences which are sharing the same label and are binned together to the same bin are encoded using the same encoding algorithm. In addition, the sequences in each bin can be encoded using different encoding algorithm than the sequences in other bins. The design of the algorithm calls for several research questions which are still needed to be investigated. Given a set of sequences (concatenated together or not) and sharing similar characteristics (such as same Huffman tree), what is the efficient or optimal encoding solution for them? What is the optimal classification method to label the bins? What is the optimal number of bins to use? Is better to adopt a window length that is fixed, variable, or dependent on the text/genome length?

Data binning process has been used in solving compression problem but for sequencing reads not for genome sequences or textual data. The following tools, Assembltrie[117], BEETL[118], BdBG[119], FaStore[122], HARC[130], Mince[138], Orcom[141], and SCALCE[143], apply data binning process but there are several differences between these algorithms and OST algorithm. Firstly, the inputs are already sequencing reads with short length (few hundreds bases) unlike the genomes or textual data. Secondly, sequencing of such reads are usually conducted at high coverage, hence, there are many redundancy in them and this is one of the motivations to apply binning process to compress these reads. Thirdly, none of these tools encode the the label of bins, in order to preserve the order of the reads in the original inputs, where this is due to the observation that the order of the reads in the input does not hold meaningful information so it can be omitted. However, a main step in OST algorithm is to encode the labels of the bins and record in QL the order these labels to preserve the order of the sequences so that during decompression the original DNA data can be restored. Lastly, applying binning process tom compress textual data (nor DNA/genomics data) was not addressed carefully in the literature.

The design of OST algorithm contributes in organizing and sorting the compression process by a divide-and-conquer methodology. The creation of bins for similar text and compress them instead of being dependent on a single encoding schema for all data as in entropy methods, or dependent on the previous sequence and its randomity (or differences with the upcoming sequence) to predict the upcoming sequences as in the prediction methods or to use in building a dictionary as in dictionary methods, or to transform the text into another text. This is supported with two observations as will be discussed and shown in the results section. Firstly, the bins with same label were compressed with same efficiency across different genomes and regardless of the content-type of the genomes (as an example repetitive or not repetitive as some genomes are known to be repetitive and others are not). Secondly, different compression algorithms seems to be better to compress different bins. Hence, OST design leads to more control and organizing of the content of the data being compressed, and what is more suitable to compress each bin rather than having the encoding/compression process dependent on the content of the data and dependent only on a single or a main algorithm.

OST-DNA This is the first implantation of OST algorithm in which the data is DNA data. The labels of the bins is computed using the encoding schema of Huffman tree of the content of the sequence. As an example, if the encoding schema is G:0, A:10, T:110, and C:111, then the label is GATC_1233 (1 indicates that G is encoded using 1 bit, A using 2 bits, and so on). The maximum number of labels (bins) hence is 65.

Note that as the windows are non-overlapping, then each nucleotide in S will be read $O(1)$ time. The cost for building Huffman tree is O($\Sigma log \Sigma$), where the number of times that Huffman trees needs to be built is $O(\frac{s}{w})$ times; hence, the cost of building all Huffman trees is $O(\frac{s\Sigma log\Sigma}{w})$. Now, due to the pigeonhole principal and to allow obtaining non-uniform distribution for the nucleotides (characters) in Σ within the content of the windows, the value of w needs to be larger than the value of $\Sigma log \Sigma$ noting that Σ is a constant. Therefore, the time cost for this step is $O(s)$.

As w value is fixed in this version of OST-DNA and is equal to...., Huffman tree is built once for each window's sequence. For the case of variable w value (hence the window will be extending until the the label of the window's sequence match one of the labels of the bins), then it will not be efficient to compute Huffman tree for the whole sequence at every extension; instead, dynamic Huffman tree can be applied.

Lastly, in order to archive all the compressed bins and QL, we tested several archiver tools such as ..... and decided to use 7z tool as it produced the best compression results.

Results

OST-DNA is implemented using python language. The code is not highly optimized in terms of speed and memory usage in both compression and decompression, but is sufficient as the main goal of this version of implementation is to proof the concept of the OST algorithm and conduct a trial testing and investigation. A set of genomes as listed in Table 2 and 3, which is the same dataset used in[110], were used for testing and comparing the results of OST-DNA with the results of the tools in Table 1.

A main metric of the performance of compression process is compression ratio which is equal to the size of the compressed file divided by the size of uncompressed (original) file. Other metrics also are considered in this paper which are compression speed (MB/s) which is equal to the size of uncompressed file (in MB) divided by the compression time, and decompression speed (MB/s) which is equal to the size of the uncompressed file (in MB) divided by decompression time.

Firstly, all newlines and headers were removed from the genomes to obtain one sequence for the whole genome. Then we run all tools in Table 1 for all tested genomes. The results are provided in Table 4. For trialing, we used seven tools: bcm, brotli, bsc, bzip2, lrzip, lzip, and xz to compress the bins generated by OST-DNA algorithm. Moreover, we run each of these seven versions of OST-DNA when the value of w is 25, 50, 100, 125, 150, 200, 300, 400, 500, 600, 800, 1000, 2000, 3000, 4000, 5000, and 10000 and with different set of labels (as will described later). The minimum value obtained by these runs for each version on each tested genome is provided in Table 4.

Table 2. Genome sequence datasets

Category	Organism	Accession	Size
Virus	Gordoniaphage GAL1[163]	GCF_001884535.1	50.7 kB
Bacteria	WS1 bacterium JGI 0000059-K21[164]	GCA_000398605.1	522 kB
Protist	Astrammina rara[164]	GCA_000211355.2	1.71 MB
Fungus	Nosema ceranae[164]	GCA_000988165.1	5.81 MB
Protist	Cryptosporidium parvumIowa II[164]	GCA_000165345.1	9.22 MB
Protist	Spironucleus salmonicida[164]	GCA_000497125.1	13.1 MB
Protist	Tieghemostelium lacteum[164]	GCA_001606155.1	23.7 MB
Fungus	Fusarium graminearumPH-1[163]	GCF_000240135.3	36.9 MB
Protist	Salpingoeca rosetta[164]	GCA_000188695.1	56.2 MB
Algae	Chondrus crispus[164]	GCA_000350225.2	106 MB
Algae	Kappaphycus alvarezii[164]	GCA_002205965.2	341 MB
Animal	Strongylocentrotus purpuratus[163]	GCF_000002235.4	1.01 GB
Plant	Picea abies[164]	GCA_900067695.1	13.4 GB

Table 3. Other DNA datasets

Dataset	No. of sequences	Size	Source	Date
Mitochondrion[163]	9,402	245 MB	RefSeq ftp: `ftp://ftp.ncbi.nlm.nih.gov/refseq/release/mitochondrion/mitochondrion.1.1.genomic.fna.gz`	15 March 2019
			`ftp://ftp.ncbi.nlm.nih.gov/refseq/release/mitochondrion/mitochondrion.2.1.genomic.fna.gz`	
NCBI Virus Complete Nucleotide Human[165]	36,745	482 MB	NCBI Virus: `https://www.ncbi.nlm.nih.gov/labs/virus/vssi/`	11 May 2020
Influenza[166]	700,001	1.22 GB	Influenza Virus Database: `ftp://ftp.ncbi.nih.gov/genomes/INFLUENZA/influenza.fna.gz`	27 April 2019
Helicobacter[164]	108,292	2.76 GB	NCBI Assembly: `https://www.ncbi.nlm.nih.gov/assembly`	24 April 2019

Table 4. Compression ration in percentage for all tested genomes using all tested tools. The indexes in the header are generated for each tested genome and provided in Table 5.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
bcm	***23.92***	***23.83***	***24.85***	23.56	23.71	24.06	23.40	24.72	21.71	24.05	20.71	24.80	6.54	21.21	2.36	16.28	23.68
blzpack	38.17	38.06	38.01	38.07	37.97	37.90	37.87	38.21	34.29	37.86	37.73	38.40	36.96	32.88	34.91	37.77	37.54
brotli	25.03	27.06	26.98	23.40	26.27	24.39	24.13	27.65	23.73	19.47	15.16	23.21	2.44	21.77	0.97	5.12	20.99
bsc	24.38	24.28	25.09	23.96	24.15	24.19	23.81	24.88	21.88	23.45	20.10	24.90	5.63	21.26	2.04	13.60	23.70
bzip2	26.70	26.66	27.25	26.71	26.75	26.86	26.50	27.05	24.15	26.89	24.71	27.22	11.72	23.38	5.56	26.41	26.85
freeze	29.29	28.48	28.91	28.29	28.24	28.32	27.76	28.99	25.94	28.42	28.13	28.81	25.31	24.42	22.24	27.94	28.04
gzip	29.16	28.80	29.07	28.74	28.68	28.53	28.16	28.99	26.12	28.47	27.86	28.82	18.81	24.51	11.50	28.33	28.38
lizard	41.35	38.34	38.75	37.96	38.02	37.76	37.35	38.87	34.46	37.71	34.33	38.29	18.42	32.10	6.47	37.16	37.60
lrzip	27.33	26.01	26.24	24.40	25.33	23.66	24.54	26.39	22.65	22.10	16.75	25.27	3.58	21.17	1.41	6.72	20.26
lz4	67.79	55.73	55.84	55.56	55.55	55.15	55.32	56.06	50.09	55.12	54.76	55.78	38.49	47.25	29.46	55.08	55.15
lzb	53.39	53.39	54.59	52.94	52.93	53.50	52.58	54.84	48.63	54.10							
lzfse	30.65	30.46	30.82	30.35	30.26	30.29	30.04	30.66	27.59	30.43	30.01	30.77	27.55	26.36	24.00	29.15	30.02
lzip	27.27	26.24	26.60	24.41	25.40	23.71	24.42	26.77	22.61	20.64	15.74	24.82	2.99	21.20	1.29	6.32	21.01
lzop	48.81	48.66	49.38	48.61	48.49	48.76	48.15	49.42	44.22	48.92	48.47	49.48	45.02	42.52	40.17	48.41	48.45
LzTurbo	31.86	31.15	31.27	30.75	30.80	30.50	30.43	31.43	28.07	30.71	29.73	31.07	16.19	26.21	8.15	30.54	30.39
nakamichi	40.17	34.93	34.59	31.30	31.78	30.56	30.22	32.62	28.35								
OST_bcm	24.12	23.94	24.85	23.59	***23.71***	23.93	***23.40***	***24.72***	***21.71***	23.76	19.78	24.69	6.02	21.22	2.35	14.63	***16.64***
OST_brotli	25.23	24.73	24.92	***23.34***	24.06	***22.69***	23.52	24.81	21.72	***17.97***	***14.11***	***22.95***	***2.17***	20.17	***0.97***	***5.03***	20.27
OST_bsc	24.58	24.44	25.11	24.04	24.16	24.13	23.84	24.88	21.91	23.33	19.27	24.80	5.17	21.27	2.03	12.59	23.28
OST_bzip2	26.90	26.72	27.28	26.67	26.74	26.77	26.45	26.97	24.05	26.59	24.26	27.14	10.63	23.37	5.47	19.36	21.22
OST_lrzip	27.53	26.52	26.25	24.96	25.52	23.82	24.64	26.34	22.70	21.67	16.28	25.06	3.29	21.22	1.41	6.90	17.60
OST_lzip	27.48	26.67	26.55	24.99	25.57	23.86	24.67	26.12	22.60	20.38	15.28	24.59	2.73	***2.96***	1.29	6.15	20.67
OST_xz	27.46	26.50	26.32	24.97	25.52	23.84	24.63	26.10	22.59	20.35	15.15	24.58	2.66	21.15	1.17	6.06	20.69
pbzip2	26.70	26.66	27.24	26.71	26.78	26.89	26.50	27.06	24.15	26.89	24.72	27.24	11.90	23.38	5.78	26.42	26.86
pigz	29.16	28.84	29.11	28.77	28.71	28.56	28.20	29.02	26.15	28.51	27.87	28.85	18.81	24.52	11.49	28.36	28.42
qzip	42.75	42.98	43.54	43.20	42.98	43.23	42.43	43.84	39.08	43.44	42.64	44.07	42.10	37.60	37.76	42.35	42.47
rzip	26.87	26.68	27.27	26.74	26.80	26.88	26.51	27.07	24.16	26.90	24.78	27.23	12.17	23.38	6.05	26.41	26.86
snzip	46.57	46.59	46.61	46.64	46.48	46.40	46.31	46.67	42.48	46.33	46.21	46.87	44.21	40.90	40.55	46.20	46.09
xz	27.26	25.96	26.26	24.39	25.35	23.68	24.39	26.34	22.59	20.61	15.65	24.82	2.92	21.19	1.16	6.27	21.05
zip	29.84	28.87	29.10	28.75	28.68	28.53	28.16	28.99	26.12	28.47	27.86	28.82	18.81	24.51	11.50	28.33	28.38
zlib	29.00	28.80	29.08	28.75	28.70	28.54	28.18	29.01	26.13	28.49	27.86	28.84	18.80	24.51	11.49	28.34	28.40
zpaq	41.01	37.46	36.34	34.87	34.76	34.49	34.31	35.83	31.67	34.67	31.62	35.16	11.94	29.41	5.85	32.12	34.31
zpipe	29.00	28.80	29.08	28.75	28.70	28.54	28.18	29.01	26.13	28.49	27.86	28.84	18.80	24.51	11.49	28.34	28.40
zstd	31.18	30.27	30.87	29.91	30.01	29.69	29.50	30.82	27.43	29.75	26.17	30.33	5.70	25.84	2.55	26.21	29.88

Table 5. Indexes of the genomes that are used in Table 4 along with the their sizes

Index	Genome/DNA	Size
1	Gordonia-phage-GAL1-GCF_001884535.1-2016-11-15.	50.7 kB
2	WS1-bacterium-JGI-0000059-K21-GCA_000398605.1-2013-05-16.	522 kB
3	Astrammina-rara-GCA_000211355.2-2011-04-27.	1.71 MB
4	Nosema-ceranae-GCA_000988165.1-2015-05-05.	5.81 MB
6	Spironucleus-salmonicida-GCA_000497125.1-2013-11-19.	9.22 MB
7	Tieghemostelium-lacteum-GCA_001606155.1-2016-04-04.	23.7 MB
8	Fusarium-graminearum-PH-1-GCF_000240135.3-2008-11-21.	36.9 MB
9	Salpingoeca-rosetta-GCA_000188695.1-2011-02-10.	56.2 MB
10	Chondrus-crispus-GCA_000350225.2-2013-05-22.	106 MB
11	Mitochondrion-2019-03-15.	245 MB
12	Kappaphycus-alvarezii-GCA_002205965.2-2018-03-09.	341 MB
13	NCBI-Virus-Complete-Nucleotide-Human-2020-05-11.	482 MB
14	Strongylocentrotus-purpuratus-GCF_000002235.4-2015-03-10.	1.01 GB
15	Influenza-2019-04-27.	1.22 GB
16	Helicobacter-2019-04-24.	2.76 GB
17	Picea-abies-GCA_900067695.1-2016-11-09.	13.4 GB

Data availability

The source code of the tool is available at `https://github.com/aalokaily/OST`.

Acknowledgements (not compulsory)

Additional information

References

1. Huffman, D. A. *et al.* A method for the construction of minimum-redundancy codes. *Proc. IRE* **40**, 1098–1101 (1952).
2. Knuth, D. E. Dynamic huffman coding. *J. algorithms* **6**, 163–180 (1985).
3. Vitter, J. S. Design and analysis of dynamic huffman codes. *J. ACM (JACM)* **34**, 825–845 (1987).
4. Shannon, C. E. A mathematical theory of communication. *ACM SIGMOBILE mobile computing communications review* **5**, 3–55 (2001).
5. Fano, R. M. *The transmission of information* (Massachusetts Institute of Technology, Research Laboratory of Electronics . . . , 1949).
6. Cover, T. M. *Elements of information theory* (John Wiley & Sons, 1999).
7. Ziv, J. & Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on information theory* **23**, 337–343 (1977).
8. Welch, T. A. A technique for high-performance data compression. *Computer* 8–19 (1984).
9. Friend, R. C. Transport Layer Security (TLS) Protocol Compression Using Lempel-Ziv-Stac (LZS). RFC 3943, DOI: 10.17487/RFC3943 (2004).
10. Oberhumer, M. Lzo-a real-time data compression library. *http://www. oberhumer. com/opensource/lzo/* (2008).
11. Storer, J. A. & Szymanski, T. G. Data compression via textual substitution. *J. ACM (JACM)* **29**, 928–951 (1982).

12. Williams, R. N. An extremely fast ziv-lempel data compression algorithm. In *[1991] Proceedings. Data Compression Conference*, 362–371 (IEEE, 1991).

13. Burrows, M. & Wheeler, D. J. A block-sorting lossless data compression algorithm. *Citeseer* (1994).

14. Tunstall, B. P. *Synthesis of noiseless compression codes*. Ph.D. thesis, Georgia Institute of Technology (1967).

15. Langdon, G. G. An introduction to arithmetic coding. *IBM J. Res. Dev.* **28**, 135–149 (1984).

16. Martín, G. Range encoding: an algorithm for removing redundancy from a digitised message. In *Video and Data Recording Conference, Southampton, 1979*, 24–27 (1979).

17. Ryabko, B. Y. Data compression by means of a "book stack". *Probl. Peredachi Informatsii* **16**, 16–21 (1980).

18. Bentley, J. L., Sleator, D. D., Tarjan, R. E. & Wei, V. K. A locally adaptive data compression scheme. *Commun. ACM* **29**, 320–330 (1986).

19. Capon, J. A probabilistic model for run-length coding of pictures. *IRE Transactions on Inf. Theory* **5**, 157–163 (1959).

20. Willems, F. M., Shtarkov, Y. M. & Tjalkens, T. J. The context-tree weighting method: basic properties. *IEEE transactions on information theory* **41**, 653–664 (1995).

21. Cleary, J. & Witten, I. Data compression using adaptive coding and partial string matching. *IEEE transactions on Commun.* **32**, 396–402 (1984).

22. Mahoney, M. V. Adaptive weighing of context models for lossless data compression. Tech. Rep., Florida Tech (2005).

23. Duda, J. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540* (2013).

24. Awan, F. S. & Mukherjee, A. Lipt: A lossless text transform to improve compression. In *Proceedings International Conference on Information Technology: Coding and Computing*, 452–460 (IEEE, 2001).

25. Cormack, G. V. & Horspool, R. N. S. Data compression using dynamic markov modelling. *The Comput. J.* **30**, 541–550 (1987).

26. Gopinath, A. & Ravisankar, M. Comparison of lossless data compression techniques. In *2020 International Conference on Inventive Computation Technologies (ICICT)*, 628–633 (IEEE, 2020).

27. Kavitha, P. A survey on lossless and lossy data compression methods. *Int. J. Comput. Sci. & Eng. Technol. (IJCSET)* **7** (2016).

28. squash. https://quixdb.github.io/squash-benchmark/.

29. Uthayakumar, J., Vengattaraman, T. & Dhavachelvan, P. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *J. King Saud Univ. Inf. Sci.* (2018).

30. Kodituwakku, S. & Amarasinghe, U. Comparison of lossless data compression algorithms for text data. *Indian journal computer science engineering* **1**, 416–425 (2010).

31. Ascii code, howpublished = https://www.ascii-code.com/, note = accessed: 2020-06-30 .

32. Gage, P. A new algorithm for data compression. *C Users J.* **12**, 23–38 (1994).

33. bcm. https://github.com/encode84/bcm.

34. blzpack. https://github.com/jibsen/brieflz/tree/master/example.

35. brieflz. https://github.com/jibsen/brieflz.

36. brotli. https://github.com/google/brotli.

37. bsc. https://github.com/IlyaGrebnov/libbsc.

38. bzip2. http://www.bzip.org/.

39. cmix. https://github.com/byronknoll/cmix.

40. compress. https://github.com/vapier/ncompress.

41. freeze. https://centos.pkgs.org/7/epel-x86_64/freeze-2.5.0-16.el7.x86_64.rpm.html.

42. Deutsch, P. *et al.* Gzip file format specification version 4.3 (1996).

43. hook. http://cs.fit.edu/~mmahoney/compression/hook.zip.

44. Huffman. https://github.com/Bestoa/huffman-codec.

45. lizard. https://github.com/inikep/lizard.

46. lrzip. https://github.com/ckolivas/lrzip.

47. lz4. https://github.com/lz4/lz4.

48. lzb. https://github.com/faragon/lzb.

49. lzfse. https://developer.apple.com/documentation/compression/algorithm/lzfse.

50. lzip. https://www.nongnu.org/lzip/.

51. lzop. https://www.lzop.org/.

52. Lzturbo. https://sites.google.com/site/powturbo/.

53. Nakamichi. http://www.sanmayce.com/Nakamichi/.

54. pigz. https://zlib.net/pigz/.

55. ppm. https://github.com/rene-puschinger/ppm.

56. quicklz. https://github.com/robottwo/quicklz.

57. rans. https://github.com/jkbonfield/rans_static.

58. rzip. https://rzip.samba.org/.

59. snzip. https://github.com/kubo/snzip.

60. snappy. https://github.com/google/snappy.

61. srank. https://www.cs.auckland.ac.nz/~peter-f/FTPfiles/srank.c.

62. lzma/xz. https://tukaani.org/xz/format.html.

63. zlib. https://github.com/madler/zlib.

64. zpipe. https://github.com/skyformat99/zpipe.

65. zstd. https://github.com/facebook/zstd.

66. Grumbach, S. & Tahi, F. A new challenge for compression algorithms: genetic sequences. *Inf. Process. & Manag.* **30**, 875–886 (1994).

67. Bose, T., Mohammed, M. H., Dutta, A. & Mande, S. S. Bind–an algorithm for loss-less compression of nucleotide sequence data. *J. biosciences* **37**, 785–789 (2012).

68. Grumbach, S. & Tahi, F. Compression of dna sequences. In *[Proceedings] DCC93: Data Compression Conference*, 340–350 (IEEE, 1993).

69. Majumder, A. B. & Gupta, S. Cbstd: a cloud based symbol table driven dna compression algorithm. In *Industry Interactive Innovations in Science, Engineering and Technology*, 467–476 (Springer, 2018).

70. Rivals, E., Delahaye, J.-P., Dauchet, M. & Delgrange, O. A guaranteed compression scheme for repetitive dna sequences. In *Proceedings of Data Compression Conference-DCC'96*, 453 (IEEE, 1996).

71. Xie, X., Zhou, S. & Guan, J. Cogi: Towards compressing genomes as an image. *IEEE/ACM transactions on computational biology bioinformatics* **12**, 1275–1285 (2015).

72. Mohammed, M. H., Dutta, A., Bose, T., Chadaram, S. & Mande, S. S. Deliminate—a fast and efficient method for loss-less compression of genomic sequences: sequence analysis. *Bioinformatics* **28**, 2527–2529 (2012).

73. Rajarajeswari, P. & Apparao, A. Dnabit compress–genome compression algorithm. *Bioinformation* **5**, 350 (2011).

74. Gupta, A. & Agarwal, S. A novel approach for compressing dna sequences using semi-statistical compressor. *Int. J. Comput. Appl.* **33**, 245–251 (2011).

75. Li, P. *et al.* Dna-compact: Dna com pression based on a p attern-a ware c ontextual modeling t echnique. *PloS one* **8**, e80377 (2013).

76. Chen, X., Li, M., Ma, B. & Tromp, J. Dnacompress: fast and effective dna sequence compression. *Bioinformatics* **18**, 1696–1698 (2002).

77. Mansouri, D., Yuan, X. & Saidani, A. A new lossless dna compression algorithm based on a single-block encoding scheme. *Algorithms* **13**, 99 (2020).

78. Tan, L., Sun, J. & Xiong, W. A compression algorithm for dna sequence using extended operations. *J. Comput. Inf. Syst.* **8**, 7685–7691 (2012).

79. Pinho, A. J., Ferreira, P. J., Neves, A. J. & Bastos, C. A. On the representability of complete genomes by multiple competing finite-context (markov) models. *PloS one* **6**, e21588 (2011).

80. Behzadi, B. & Le Fessant, F. Dna compression challenge revisited: a dynamic programming approach. In *Annual Symposium on Combinatorial Pattern Matching*, 190–200 (Springer, 2005).

81. Mishra, K. N. & Aaggarwal, A. An efficient horizontal and vertical method for online dna sequence compression. *Int. J. Comput. Appl.* **3**, 39–46 (2010).

82. Manzini, G. & Rastero, M. A simple and fast dna compressor. *Software: Pract. Exp.* **34**, 1397–1411 (2004).

83. Pratas, D., Pinho, A. J. & Ferreira, P. J. Efficient compression of genomic sequences. In *2016 Data Compression Conference (DCC)*, 231–240 (IEEE, 2016).

84. Pratas, D., Hosseini, M. & Pinho, A. J. Geco2: an optimized tool for lossless compression and analysis of dna sequences. In *International Conference on Practical Applications of Computational Biology & Bioinformatics*, 137–145 (Springer, 2019).

85. Rajeswari, P. R., Apparo, A. & Kumar, V. Genbit compress tool (gbc): A java-based tool to compress dna sequences and compute compression ratio (bits/base) of genomes. *arXiv preprint arXiv:1006.1193* (2010).

86. Chen, X., Kwong, S. & Li, M. A compression algorithm for dna sequences and its applications in genome comparison. *Genome informatics* **10**, 51–61 (1999).

87. Korodi, G. & Tabus, I. An efficient normalized maximum likelihood algorithm for dna sequence compression. *ACM Transactions on Inf. Syst. (TOIS)* **23**, 3–34 (2005).

88. RAJESWARI, P. R. *et al.* Huffbit compress–algorithm to compress dna sequences using extended binary trees. *J. Theor. Appl. Inf. Technol.* **13** (2010).

89. Ouyang, J., Feng, P. & Kang, J. Fast compression of huge dna sequence data. In *2012 5th International Conference on BioMedical Engineering and Informatics*, 885–888 (IEEE, 2012).

90. Alyami, S. & Huang, C.-h. Nongreedy unbalanced huffman tree compressor for single and multifasta files. *J. Comput. Biol.* (2019).

91. Hosseini, M., Pratas, D. & Pinho, A. J. A survey on data compression methods for biological sequences. *Information* **7**, 56 (2016).

92. Habib, N., Ahmed, K., Jabin, I. & Rahman, M. M. Modified huffbit compress algorithm–an application of r. *J. integrative bioinformatics* **15** (2018).

93. Kryukov, K., Ueda, M. T., Nakagawa, S. & Imanishi, T. Nucleotide archival format (naf) enables efficient lossless reference-free compression of dna sequences. *Bioinformatics* **35**, 3826–3828 (2019).

94. Tabus, I., Korodi, G. & Rissanen, J. Dna sequence compression using the normalized maximum likelihood model for discrete regression. In *Data Compression Conference, 2003. Proceedings. DCC 2003*, 253–262 (IEEE, 2003).

95. Mansouri, D. & Yuan, X. One-bit dna compression algorithm. In *International Conference on Neural Information Processing*, 378–386 (Springer, 2018).

96. Roy, S., Mondal, S., Khatua, S. & Biswas, M. An efficient compression algorithm for forthcoming new species. *Int. J. Hybrid Inf. Tech* **8**, 323–332 (2015).

97. Zhu, Z., Zhou, J., Ji, Z. & Shi, Y.-H. Dna sequence compression using adaptive particle swarm optimization-based memetic algorithm. *IEEE Transactions on Evol. Comput.* **15**, 643–658 (2011).

98. pufferfish. https://github.com/alexholehouse/pufferfish.

99. Roy, S., Bhagot, A., Sharma, K. A. & Khatua, S. Sbvrldnacomp: An effective dna sequence compression algorithm. *Int. J. Comput. Sci. Appl* **5**, 73–85 (2015).

100. Sardaraz, M., Tahir, M., Ikram, A. A. & Bajwa, H. Seqcompress: An algorithm for biological sequence compression. *Genomics* **104**, 225–228 (2014).

101. Al-Okaily, A., Almarri, B., Al Yami, S. & Huang, C.-H. Toward a better compression for dna sequences using huffman encoding. *J. Comput. Biol.* **24**, 280–288 (2017).

102. Cao, M. D., Dix, T. I., Allison, L. & Mears, C. A simple statistical algorithm for biological sequence compression. In *2007 Data Compression Conference (DCC'07)*, 43–52 (IEEE, 2007).

103. Goel, S. *et al.* A compression algorithm for dna that uses ascii values. In *2014 IEEE International Advance Computing Conference (IACC)*, 739–743 (IEEE, 2014).

104. Keerthy, A. & Priya, S. M. Lempel-ziv-welch compression of dna sequence data with indexed multiple dictionaries. *Int. J. Appl. Eng. Res* **12**, 5610–5615 (2017).

105. Matsumoto, T., Sadakane, K. & Imai, H. Biological sequence compression algorithms. *Genome informatics* **11**, 43–52 (2000).

106. Chen, M., Li, R. & Yang, L. Optimized context weighting for the compression of the un-repetitive genome sequence fragment. *Wirel. Pers. Commun.* **103**, 921–939 (2018).

107. Chen, W.-H., Lu, Y.-W., Lai, F., Chien, Y.-H. & Hwu, W.-L. Integrating human genome database into electronic health record with sequence alignment and compression mechanism. *J. medical systems* **36**, 2587–2597 (2012).

108. Eric, P. V., Gopalakrishnan, G. & Karunakaran, M. An optimal seed based compression algorithm for dna sequences. *Adv. bioinformatics* **2016** (2016).

109. Rexline, S., Aju, R. & Trujilla, L. Higher compression from burrows-wheeler transform for dna sequence. *Int. J. Comput. Appl* **173**, 11–15 (2017).

110. Kryukov, K., Ueda, M. T., Nakagawa, S. & Imanishi, T. Sequence compression benchmark (scb) database—a comprehensive evaluation of reference-free compressors for fasta-formatted sequences. *GigaScience* **9**, giaa072 (2020).

111. Bakr, N. S., Sharawi, A. A. *et al.* Dna lossless compression algorithms. *Am. J. Bioinforma. Res.* **3**, 72–81 (2013).

112. Jahaan, A., Ravi, T. & Panneer Arokiaraj, S. A comparative study and survey on existing dna compression techniques. *Int. J. Adv. Res. Comput. Sci.* **8** (2017).

113. Lipman, D. J. & Pearson, W. R. Rapid and sensitive protein similarity searches. *Science* **227**, 1435–1441 (1985).

114. Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L. & Rice, P. M. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research* **38**, 1767–1771 (2010).

115. genbank. https://www.ncbi.nlm.nih.gov/genbank/statistics/.

116. alapy. http://alapy.com/services/alapy-compressor/.

117. Ginart, A. A. *et al.* Optimal compressed representation of high throughput sequence data via light assembly. *Nat. communications* **9**, 1–9 (2018).

118. Cox, A. J., Bauer, M. J., Jakobi, T. & Rosone, G. Large-scale compression of genomic sequence databases with the burrows–wheeler transform. *Bioinformatics* **28**, 1415–1419 (2012).

119. Wang, R., Li, J., Bai, Y., Zang, T. & Wang, Y. Bdbg: a bucket-based method for compressing genome sequencing data with dynamic de bruijn graphs. *PeerJ* **6**, e5611 (2018).

120. Deorowicz, S. & Grabowski, S. Compression of dna sequence reads in fastq format. *Bioinformatics* **27**, 860–862 (2011).

121. Roguski, Ł. & Deorowicz, S. Dsrc 2—industry-oriented compression of fastq files. *Bioinformatics* **30**, 2213–2215 (2014).

122. Roguski, Ł., Ochoa, I., Hernaez, M. & Deorowicz, S. Fastore: a space-saving solution for raw sequencing data. *Bioinformatics* **34**, 2748–2756 (2018).

123. Bonfield, J. K. & Mahoney, M. V. Compression of fastq and sam format sequencing data. *PloS one* **8**, e59190 (2013).

124. fqpack. https://sourceforge.net/projects/fqpack/.

125. Deorowicz, S. Fqsqueezer: k-mer-based compression of sequencing data. *Sci. reports* **10**, 1–9 (2020).

126. Daily, K., Rigor, P., Christley, S., Xie, X. & Baldi, P. Data structures and compression algorithms for high-throughput sequencing technologies. *BMC bioinformatics* **11**, 514 (2010).

127. Cogo, V. V., Paulo, J. & Bessani, A. Genodedup: Similarity-based deduplication and delta-encoding for genome sequencing data. *IEEE Transactions on Comput.* (2020).

128. Tembe, W., Lowey, J. & Suh, E. G-sqz: compact encoding of genomic sequence and quality data. *Bioinformatics* **26**, 2192–2194 (2010).

129. Xing, Y. *et al.* Gtz: a fast compression and cloud transmission tool optimized for fastq files. *BMC bioinformatics* **18**, 233–242 (2017).

130. Chandak, S., Tatwawadi, K. & Weissman, T. Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis. *Bioinformatics* **34**, 558–567 (2018).

131. Zhang, Y., Patel, K., Endrawis, T., Bowers, A. & Sun, Y. A fastq compressor based on integer-mapped k-mer indexing for biologist. *Gene* **579**, 75–81 (2016).

132. Grassi, E., Di Gregorio, F. & Molineris, I. Kungfq: A simple and powerful approach to compress fastq files. *IEEE/ACM transactions on computational biology bioinformatics* **9**, 1837–1842 (2012).

133. Benoit, G. *et al.* Reference-free compression of high throughput sequencing data with a probabilistic de bruijn graph. *BMC bioinformatics* **16**, 288 (2015).

134. Al Yami, S. & Huang, C.-H. Lfastqc: A lossless non-reference-based fastq compressor. *PLoS One* **14**, e0224806 (2019).

135. Nicolae, M., Pathak, S. & Rajasekaran, S. Lfqc: a lossless compression algorithm for fastq files. *Bioinformatics* **31**, 3276–3281 (2015).

136. Zhang, Y. *et al.* Light-weight reference-based compression of fastq data. *BMC bioinformatics* **16**, 188 (2015).

137. Kim, M. *et al.* Metacram: an integrated pipeline for metagenomic taxonomy identification and compression. *BMC bioinformatics* **17**, 94 (2016).

138. Patro, R. & Kingsford, C. Data-dependent bucketing improves reference-free compression of sequencing reads. *Bioinformatics* **31**, 2770–2777 (2015).

139. Liu, Y., Yu, Z., Dinger, M. E. & Li, J. Index suffix–prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression. *Bioinformatics* **35**, 2066–2074 (2019).

140. Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M. & Yorke, J. A. Reducing storage requirements for biological sequence comparison. *Bioinformatics* **20**, 3363–3369 (2004).

141. Grabowski, S., Deorowicz, S. & Roguski, Ł. Disk-based compression of data from genome sequencing. *Bioinformatics* **31**, 1389–1395 (2015).

142. Jones, D. C., Ruzzo, W. L., Peng, X. & Katze, M. G. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic acids research* **40**, e171–e171 (2012).

143. Hach, F., Numanagić, I., Alkan, C. & Sahinalp, S. C. Scalce: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics* **28**, 3051–3057 (2012).

144. Howison, M. High-throughput compression of fastq data with seqdb. *IEEE/ACM transactions on computational biology bioinformatics* **10**, 213–218 (2012).

145. Kozanitis, C., Saunders, C., Kruglyak, S., Bafna, V. & Varghese, G. Compressing genomic sequence fragments using slimgene. *J. Comput. Biol.* **18**, 401–413 (2011).

146. Jeon, Y. J., Park, S. H., Ahn, S. M. & Hwang, H. J. Solidzipper: A high speed encoding method for the next-generation sequencing data. *Evol. Bioinforma.* **7**, EBO–S6618 (2011).

147. Chandak, S., Tatwawadi, K., Ochoa, I., Hernaez, M. & Weissman, T. Spring: a next-generation compressor for fastq data. *Bioinformatics* **35**, 2674–2676 (2019).

148. Bhola, V., Bopardikar, A. S., Narayanan, R., Lee, K. & Ahn, T. No-reference compression of genomic data stored in fastq format. In *2011 IEEE International Conference on Bioinformatics and Biomedicine*, 147–150 (IEEE, 2011).

149. Li, H. *et al.* The sequence alignment/map format and samtools. *Bioinformatics* **25**, 2078–2079 (2009).

150. Fritz, M. H.-Y., Leinonen, R., Cochrane, G. & Birney, E. Efficient storage of high throughput dna sequencing data using reference-based compression. *Genome research* **21**, 734–740 (2011).

151. Hach, F., Numanagic, I. & Sahinalp, S. C. Deez: reference-based compression by local assembly. *Nat. methods* **11**, 1082–1084 (2014).

152. Campagne, F., Dorff, K. C., Chambwe, N., Robinson, J. T. & Mesirov, J. P. Compression of structured high-throughput sequencing data. *PLoS one* **8**, e79871 (2013).

153. Li, P. *et al.* Hugo: Hierarchical multi-reference genome compression for aligned reads. *J. Am. Med. Informatics Assoc.* **21**, 363–373 (2014).

154. Popitsch, N. & von Haeseler, A. Ngc: lossless and lossy compression of aligned high-throughput sequencing data. *Nucleic acids research* **41**, e27–e27 (2013).

155. samcomp. https://sourceforge.net/projects/samcomp/.

156. Sakib, M. N., Tang, J., Zheng, W. J. & Huang, C.-T. Improving transmission efficiency of large sequence alignment/map (sam) files. *PloS one* **6**, e28251 (2011).

157. Bonfield, J. K. The scramble conversion tool. *Bioinformatics* **30**, 2818 (2014).

158. Levenshtein. http://www.compression.ru/download/articles/int/levenstein_1968_on_the_redundancy_and_delay.pdf, author=Vladimir Levenshtein, year=1968.

159. Elias, P. Universal codeword sets and representations of the integers. *IEEE transactions on information theory* **21**, 194–203 (1975).

160. Salomon, D. *Data compression: the complete reference* (Springer Science & Business Media, 2004).

161. Fraenkel, A. S. & Kleinb, S. T. Robust universal complete codes for transmission and compression. *Discret. Appl. Math.* **64**, 31–55 (1996).

162. Stout, Q. Improved prefix encodings of the natural numbers (corresp.). *IEEE Transactions on Inf. Theory* **26**, 607–609 (1980).

163. O'Leary, N. A. *et al.* Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation. *Nucleic acids research* **44**, D733–D745 (2016).

164. Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. & Sayers, E. W. Genbank. *Nucleic acids research* **44**, D67–D72 (2016).

165. Brister, J. R., Ako-Adjei, D., Bao, Y. & Blinkova, O. Ncbi viral genomes resource. *Nucleic acids research* **43**, D571–D577 (2015).

166. Bao, Y. *et al.* The influenza virus resource at the national center for biotechnology information. *J. virology* **82**, 596–601 (2008).