LRCB: A Comprehensive Benchmark Evaluation of Reference-free Lossless Compression Tools for Genomics Sequencing Long Reads Data (Supplementary Materials)

Hui Sun, Yingfeng Zheng, Huidong Ma, Haonan Xie,
Cheng Zhong, Xiaoguang Liu, Gang Wang

November 1, 2023

This document provides detailed information on the acquisition of benchmark testing datasets, the algorithms tested, and additional experiments. Our achieved benchmark tests tool LRCB is available at `https://github.com/fahaihi/LRCB`.

Contents

1 Benchmark Algorithms Details

1.1 General-Purpose Compressors

1.1.1 ZPAQ

ZPAQ[1] is a command-line archiver that is free, open-source and allows for incremental journaling. It has five compression methods to choose from. The first method, LZ77, compresses by replacing duplicate strings with pointers to previous occurrences. Method 2 is similar but takes more time to find better matches using a suffix array instead of a hash table. Method 3 uses BWT or LZ77 for long matches, a 1-order context model, and arithmetic coding for literals based on the file type. Method 4 uses either LZ77, BWT or a high-order context model. Lastly, Method 5 uses a complex, high-order context mixing model with over 20-bit prediction components.

In this paper, the specific commands for performing compression and decompression using zpaq V7.15[1] are as follows.

```
# compression
zpaq a file.zpaq file.reads -method 5 -threads 16
# decompression
zpaq x file.zpaq -method 5 -threads 16
```

1.1.2 LZMA and LZMA2

The LZMA (Lempel-Ziv-Markov chain Algorithm) [2] is a commonly used compression algorithm that merges the LZ77 algorithm and Arithmetic Coding[3, 4]. It compresses and decompresses data through a dictionary, sliding windows, and dynamic encoding.

LZ77 (Lempel-Ziv-77) was proposed by Abraham Lempel and Jacob Ziv[5] in 1977, mainly based on dictionary encoding, and achieves compression by utilizing the references of previously occurring repeated data. It uses the concept of sliding windows and lookup buffer to operate Data. The basic principle of the LZ77 algorithm is to search for the longest substring matching the current position within the sliding window and express the matching result as a pointer (offset and length). In this way, when compressing data, only Data compression is achieved by storing pointers instead of duplicate data.

We perform long reads collection compression using the LZMA algorithm built into the 7-Zip application[6], the detailed compression and decompression command is as follows.

```
# compression
7zz a -m0=lzma -mx9 -mmt16 file.7z file.reads
# decompression
7zz x -y -mx9 -mmt16 file.7z
```

LZMA2 improves the multi-threading capability and performance of the LZMA algorithm and better handles incompressible data, so the compression performance is slightly improved. We also used the built-in LZMA2 algorithm in the 7-Zip application [6].

```
# compression
7zz a -m0=lzma2 -mx9 -mmt16 file.7z file.reads
# decompression
7zz x -y -mx9 -mmt16 file.7z
```

1.1.3 PPMD

PPMD is a context-based compressor, and its core idea is the Partial Matching Prediction (PPM) algorithm proposed by Cleary and Witten[7]. PPM is a statistical modeling technique that uses a set of previous symbols in the input to predict the next symbol to reduce the output data's entropy. PPM differs from a dictionary because PPM predicts the next symbol instead of trying to find the next symbol in the dictionary to encode[7, 8, 9].

We utilized the PPMD in the 7-Zip [6] to compress long reads collection.

```
# compression
7zz a -m0=ppmd -mx9 -mmt16 file.7z file.reads
```

```
# decompression
7zz x -y -mx9 -mmt16 file.7z
```

1.1.4 Pigz

Pigz[10], short for "parallel implementation of gzip[11]", is a highly efficient alternative to Gzip that takes full advantage of multiple processors and cores during the compression process. Developed by Mark Adler, Pigz incorporates the Zlib[12] and pthread libraries to achieve its functionality.

In our experiments, we used Pigz V2.7. The detailed compression and decompression commands are as follows.

```
# compression
pigz -c -11 -p 16 file.reads > file.gz
# decompression
pigz -dc -p 16 file.gz > file.de_reads
```

1.1.5 PBzip2

PBzip2[13] is a parallel implementation of the Bzip2[14] block-sorting file compression algorithm that uses pthreads and achieves near-linear speedup on SMP devices. PBzip2 utilizes the Burrows-Wheeler block sorting algorithm [15, 16]for compressing files, along with Huffman coding[17] for efficient text compression.

This manuscript uses parallel BZIP2 V1.1.13 [13] to compress sequencing long reads.

```
# compression
pbzip2 -9 -m2000 -p16 -c file.reads > file.bz2
# decompression
pbzip2 -dc -9 -p16 -m2000 file.bz2
```

1.1.6 XZ

XZ Utils [18] is free general-purpose data compression software with a high compression ratio. XZ Utils were written for POSIX-like systems, but also work on some not-so-POSIX systems. XZ Utils are the successor to LZMA Utils.

In our experiments, we used XZ V5.5.0. The compression and decompression commands are as follows.

```
# compression
xz -z9ke file.reads -T 16
# decompression
xz -dk file.reads.xz -T 16
```

1.1.7 Brotli

Brotli [19] is a general-purpose lossless compression algorithm that combines the use of a modern variant of the LZ77 algorithm, Huffman coding, and second-order context modeling to compress data with compression ratios comparable to the best general-purpose compression methods available. It is similar in speed to deflate, but has a higher compression density.

We used Brotli V1.1.0 to finish the experiments. The detailed commands are as follows.

```
# compression
brotli file.reads -o file.brotli
# decompression
brotli -d file.brotli -o file.brotli.reads
```

1.1.8 BSC

BSC [20] is a high-performance file compressor based on lossless block-ordered data compression algorithm, block-ordered data compression algorithm, high-performance file compressor.

This manuscript uses BSC V3.3.2 to compress and decompress long reads.

```
# compression
bsc e file.reads file.bsc -e2
# decompression
bsc d file.bsc file.bsc.reads
```

1.1.9 Zstd

Zstd (Zstandard) [21] is a fast compression algorithm that provides high compression ratios. It also offers a special mode for small data called dictionary compression. The reference library offers a very wide range of speed/compression tradeoffs and is backed by an extremely fast decoder.

We used Zstd V1.5.5 for our experiments. The detailed commands are as follows.

```
# compression
zstd -19 file.reads -o file.zstd
# decompression
zstd -d file.zstd -o file.zstd.reads
```

1.1.10 LZ4

LZ4 [22] is a lossless compression algorithm with a compression speed of >500 MB/s per core (>0.15 bytes/cycle). Its decoder is extremely fast, up to several GB/s (1 byte/cycle) per kernel. It also has a high-compression derivative called LZ4_HC that tailors CPU time to the compression rate.The LZ4 library is available as open source software under the BSD license.

In our work, we used LZ4 V1.9.4 to compress and decompress datasets.

```
# compression
lz4 -9 file.reads file.lz4 -BD
# decompression
lz4 -d file.lz4 file.lz4.reads
```

1.1.11 Lizard

Lizard (formerly LZ5) [23] is a lossless compression algorithm that contains 4 compression methods: fastLZ4, LIZv1, fastLZ4+Huffman and LIZv1+Huffman.

In our experiments, we used Lizard V1.0.0. The compression and decompression commands are as follows.

```
# compression
lizard -49 file.reads file.lizard -BD
# decompression
lizard -d file.lizard file.lizard.reads
```

1.1.12 Brieflz

Brieflz [24] is a small and fast open source implementation of a Lempel-Ziv style compression algorithm. The focus of the algorithm is on speed and code footprint, and the compression ratio achieved is quite good compared to similar algorithms.

We used Brieflz V1.3.0 for the experiments. And the commands are as follows.

```
# compression
blzpack -9 file.reads file.brieflz
# decompression
```

```
blzpack -d file.brieflz file.brieflz.reads
```

1.1.13 Lzop

Lzop [25] uses the LZO data compression library to provide compression services. Compared to gzip, it sacrifices a certain compression ratio to get higher compression and decompression speeds (at the cost of a certain compression ratio).

In the manuscript, we used Lzop V1.03 for experiments.

```
# compression
lzop -9 file.reads -o file.lzop
# decompression
lzop -d file.lzop -o file.lzop.reads
```

1.1.14 SnZip

SnZip [26] is one of command line tools using snappy. This supports several file formats; framing-format, old framing-format, hadoop-snappy format, raw format and obsolete three formats used by snzip, snappy-java and snappy-in-java before official framing-format was defined. The default format is framing-format.

SnZip V1.0.5 was used to finish the experiments. The detailed commands are as follows.

```
# compression
snzip -k -t snzip file.reads
# decompression
snzip -kd -t snzip file.reads.snz
```

1.1.15 Cmix

cmix[27] is a neural network based lossless compression algorithm aimed at optimizing compression ratio at the cost of high CPU/memory usage, and it uses thousands of context models followed by an NN based mixer. We used Cmix V19 to finish the experiments.

```
# compression
cmix -c file.reads file.cmix
# decompression
cmix -d file.cmix file.cmix.reads
```

1.1.16 LSTM-compressor

LSTM-compressor[28] is an LSTM-based lossless compression algorithm that uses the same LSTM module and preprocessing code as CMIX. LSTM-compress currently only supports compression of a single file.

In this manuscript, we used LSTM-compress V3. The detailed commands are as follows.

```
# compression
lstm-compress -c file.reads file.lstm
# decompression
lstm-compress -d file.lstm file.lstm.reads
```

1.1.17 NNCP

NNCP [29] is a lossless compression algorithm based on LSTM and supports multi-GPU parallel processing. NNCP is an experiment to build a practical lossless data compressor with neural networks. The latest version uses a Transformer model.

In this manuscript, we used NNCP V2021-06-01 to finish the experiments. The detailed commands are as follows.

```
# compression
nncp c file.reads file.nncp -T 16 --cuda
# decompression
nncp d file.nncp file.nncp.reads -T 16 --cuda
```

1.1.18 DZip

DZip [30] is a state-of-the-art general-purpose lossless compression algorithm based on deep learning. DZip includes two compression modes, combined mode and bootstrap mode. In this experiment, we use the combined mode with the higher compression rate.

The detailed commands of DZip are as follows.

```
# compression
sh ./compress.sh file.reads file.dzip com model
# decompression
sh ./decompress.sh file.dzip file.dzip.reads com model
```

1.2 Specialized Compressors

1.2.1 NanoSpring

In 2023, Meng et al. proposed NanoSpring [31], a novel lossless reference-free compressor explicitly designed for nanopore sequencing long reads. NanoSpring employs an approximate assembly approach and demonstrates improved compression performance on higher-quality datasets. Compared to general-purpose compressors and Enano [32], NanoSpring achieves a notable 3-6 fold improvement in compression on the human whole-genome dataset. Furthermore, NanoSpring exhibits competitive compression ratios and resource utilization compared to the state-of-the-art tool CoLoRd [33] while significantly outperforming it regarding decompression speed.

NanoSpring can be downloaded for free at `https://github.com/qm2/NanoSpring`. We use V0.2 for compression, and below are the detailed commands.

```
# compression
NanoSpring -c -i file.fastq -o file.nanospring -t 16
# decompression
NanoSpring -d -i file.nanospring -o file.nanospring.reads -t 16
```

1.2.2 CoLoRd

CoLoRd [33], proposed by Marek Kokot et al. in 2022, has received substantial subsequent support and maintenance. CoLoRd is designed to effectively compress long reads data generated by the ONT and PacBio third-generation sequencing platforms. The compression approach in CoLoRd is based on overlap graphs [34] commonly utilized by long-read assemblers [35, 36]. However, the objective of the CoLoRd algorithm is not to identify actual overlaps but rather to identify overlaps that facilitate efficient compression of read differences. This strategy allows for a trade-off in accuracy to enhance the performance in terms of compression throughput. Additionally, the CoLoRd algorithm supports complete FastQ file compression and parallel acceleration on multi-core CPUs.

CoLoRd can be downloaded for free at `https://github.com/refresh-bio/CoLoRd`. We use V1.2.0 for compression and decompression, and below are the detailed commands.

```
# compression
colord compress-ont -q org -p ratio -t 16 file.fastq file.colord
# decompression
colord decompress file.colord file.colord.fastq
```

1.2.3 FastqCLS

The FastqCLS [37] compressor was proposed by Dohyeon Lee et al. in 2022, presenting an algorithm that enables comprehensive compression of FastQ files while facilitating multi-threaded parallel processing. FastqCLS offers lossless reference-free compression for sequences of varying lengths, including long and short sequences. The fundamental principle of the FastqCLS algorithm revolves around a sequence rearrangement computation model developed by the authors. By reordering the sequencing reads and employing the ZPAQ [1] text compression algorithm with contextual modelling, FastqCLS achieves significant compression gains.

FastqCLS is available for free download at `https://github.com/Krlucete/FastqCLS` and is implemented using Python and C++. The detailed compression and decompression commands are provided below.

```
# compression
python3 cle_reads.py -t 16 -i file.fastq
# decompression
python3 cle_reads_decomp.py -t 16 -i file.zq_seq
```

1.2.4 GenoZip

Genozip is a comprehensive and versatile compression program designed specifically for genetic information [38]. Genozip is a software designed for genomic compression, providing universality (support for all standard genomic file formats), high compression ratios, speed, feature richness and extensibility. Genozip delivers high-performance compression for widely used genomic data formats in genomics research, namely FastQ, SAM/BAM/CRAM, VCF, GVF, FastA, PHYLIP and 23andMe formats.

GenoZip is a commercial compression software. The official website of GenoZip (`https://www.genozip.com`) provides detailed instructions on how to use the software. In our experiment, we used version V14 for compression and decompression, and the specific commands are as follows.

```
# compression
genozip --threads 16 --input generic file.reads --force -o file.genozip
# decompression
genounzip --threads 16 file.genozip -o file.genozip.reads
```

1.2.5 Enano

Enano [32] was proposed by Guillermo et al. in 2020. ENANO parses the input FASTQ file in the form of blocks, where the boundaries of each block are dynamically determined to ensure that each read is fully contained within a single block. Each block has three streams: read identifiers, base call sequences, and quality score sequences. For each stream, a specific context model is used to determine the probability distribution of each symbol, which is then passed to an arithmetic encoder to generate the output bitstream. These probability distributions are dynamically updated as the file is sequentially compressed. The decompressor works symmetrically to synchronize with the compressor.

The Enano compression toolkit can be obtained at `https://github.com/guilledufort/EnanoFASTQ`. We used the following commands to compress long reads data collection.

```
# compression
enano -c -t 16 file.fastq file.enano
# decompression
enano -d -t 16 file.enano file.enano.fastq)
```

1.2.6 Spring

Shubham [39] et al. developed the Spring Compression Toolkit in 2019. It utilizes a hash-based sequence reordering algorithm first used in the HARC [40] algorithm. However, the Spring compressor allows for variable-length compression of long reads data, which HARC does not support.

The Spring compressor can be found at `https://github.com/shubhamchandak94/Spring`. Below is a detailed guide on compressing a long reads dataset using the Spring compressor.

```
# compression
spring -c -l -i file.fastq --no-quality --no-ids -o file.spring -t 16
# decompression
spring -d -l -i file.spring -o file.spring.reads -t 16
```

1.2.7 GeCo series

The GeCo algorithm series comprises GeCo [41], GeCo2 [42], and GeCo3 [43], proposed by Diogo Pratas et al. in 2016, 2019, and 2020, respectively. The GeCo algorithm series is primarily designed for compressing DNA string sequences. In our benchmark tests, we preserved a 32-bit sequence length index for the GeCo algorithm series to ensure lossless recovery of the original data.

Within the GeCo algorithm series, GeCo [41] introduces novel Extended Finite-Context Models (XFCMs) that allow tolerance for substitution errors, enabling reference-free and reference-based DNA sequence compression. GeCo2 [42] is an improved version of GeCo, where the authors enhance the model blending by assigning a specific decay factor to each context model or tolerant context model. Furthermore, GeCo2 incorporates specific cache hash sizes and can exclusively run context models with reverse repetitions. Building upon the GeCo and GeCo2 algorithms, the research team developed GeCo3 [43], a novel genome sequence compressor that utilizes neural networks to blend multiple contexts and substitution-tolerant context models.

The GeCo algorithm series can be accessed from the following repositories: GeCo (`https://github.com/cobilab/geco`), GeCo2 (`https://github.com/cobilab/geco2`), GeCo3 (`https://github.com/cobilab/geco3`). Since different versions of the algorithm tend to offer varying performance improvements, such as enhanced compression ratios or reduced time overhead, all three algorithms were included in our testing. To compress a long reads collection, please utilize the following command.

```
# compression and decompression using GeCo
GeCo -l 5 file.reads
GeDe file.reads.co
# compression and decompression using GeCo2
GeCo2 -v -l 5 file.reads
GeDe2 file.reads.co
# compression and decompression using GeCo3
GeCo3 -l 5 -lr 0.03 -hs 40 file.reads
GeDe3 file.reads.co
```

1.2.8 NAF

The Nucleotide Archival Format (NAF) [44] was introduced by Kirill Kryukov et al. in 2019. The NAF compressor operates by dividing the input FastQ file into multiple data streams and processing each stream separately. In NAF, sequences are concatenated, and their lengths are stored separately. Then, the combined sequence is transformed into a 4-bit encoding, with each byte storing two nucleotides. Finally, the generated data streams are compressed using the Zstd [21] general-purpose compressor.

For our benchmark testing, we utilized NAF version V1.3.0, which can be obtained from the following repository: https://github.com/KirillKryukov/naf. Below are the detailed commands for conducting benchmark testing with NAF.

```
# use script to transform sequences
python3 fastq_to_single_fasta.py file.fastq
# compression
ennaf file.fasta --temp-dir ./ -o file.naf
# decompression
unnaf file.naf -o file.naf.fasta
```

Note: the python script file is available at `https://github.com/fahaihi/LRCB`

1.2.9 MFCompress

MFCompress [45] was proposed by Armando in 2014, specifically designed for compressing FASTA and multi-FASTA files. Compared to gzip and other tools used for multi-FASTA files, MFCompress can provide an additional average compression gain of nearly 50%, effectively doubling the available storage space. To compress data, we simply convert the reads file into a FastA file and record the sequence lengths. MFCompress uses single finite-context models for encoding the header text, as well as multiple competing finite-context models for encoding the main stream of the DNA sequences [45, 46].

The MFCompress compression toolkit can be obtained at `http://bioinformatics.ua.pt/software/mfcompress/`. Below are the detailed commands for compression and decompression using MFCompress.

```
# use script to transform sequences
python3 fastq_to_single_fasta.py file.fastq
# compression
MFCompressC file.fasta
# decompression
MFCompressD file.fasta.mfc
```

2 Datasets Acquisition

We evaluated the compression performance and robustness of SOTA long reads collection compression toolkits using a comprehensive datasets consisting of 31 groups of large-scale real datasets. The detailed link address of the benchmark datasets are as follows:
D1: `https://www.ebi.ac.uk/ena/browser/view/ERR3077524`
D2: `https://www.ebi.ac.uk/ena/browser/view/ERR11274574`
D3: `https://www.ebi.ac.uk/ena/browser/view/SRR25685106`
D4: `https://www.ebi.ac.uk/ena/browser/view/ERR3077535`
D5: `https://www.ebi.ac.uk/ena/browser/view/ERR2708436`
D6: `https://www.ebi.ac.uk/ena/browser/view/ERR2708427`
D7: `https://www.ebi.ac.uk/ena/browser/view/SRR1204468`
D8: `http://gembox.cbcb.umd.edu/mhap/raw/yeast_filtered.fastq.gz`
D9: `https://www.ebi.ac.uk/ena/browser/view/ERR11011595`
D10: `https://www.ebi.ac.uk/ena/browser/view/SRR25689478`
D11: `https://downloads.pacbcloud.com/public/dataset/MicrobialMultiplexing_48plex/48-plex_sequences/lima.bc1019--bc1019.subreadset.fastq.gz`
D12: `https://www.ebi.ac.uk/ena/browser/view/SRR25503121`
D13: `https://www.ebi.ac.uk/ena/browser/view/ERR4179766`
D14: `https://www.ebi.ac.uk/ena/browser/view/SRR25743051`
D15: `https://www.ebi.ac.uk/ena/browser/view/SRR25503117`
D16: `https://www.ebi.ac.uk/ena/browser/view/ERR4179765`
D17: `https://downloads.pacbcloud.com/public/dataset/MicrobialMultiplexing_48plex/48-plex_sequences/lima.bc1099--bc1099.subreadset.fastq.gz`
D18: `https://www.ebi.ac.uk/ena/browser/view/SRR25655962`
D19: `https://www.ebi.ac.uk/ena/browser/view/SRR25601474`
D20: `https://www.ebi.ac.uk/ena/browser/view/SRR12121586`
D21: `https://www.ebi.ac.uk/ena/browser/view/SRR25731491`
D22: `https://www.ebi.ac.uk/ena/browser/view/SRR25555001`
D23: `https://www.ebi.ac.uk/ena/browser/view/SRR12121585`
D24: `https://www.ebi.ac.uk/ena/browser/view/SRR25750558`
D25: `https://www.ebi.ac.uk/ena/browser/view/SRR25750949`
D26: `https://www.ebi.ac.uk/ena/browser/view/SRR25647249`
D27: `https://www.ebi.ac.uk/ena/browser/view/SRR23822210`
D28: `https://www.ebi.ac.uk/ena/browser/view/ERR5396170`
D29: `http://gembox.cbcb.umd.edu/mhap/raw/athal_filtered.fastq.gz`
D30: `https://www.ebi.ac.uk/ena/browser/view/SRR11292120`
D31: `https://www.ebi.ac.uk/ena/browser/view/SRR10382244`

Using SRA-Tools [47] or the Linux 'wget' command to download the benchmark datasets:

```
# Download D1 Using SRA-Tools
prefetch ERR3077524
fastq-dump ERR3077524
# Download D8 Using 'wget' Command
wget http://gembox.cbcb.umd.edu/mhap/raw/yeast_filtered.fastq.gz
gunzip yeast_filtered.fastq.gz
```

3 Additional Results

CoLoRd

Table 1: Experimental Results Obtained by Running CoLoRd Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.672	0.002	2.037	0.001	1.922
D2	0.002	0.360	0.004	1.656	0.001	1.632
D3	0.005	0.876	0.003	1.760	0.001	1.649
D4	0.011	1.599	0.003	1.868	0.002	1.665
D5	0.032	1.914	0.004	2.353	0.003	1.688
D6	0.046	1.472	0.008	2.501	0.006	1.752
D7	0.193	1.652	0.067	3.454	0.021	1.877
D8	0.199	1.216	0.025	5.073	0.027	2.034
D9	0.340	1.249	45.521	14.960	0.053	2.259
D10	0.079	0.233	0.172	10.456	0.063	2.456
D11	0.391	1.026	0.050	11.209	0.048	2.445
D12	0.198	0.445	0.259	11.211	0.076	2.862
D13	0.329	0.667	0.184	11.209	0.047	2.815
D14	0.341	0.669	0.296	11.208	0.097	3.059
D15	0.265	0.524	0.138	11.210	0.087	2.826
D16	0.402	0.715	0.214	11.211	0.109	2.975
D17	0.605	1.003	0.078	11.211	0.074	3.437
D18	0.290	0.415	0.151	11.212	0.111	3.301
D19	0.450	0.604	0.381	11.210	0.138	3.458
D20	0.112	0.147	0.243	11.210	0.105	3.432
D21	0.323	0.419	0.203	12.407	0.169	3.501
D22	0.112	0.140	0.262	11.211	0.111	3.511
D23	1.007	1.264	0.186	11.212	0.147	3.496
D24	0.533	0.579	0.222	13.435	0.157	3.782
D25	0.526	0.518	0.236	15.797	0.198	4.028
D26	0.418	0.223	0.497	21.751	0.269	5.949
D27	0.209	0.110	0.744	17.953	0.316	5.976
D28	0.982	0.469	0.850	13.026	0.181	6.416
D29	2.515	1.166	0.304	13.466	0.342	6.566
D30	0.816	0.334	0.654	28.585	0.464	7.497
D31	0.860	0.296	0.816	32.674	0.499	8.634
WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	CV (%)
0.518	0.773	52.777	32.674	3.923	8.634	9.174

NanoSpring

Table 2: Experimental Results Obtained by Running NanoSpring Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.006	1.988	0.001	0.505	0.001	0.348
D2	0.002	0.418	0.005	0.451	0.001	0.067
D3	0.005	1.030	0.003	0.795	0.001	0.115
D4	0.014	1.965	0.001	0.385	0.001	0.317
D5	0.033	1.974	0.003	0.665	0.001	0.827
D6	0.049	1.565	0.007	1.169	0.001	1.374
D7	0.226	1.930	0.072	38.487	0.004	3.987
D8	0.314	1.920	0.023	6.085	0.006	5.996
D9	**NONE**	**NONE**	**NONE**	**NONE**	**NONE**	**NONE**
D10	0.089	0.263	0.259	13.693	0.004	0.858
D11	0.672	1.764	0.061	10.862	0.011	12.500
D12	0.208	0.468	13.406	19.049	0.010	1.292
D13	0.334	0.678	0.926	17.442	0.009	1.488
D14	0.340	0.668	14.921	19.314	0.013	1.513
D15	0.345	0.682	0.236	12.834	0.008	1.325
D16	0.424	0.754	1.287	18.026	0.010	1.999
D17	1.080	1.792	0.073	16.006	0.019	20.513
D18	0.401	0.574	0.229	14.335	0.009	5.406
D19	0.435	0.584	23.296	23.713	0.017	2.039
D20	0.163	0.213	0.413	15.615	0.009	1.881
D21	0.374	0.486	0.216	9.412	0.009	5.977
D22	0.162	0.203	0.372	12.939	0.009	1.915
D23	1.329	1.669	0.752	15.340	0.023	12.930
D24	0.662	0.719	0.273	13.579	0.014	10.016
D25	0.620	0.611	0.470	14.086	0.014	9.278
D26	0.430	0.229	1.032	17.623	0.019	5.274
D27	0.465	0.246	0.793	17.155	0.017	4.220
D28	1.257	0.601	1.842	19.273	0.032	4.869
D29	4.049	1.877	0.261	19.561	0.062	20.589
D30	0.879	0.359	0.719	16.629	0.026	13.671
D31	1.041	0.358	0.969	17.835	0.029	13.672

WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
0.682	0.953	62.921	38.487	0.389	20.589	12.036

GeCo

Table 3: Experimental Results Obtained by Running GeCo Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.005	1.870	0.008	4.814	0.008	4.814
D2	0.002	0.375	0.007	4.814	0.007	4.814
D3	0.004	0.757	0.011	4.814	0.012	4.814
D4	0.013	1.824	0.018	4.814	0.018	4.814
D5	0.031	1.881	0.040	4.814	0.041	4.814
D6	0.040	1.296	0.079	4.814	0.081	4.814
D7	0.219	1.871	0.278	4.814	0.279	4.814
D8	0.285	1.745	0.398	4.814	0.403	4.814
D9	0.387	1.423	0.608	4.814	0.618	4.814
D10	0.106	0.315	0.674	4.814	0.701	4.814
D11	0.562	1.476	0.908	4.814	0.939	4.814
D12	0.155	0.350	0.629	4.814	0.648	4.814
D13	0.341	0.692	1.083	4.814	1.102	4.814
D14	0.252	0.495	0.804	4.814	0.851	4.814
D15	0.400	0.791	1.090	4.814	1.101	4.814
D16	0.401	0.712	1.260	4.814	1.255	4.814
D17	0.928	1.540	1.392	4.814	1.397	4.814
D18	0.767	1.098	1.445	4.814	1.466	4.814
D19	0.374	0.502	1.521	4.814	1.634	4.814
D20	0.160	0.210	1.561	4.814	1.601	4.814
D21	0.964	1.251	1.689	4.814	1.761	4.814
D22	0.166	0.208	1.621	4.814	1.657	4.814
D23	1.280	1.607	1.755	4.814	1.838	4.814
D24	1.195	1.299	2.041	4.814	2.083	4.814
D25	1.394	1.375	2.127	4.814	2.237	4.814
D26	2.058	1.100	3.965	4.814	4.013	4.814
D27	0.554	0.293	3.940	4.814	4.101	4.814
D28	1.012	0.484	4.887	4.814	4.399	4.814
D29	3.826	1.773	4.809	4.814	4.854	4.814
D30	3.201	1.311	5.178	4.814	5.192	4.814
D31	3.839	1.320	6.112	4.814	6.185	4.814
WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.025	1.072	51.938	4.814	52.482	4.814	10.051

MFCompress

Table 4: Experimental Results Obtained by Running MFCompress Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.911	0.004	0.502	0.003	0.501
D2	0.002	0.425	0.003	0.502	0.003	0.501
D3	0.005	0.865	0.005	0.502	0.005	0.501
D4	0.013	1.893	0.008	0.502	0.006	0.501
D5	0.032	1.943	0.019	0.502	0.014	0.501
D6	0.046	1.480	0.034	0.502	0.033	0.501
D7	0.222	1.899	0.134	0.502	0.095	0.501
D8	0.305	1.867	0.190	0.502	0.145	0.501
D9	0.439	1.615	0.305	0.502	0.249	0.501
D10	0.160	0.476	0.367	0.502	0.400	0.501
D11	0.655	1.719	0.423	0.502	0.367	0.501
D12	0.184	0.413	0.289	0.502	0.268	0.501
D13	0.467	0.949	0.535	0.502	0.561	0.501
D14	0.297	0.584	0.376	0.502	0.326	0.501
D15	0.647	1.282	0.541	0.502	0.536	0.501
D16	0.546	0.971	0.577	0.502	0.616	0.501
D17	1.054	1.748	0.618	0.502	0.521	0.501
D18	1.097	1.571	0.743	0.502	0.697	0.501
D19	0.458	0.614	0.714	0.502	0.758	0.501
D20	0.763	1.000	0.799	0.502	0.849	0.501
D21	1.300	1.688	0.829	0.502	1.153	0.501
D22	0.815	1.021	0.826	0.502	0.886	0.501
D23	1.368	1.717	0.836	0.502	0.720	0.501
D24	1.448	1.574	0.966	0.502	0.895	0.501
D25	1.670	1.647	1.059	0.502	0.959	0.501
D26	3.020	1.613	1.994	0.502	1.874	0.501
D27	1.984	1.053	2.426	0.502	2.106	0.501
D28	1.828	0.874	2.190	0.502	2.528	0.501
D29	3.903	1.809	2.330	0.502	1.934	0.501
D30	3.786	1.550	2.518	0.502	2.293	0.501
D31	4.552	1.566	3.044	0.502	2.763	0.501

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.360	1.333	25.702	0.502	24.563	0.501	8.788

GeCo3

Table 5: Experimental Results Obtained by Running GeCo3 Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.005	1.853	0.015	0.502	0.015	0.502
D2	0.002	0.371	0.021	0.502	0.021	0.502
D3	0.004	0.764	0.027	0.502	0.027	0.502
D4	0.013	1.817	0.037	0.502	0.037	0.502
D5	0.031	1.865	0.087	0.502	0.088	0.502
D6	0.040	1.301	0.162	0.502	0.164	0.502
D7	0.216	1.850	0.591	0.502	0.544	0.502
D8	0.290	1.778	0.765	0.502	0.759	0.502
D9	0.427	1.571	1.306	0.502	1.263	0.502
D10	0.170	0.506	1.433	0.502	1.384	0.502
D11	0.609	1.597	1.541	0.502	1.556	0.502
D12	0.158	0.356	1.702	0.502	1.728	0.502
D13	0.443	0.901	2.008	0.502	2.016	0.502
D14	0.255	0.502	1.972	0.502	1.998	0.502
D15	0.672	1.330	2.080	0.502	2.103	0.502
D16	0.522	0.928	2.292	0.502	2.314	0.502
D17	1.013	1.680	2.470	0.502	2.497	0.502
D18	1.140	1.633	2.993	0.502	2.918	0.502
D19	0.402	0.540	3.094	0.502	3.166	0.502
D20	1.013	1.327	3.275	0.502	3.205	0.502
D21	1.348	1.750	3.194	0.502	3.214	0.502
D22	1.087	1.363	3.392	0.502	3.461	0.502
D23	1.342	1.685	3.413	0.502	3.626	0.502
D24	1.448	1.574	4.089	0.502	4.122	0.502
D25	1.691	1.668	4.527	0.502	4.538	0.502
D26	3.116	1.664	8.334	0.502	8.243	0.502
D27	2.414	1.281	8.168	0.502	8.160	0.502
D28	2.193	1.049	9.042	0.502	9.053	0.502
D29	3.827	1.774	9.268	0.502	9.316	0.502
D30	3.786	1.550	10.429	0.502	10.214	0.502
D31	4.554	1.566	16.864	0.502	15.778	0.502
WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.408	1.335	108.591	0.502	107.528	0.502	8.599

FastqCLS

Table 6: Experimental Results Obtained by Running FastQCLS Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.916	0.003	0.065	0.017	0.492
D2	0.002	0.360	0.002	0.066	0.023	0.790
D3	0.004	0.821	0.002	0.067	0.029	0.812
D4	0.013	1.892	0.003	0.068	0.041	0.838
D5	0.032	1.926	0.005	0.079	0.049	1.685
D6	0.045	1.448	0.010	0.095	0.051	3.247
D7	0.218	1.866	0.036	0.165	0.075	11.989
D8	0.307	1.880	0.052	0.165	0.134	12.862
D9	0.464	1.705	0.095	0.166	0.207	12.813
D10	0.277	0.823	0.112	0.166	0.227	12.831
D11	0.654	1.717	0.123	0.166	0.274	12.858
D12	0.163	0.367	0.173	0.167	0.294	12.826
D13	0.519	1.055	0.171	0.166	0.327	12.855
D14	0.278	0.546	0.186	0.167	0.356	12.883
D15	0.736	1.456	0.171	0.166	0.368	12.861
D16	0.616	1.096	0.192	0.166	0.388	12.866
D17	1.082	1.795	0.203	0.165	0.398	12.847
D18	1.131	1.620	0.248	0.167	0.467	12.691
D19	0.433	0.581	0.279	0.176	0.502	12.795
D20	1.120	1.468	0.273	0.177	0.531	12.823
D21	1.336	1.735	0.277	0.178	0.527	12.795
D22	1.191	1.493	0.286	0.182	0.540	12.844
D23	1.411	1.771	0.651	35.338	0.544	12.788
D24	1.555	1.690	0.754	35.272	0.611	12.779
D25	1.735	1.711	0.809	35.358	0.674	12.705
D26	3.012	1.609	1.556	35.137	1.265	12.801
D27	2.813	1.493	1.586	35.175	1.282	12.757
D28	2.643	1.264	1.767	56.717	1.438	12.860
D29	3.978	1.844	1.841	35.399	1.464	12.949
D30	3.875	1.587	1.977	35.194	1.586	12.798
D31	4.647	1.598	2.351	35.123	1.936	12.969
WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.493	1.424	16.194	56.717	16.625	12.969	8.432

XZ

Table 7: Experimental Results Obtained by Running XZ Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.949	0.006	0.256	0.000	0.025
D2	0.002	0.370	0.006	0.368	0.000	0.036
D3	0.005	0.877	0.011	0.449	0.000	0.045
D4	0.013	1.922	0.018	0.593	0.001	0.057
D5	0.033	1.988	0.047	0.783	0.001	0.063
D6	0.047	1.526	0.063	1.483	0.001	0.063
D7	0.235	2.007	0.073	4.247	0.004	0.064
D8	0.319	1.955	0.081	5.940	0.006	0.063
D9	0.467	1.718	0.081	10.026	0.008	0.063
D10	0.146	0.435	0.075	12.040	0.005	0.063
D11	0.608	1.596	0.112	13.490	0.012	0.063
D12	0.209	0.469	0.099	13.195	0.006	0.063
D13	0.429	0.871	0.149	13.522	0.011	0.063
D14	0.339	0.666	0.121	13.455	0.009	0.063
D15	0.742	1.470	0.170	13.854	0.015	0.063
D16	0.514	0.914	0.162	13.706	0.013	0.063
D17	1.009	1.674	0.183	14.320	0.020	0.063
D18	1.243	1.780	0.184	14.767	0.023	0.063
D19	0.509	0.684	0.145	14.118	0.014	0.063
D20	1.185	1.552	0.201	14.768	0.023	0.063
D21	1.460	1.896	0.238	14.965	0.026	0.063
D22	1.267	1.589	0.236	14.768	0.024	0.063
D23	1.477	1.854	0.251	14.997	0.027	0.063
D24	1.613	1.754	0.267	14.893	0.031	0.063
D25	1.850	1.825	0.274	14.928	0.033	0.063
D26	3.353	1.791	0.479	14.897	0.059	0.063
D27	2.733	1.450	0.476	14.705	0.056	0.063
D28	1.159	0.554	0.265	13.983	0.033	0.063
D29	4.236	1.964	0.573	15.080	0.074	0.063
D30	4.170	1.707	0.589	14.860	0.074	0.063
D31	5.019	1.726	0.706	14.862	0.089	0.063

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.497	1.437	6.341	15.08	0.698	0.064	9.592

GeCo2

Table 8: Experimental Results Obtained by Running GeCo2 Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.005	1.899	0.003	0.502	0.003	0.502
D2	0.002	0.438	0.003	0.502	0.003	0.502
D3	0.005	0.872	0.004	0.502	0.005	0.502
D4	0.013	1.898	0.007	0.502	0.011	0.502
D5	0.032	1.942	0.026	0.502	0.019	0.502
D6	0.045	1.463	0.031	0.502	0.048	0.502
D7	0.222	1.901	0.125	0.502	0.136	0.502
D8	0.303	1.856	0.183	0.502	0.187	0.502
D9	0.455	1.674	0.314	0.502	0.335	0.502
D10	0.202	0.600	0.398	0.502	0.407	0.502
D11	0.654	1.717	0.424	0.502	0.413	0.502
D12	0.207	0.466	0.353	0.502	0.377	0.502
D13	0.498	1.011	0.573	0.502	0.575	0.502
D14	0.312	0.612	0.455	0.502	0.481	0.502
D15	0.754	1.492	0.610	0.502	0.618	0.502
D16	0.584	1.039	0.684	0.502	0.651	0.502
D17	1.068	1.771	0.666	0.502	0.660	0.502
D18	1.189	1.703	0.791	0.502	0.804	0.502
D19	0.453	0.608	0.842	0.502	0.867	0.502
D20	1.114	1.460	0.864	0.502	0.888	0.502
D21	1.394	1.810	0.880	0.502	0.875	0.502
D22	1.194	1.497	0.887	0.502	0.918	0.502
D23	1.389	1.744	0.906	0.502	0.880	0.502
D24	1.515	1.647	1.047	0.502	1.048	0.502
D25	1.758	1.734	1.144	0.502	1.164	0.502
D26	3.250	1.736	2.114	0.502	2.152	0.502
D27	2.655	1.409	2.240	0.502	1.501	0.502
D28	2.495	1.193	1.637	0.502	1.668	0.502
D29	3.952	1.832	1.627	0.502	1.647	0.502
D30	3.964	1.623	1.772	0.502	1.817	0.502
D31	4.766	1.639	2.115	0.502	2.162	0.502

WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.499	1.429	23.725	0.502	23.320	0.502	8.418

Lzma

Table 9: Experimental Results Obtained by Running Lzma Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.948	0.004	0.276	0.000	0.025
D2	0.002	0.428	0.002	0.380	0.000	0.036
D3	0.005	0.911	0.006	0.450	0.000	0.044
D4	0.013	1.918	0.013	0.572	0.000	0.058
D5	0.033	1.986	0.034	0.665	0.001	0.065
D6	0.047	1.513	0.058	0.665	0.001	0.065
D7	0.234	2.001	0.260	0.665	0.003	0.065
D8	0.317	1.939	0.371	0.665	0.004	0.065
D9	0.458	1.685	0.578	0.665	0.006	0.065
D10	0.126	0.374	0.531	0.665	0.003	0.065
D11	0.601	1.578	0.831	0.665	0.009	0.065
D12	0.242	0.544	0.366	0.665	0.004	0.065
D13	0.436	0.885	0.865	0.664	0.007	0.065
D14	0.369	0.724	0.542	0.665	0.007	0.065
D15	0.705	1.395	1.099	0.665	0.010	0.065
D16	0.518	0.921	1.021	0.665	0.009	0.065
D17	0.998	1.655	1.341	0.665	0.014	0.065
D18	1.232	1.764	1.533	0.665	0.016	0.065
D19	0.548	0.735	1.008	0.665	0.009	0.065
D20	1.152	1.509	1.540	0.665	0.015	0.065
D21	1.448	1.880	1.713	0.665	0.016	0.065
D22	1.232	1.545	1.625	0.665	0.014	0.065
D23	1.472	1.848	1.698	0.665	0.019	0.065
D24	1.603	1.742	1.957	0.665	0.021	0.065
D25	1.839	1.814	2.176	0.665	0.024	0.065
D26	3.322	1.775	4.064	0.665	0.036	0.065
D27	2.663	1.413	3.794	0.665	0.038	0.065
D28	2.297	1.098	4.309	0.665	0.030	0.065
D29	4.211	1.952	4.856	0.665	0.048	0.065
D30	4.146	1.698	5.002	0.665	0.056	0.065
D31	4.989	1.716	6.063	0.665	0.067	0.065
WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.533	1.448	49.260	0.665	0.487	0.065	8.926

Lzma2

Table 10: Experimental Results Obtained by Running Lzma2 Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.948	0.004	0.274	0.000	0.029
D2	0.002	0.428	0.002	0.378	0.000	0.037
D3	0.005	0.911	0.006	0.448	0.000	0.048
D4	0.013	1.919	0.013	0.570	0.000	0.070
D5	0.033	1.986	0.035	0.664	0.001	0.167
D6	0.047	1.514	0.058	0.665	0.001	0.297
D7	0.234	2.004	0.081	3.448	0.001	1.175
D8	0.319	1.951	0.088	4.995	0.001	1.633
D9	0.465	1.710	0.132	7.022	0.002	2.650
D10	0.148	0.440	0.129	6.692	0.002	2.844
D11	0.606	1.590	0.172	6.956	0.002	3.665
D12	0.243	0.547	0.068	6.703	0.002	3.804
D13	0.444	0.902	0.150	6.850	0.003	4.390
D14	0.370	0.728	0.094	6.755	0.003	4.373
D15	0.734	1.453	0.187	6.926	0.003	4.740
D16	0.529	0.940	0.212	6.830	0.003	4.484
D17	1.005	1.668	0.259	7.078	0.003	4.851
D18	1.242	1.778	0.264	7.056	0.003	4.907
D19	0.553	0.742	0.172	6.790	0.003	4.380
D20	1.179	1.545	0.276	6.988	0.003	4.788
D21	1.457	1.892	0.318	7.094	0.004	4.965
D22	1.261	1.581	0.323	7.056	0.004	4.808
D23	1.477	1.854	0.333	7.153	0.004	4.958
D24	1.612	1.752	0.346	7.059	0.004	4.895
D25	1.848	1.823	0.380	7.132	0.004	4.931
D26	3.348	1.789	0.711	7.119	0.007	4.917
D27	2.723	1.445	0.676	6.971	0.007	4.741
D28	2.491	1.191	0.763	6.937	0.008	4.611
D29	4.225	1.959	0.833	7.176	0.008	5.004
D30	4.166	1.706	0.843	7.094	0.008	4.875
D31	5.012	1.724	1.016	7.098	0.010	4.883

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.555	1.465	8.944	7.176	0.104	5.004	8.836

Brotli

Table 11: Experimental Results Obtained by Running Brotli Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.006	1.959	0.019	0.217	0.000	0.018
D2	0.002	0.360	0.024	0.191	0.000	0.017
D3	0.004	0.824	0.032	0.201	0.000	0.017
D4	0.013	1.935	0.047	0.202	0.000	0.017
D5	0.035	2.124	0.114	0.198	0.001	0.017
D6	0.048	1.559	0.207	0.243	0.001	0.017
D7	0.243	2.074	0.890	0.226	0.003	0.017
D8	0.326	1.996	1.205	0.228	0.003	0.017
D9	0.519	1.907	1.880	0.231	0.005	0.017
D10	0.240	0.713	1.924	0.222	0.003	0.017
D11	0.567	1.488	2.707	0.244	0.005	0.017
D12	0.212	0.477	3.022	0.191	0.003	0.017
D13	0.363	0.738	3.242	0.254	0.004	0.017
D14	0.338	0.665	3.425	0.200	0.003	0.017
D15	0.893	1.768	3.514	0.229	0.008	0.017
D16	0.444	0.790	3.692	0.251	0.004	0.017
D17	0.969	1.607	4.247	0.238	0.010	0.019
D18	1.362	1.951	4.921	0.228	0.014	0.017
D19	0.468	0.628	4.858	0.209	0.005	0.017
D20	1.369	1.794	5.550	0.229	0.012	0.018
D21	1.596	2.073	5.474	0.226	0.016	0.017
D22	1.466	1.838	5.274	0.228	0.013	0.018
D23	1.564	1.963	5.698	0.226	0.020	0.019
D24	1.724	1.874	6.571	0.228	0.020	0.017
D25	1.983	1.956	7.219	0.228	0.018	0.017
D26	3.701	1.977	13.137	0.225	0.032	0.017
D27	3.092	1.641	12.148	0.229	0.029	0.017
D28	1.114	0.532	9.955	0.256	0.017	0.017
D29	4.313	1.999	15.061	0.228	0.041	0.017
D30	4.433	1.815	16.544	0.229	0.052	0.019
D31	5.345	1.838	19.752	0.228	0.051	0.018

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.594	1.512	162.353	0.256	0.393	0.019	10.462

BSC

Table 12: Experimental Results Obtained by Running BSC Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.897	0.001	0.119	0.001	0.123
D2	0.002	0.397	0.001	0.119	0.000	0.122
D3	0.004	0.825	0.001	0.128	0.001	0.132
D4	0.013	1.922	0.002	0.159	0.001	0.137
D5	0.032	1.928	0.004	0.159	0.003	0.153
D6	0.046	1.487	0.007	0.158	0.004	0.153
D7	0.226	1.932	0.028	0.159	0.019	0.153
D8	0.317	1.942	0.039	0.159	0.027	0.153
D9	0.506	1.861	0.063	0.158	0.043	0.151
D10	0.484	1.439	0.070	0.158	0.041	0.152
D11	0.712	1.868	0.089	0.159	0.059	0.153
D12	0.184	0.414	0.068	0.157	0.027	0.152
D13	0.610	1.240	0.094	0.159	0.054	0.153
D14	0.294	0.577	0.083	0.158	0.036	0.153
D15	0.964	1.908	0.118	0.159	0.080	0.153
D16	0.717	1.275	0.108	0.159	0.062	0.153
D17	1.118	1.855	0.142	0.159	0.094	0.153
D18	1.281	1.835	0.159	0.159	0.110	0.152
D19	0.587	0.787	0.128	0.158	0.061	0.153
D20	1.369	1.794	0.173	0.158	0.116	0.152
D21	1.460	1.896	0.180	0.159	0.125	0.153
D22	1.459	1.829	0.180	0.158	0.123	0.152
D23	1.450	1.820	0.184	0.159	0.124	0.153
D24	1.639	1.782	0.207	0.158	0.139	0.153
D25	1.840	1.815	0.230	0.158	0.157	0.152
D26	3.487	1.863	0.426	0.159	0.294	0.153
D27	3.387	1.797	0.423	0.159	0.279	0.153
D28	1.272	0.608	0.329	0.158	0.145	0.152
D29	4.155	1.926	0.508	0.159	0.346	0.153
D30	4.208	1.723	0.536	0.158	0.359	0.152
D31	5.072	1.744	0.642	0.158	0.430	0.152

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.6	1.548	5.223	0.159	3.36	0.153	8.994

Zstd

Table 13: Experimental Results Obtained by Running Zstd Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.006	2.078	0.005	0.113	0.000	0.010
D2	0.002	0.378	0.006	0.118	0.000	0.010
D3	0.005	0.918	0.009	0.130	0.000	0.010
D4	0.014	2.075	0.013	0.155	0.000	0.010
D5	0.035	2.117	0.031	0.224	0.000	0.010
D6	0.051	1.641	0.055	0.220	0.000	0.010
D7	0.244	2.089	0.217	0.226	0.001	0.011
D8	0.340	2.083	0.312	0.225	0.001	0.011
D9	0.536	1.970	0.524	0.225	0.002	0.011
D10	0.400	1.189	0.717	0.212	0.002	0.011
D11	0.652	1.711	0.784	0.220	0.003	0.011
D12	0.230	0.518	0.759	0.211	0.002	0.011
D13	0.480	0.975	1.013	0.214	0.003	0.011
D14	0.361	0.710	0.893	0.211	0.002	0.011
D15	0.970	1.921	1.031	0.214	0.003	0.011
D16	0.584	1.038	1.145	0.218	0.003	0.011
D17	1.087	1.803	1.209	0.219	0.004	0.011
D18	1.375	1.970	1.376	0.226	0.004	0.011
D19	0.547	0.734	1.443	0.215	0.003	0.011
D20	1.438	1.884	1.421	0.219	0.004	0.011
D21	1.590	2.064	1.536	0.226	0.004	0.011
D22	1.538	1.928	1.499	0.223	0.003	0.011
D23	1.551	1.947	1.571	0.227	0.006	0.011
D24	1.758	1.911	1.828	0.220	0.006	0.011
D25	2.001	1.974	2.001	0.226	0.006	0.011
D26	3.707	1.980	3.720	0.227	0.008	0.011
D27	3.357	1.781	3.738	0.218	0.006	0.011
D28	1.194	0.571	2.427	0.213	0.005	0.011
D29	4.444	2.060	4.167	0.228	0.009	0.011
D30	4.534	1.856	4.323	0.215	0.010	0.011
D31	5.467	1.880	5.209	0.218	0.011	0.011

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.666	1.605	44.982	0.228	0.111	0.011	9.955

Zpaq

Table 14: Experimental Results Obtained by Running Zpaq Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.916	0.017	0.498	0.017	0.490
D2	0.001	0.322	0.024	0.797	0.023	0.796
D3	0.005	0.864	0.032	0.815	0.033	0.810
D4	0.013	1.915	0.041	0.856	0.044	0.839
D5	0.032	1.946	0.050	1.696	0.051	1.656
D6	0.049	1.574	0.054	3.285	0.052	3.223
D7	0.219	1.873	0.058	12.217	0.057	12.119
D8	0.312	1.909	0.107	13.319	0.109	12.885
D9	0.495	1.820	0.159	14.314	0.161	12.856
D10	0.520	1.546	0.168	14.282	0.174	12.809
D11	0.663	1.740	0.204	14.302	0.205	12.876
D12	0.182	0.409	0.201	14.230	0.199	12.829
D13	0.641	1.302	0.224	14.180	0.226	12.840
D14	0.298	0.585	0.247	14.356	0.246	12.895
D15	0.932	1.845	0.255	14.313	0.257	12.879
D16	0.759	1.349	0.273	14.187	0.278	12.837
D17	1.067	1.770	0.279	14.274	0.282	12.927
D18	1.244	1.782	0.327	14.301	0.338	12.930
D19	0.564	0.757	0.334	14.394	0.334	12.942
D20	1.315	1.723	0.377	14.266	0.381	12.873
D21	1.431	1.858	0.378	14.267	0.383	12.865
D22	1.403	1.759	0.379	14.266	0.388	12.814
D23	1.422	1.784	0.384	14.342	0.388	12.852
D24	1.614	1.754	0.438	14.278	0.443	12.865
D25	1.801	1.776	0.486	14.347	0.488	12.857
D26	3.366	1.798	0.831	14.271	0.846	12.927
D27	3.221	1.709	0.877	14.256	0.884	12.918
D28	3.882	1.857	0.942	14.317	0.965	12.933
D29	4.065	1.884	0.981	14.309	0.996	13.052
D30	4.110	1.683	1.064	14.251	0.996	13.007
D31	4.950	1.703	1.199	14.117	1.158	12.963
WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.669	1.565	11.390	14.394	11.402	13.052	8.250

Spring

Table 15: Experimental Results Obtained by Running Spring Algorithm on Benchmark Datasets

DataSets (ID)	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.931	0.001	0.165	0.001	0.116
D2	0.002	0.391	0.001	0.257	0.000	0.169
D3	0.005	0.866	0.001	0.308	0.001	0.230
D4	0.013	1.933	0.001	0.398	0.001	0.256
D5	0.032	1.951	0.001	0.935	0.001	0.717
D6	0.049	1.563	0.002	1.327	0.001	1.002
D7	0.226	1.935	0.006	6.265	0.003	4.441
D8	0.316	1.937	0.010	6.431	0.005	4.565
D9	0.500	1.838	0.014	6.926	0.007	4.904
D10	0.367	1.091	0.016	8.817	0.009	6.244
D11	0.707	1.855	0.021	9.349	0.011	6.917
D12	0.205	0.462	0.020	1.617	0.010	1.317
D13	0.602	1.223	0.025	6.201	0.012	4.001
D14	0.317	0.623	0.022	2.131	0.012	1.673
D15	0.936	1.852	0.024	10.097	0.014	7.107
D16	0.705	1.253	0.026	6.576	0.015	4.297
D17	1.118	1.854	0.029	10.186	0.016	7.487
D18	1.270	1.819	0.032	11.190	0.018	7.860
D19	0.623	0.836	0.036	3.321	0.018	2.525
D20	1.345	1.762	0.034	10.762	0.021	7.976
D21	1.461	1.897	0.038	12.561	0.023	8.772
D22	1.436	1.801	0.036	10.866	0.021	7.959
D23	1.445	1.814	0.042	12.720	0.024	7.595
D24	1.616	1.757	0.044	11.779	0.024	8.748
D25	1.835	1.810	0.047	13.559	0.026	9.376
D26	3.467	1.852	0.083	12.181	0.046	9.248
D27	3.258	1.729	0.084	12.900	0.041	9.696
D28	3.950	1.889	0.107	7.919	0.058	5.483
D29	4.150	1.924	0.106	13.229	0.057	9.697
D30	4.165	1.705	0.107	14.660	0.049	11.200
D31	5.024	1.728	0.130	12.968	0.067	10.922

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.693	1.577	1.146	14.660	0.612	11.200	8.395

Lizard

Table 16: Experimental Results Obtained by Running Lizard Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.006	2.140	0.004	0.070	0.000	0.009
D2	0.002	0.481	0.009	0.070	0.000	0.005
D3	0.006	1.178	0.007	0.073	0.000	0.009
D4	0.015	2.149	0.009	0.073	0.000	0.009
D5	0.036	2.168	0.020	0.073	0.000	0.009
D6	0.059	1.893	0.037	0.073	0.001	0.006
D7	0.252	2.158	0.134	0.071	0.001	0.006
D8	0.353	2.162	0.191	0.071	0.001	0.009
D9	0.565	2.079	0.312	0.073	0.002	0.009
D10	0.617	1.834	0.411	0.071	0.002	0.006
D11	0.740	1.943	0.406	0.071	0.003	0.006
D12	0.306	0.688	0.795	0.071	0.002	0.006
D13	0.790	1.605	0.600	0.071	0.003	0.009
D14	0.456	0.897	0.801	0.073	0.003	0.006
D15	1.069	2.115	0.536	0.073	0.003	0.009
D16	0.935	1.662	0.676	0.070	0.001	0.009
D17	1.190	1.975	0.659	0.072	0.004	0.007
D18	1.421	2.035	0.765	0.073	0.004	0.009
D19	0.804	1.079	1.046	0.073	0.002	0.009
D20	1.515	1.984	0.841	0.074	0.004	0.009
D21	1.618	2.101	0.806	0.073	0.004	0.007
D22	1.611	2.020	0.829	0.072	0.004	0.009
D23	1.637	2.055	0.791	0.074	0.004	0.009
D24	1.863	2.025	0.943	0.071	0.006	0.006
D25	2.063	2.035	1.090	0.074	0.007	0.006
D26	3.846	2.055	1.973	0.073	0.008	0.009
D27	3.762	1.997	2.051	0.073	0.008	0.006
D28	1.415	0.676	5.046	0.072	0.008	0.006
D29	4.644	2.153	2.278	0.073	0.013	0.009
D30	4.763	1.950	2.714	0.070	0.013	0.009
D31	5.727	1.970	3.197	0.071	0.016	0.009

WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.813	1.783	29.977	0.074	0.125	0.009	8.869

PPMD

Table 17: Experimental Results Obtained by Running PPMD Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.006	2.031	0.002	0.253	0.002	0.252
D2	0.002	0.402	0.001	0.253	0.001	0.252
D3	0.005	0.913	0.002	0.253	0.002	0.252
D4	0.014	2.057	0.005	0.253	0.003	0.252
D5	0.034	2.056	0.011	0.253	0.010	0.252
D6	0.050	1.613	0.015	0.253	0.016	0.252
D7	0.243	2.074	0.073	0.253	0.076	0.252
D8	0.340	2.079	0.101	0.253	0.107	0.252
D9	0.538	1.976	0.156	0.253	0.170	0.252
D10	0.577	1.716	0.166	0.253	0.179	0.252
D11	0.731	1.917	0.209	0.253	0.226	0.252
D12	0.196	0.440	0.076	0.253	0.079	0.252
D13	0.705	1.432	0.216	0.253	0.234	0.252
D14	0.323	0.634	0.118	0.253	0.123	0.252
D15	1.010	1.999	0.285	0.253	0.311	0.252
D16	0.834	1.483	0.251	0.253	0.269	0.252
D17	1.168	1.937	0.342	0.253	0.364	0.252
D18	1.354	1.939	0.401	0.253	0.434	0.252
D19	0.641	0.860	0.238	0.253	0.246	0.252
D20	1.429	1.872	0.411	0.253	0.445	0.252
D21	1.554	2.018	0.462	0.253	0.504	0.252
D22	1.525	1.912	0.438	0.253	0.475	0.252
D23	1.551	1.946	0.456	0.253	0.499	0.252
D24	1.751	1.904	0.523	0.253	0.557	0.252
D25	1.962	1.935	0.573	0.253	0.637	0.252
D26	3.666	1.959	1.085	0.253	1.179	0.252
D27	3.540	1.879	1.004	0.253	1.079	0.252
D28	4.166	1.993	1.190	0.253	1.282	0.252
D29	4.441	2.059	1.286	0.253	1.432	0.252
D30	4.479	1.834	1.314	0.253	1.441	0.252
D31	5.397	1.856	1.595	0.253	1.730	0.252
WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.819	1.701	13.005	0.253	14.112	0.252	8.848

GenoZip

Table 18: Experimental Results Obtained by Running Genozip Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.006	1.957	0.000	0.076	0.000	0.054
D2	0.002	0.376	0.003	0.393	0.000	0.065
D3	0.005	0.929	0.004	0.456	0.000	0.081
D4	0.014	1.968	0.000	0.161	0.000	0.119
D5	0.033	1.999	0.001	0.378	0.000	0.259
D6	0.055	1.758	0.002	0.942	0.001	0.947
D7	0.229	1.957	0.002	0.813	0.001	0.787
D8	0.321	1.966	0.001	0.904	0.001	0.816
D9	0.531	1.953	0.003	0.652	0.001	0.769
D10	0.662	1.967	0.003	0.631	0.001	0.853
D11	0.643	1.686	0.076	2.963	0.001	0.738
D12	0.229	0.515	0.069	2.918	0.001	0.665
D13	0.913	1.856	0.004	0.659	0.001	0.846
D14	0.361	0.709	0.081	2.908	0.001	0.708
D15	0.996	1.972	0.004	0.669	0.001	0.766
D16	1.049	1.866	0.005	0.623	0.001	0.800
D17	1.061	1.760	0.114	3.005	0.002	0.685
D18	1.346	1.928	0.006	1.151	0.003	1.030
D19	0.587	0.787	0.122	2.852	0.002	1.314
D20	1.506	1.973	0.006	0.931	0.003	0.836
D21	1.485	1.929	0.006	0.664	0.003	0.792
D22	1.577	1.978	0.008	0.709	0.003	0.740
D23	1.547	1.942	0.006	0.875	0.003	0.761
D24	1.782	1.938	0.010	0.617	0.004	0.750
D25	1.964	1.937	0.008	0.796	0.004	0.749
D26	3.606	1.926	0.015	0.651	0.007	0.742
D27	3.722	1.975	0.014	0.709	0.009	0.739
D28	4.133	1.977	0.015	0.523	0.008	0.725
D29	4.180	1.938	0.018	0.567	0.009	0.726
D30	4.705	1.926	0.016	0.745	0.009	0.759
D31	5.111	1.758	0.134	1.707	0.055	1.848
WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.825	1.713	0.756	3.005	0.135	1.848	8.496

Enano

Table 19: Experimental Results Obtained by Running Enano Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.005	1.908	0.001	0.062	0.001	0.054
D2	0.004	0.842	0.001	0.059	0.002	0.050
D3	0.007	1.373	0.001	0.062	0.002	0.046
D4	0.013	1.937	0.002	0.062	0.003	0.052
D5	0.032	1.948	0.004	0.059	0.007	0.048
D6	0.057	1.822	0.008	0.060	0.013	0.052
D7	0.225	1.927	0.026	0.062	0.044	0.048
D8	0.319	1.951	0.033	0.060	0.065	0.050
D9	0.509	1.872	0.051	0.059	0.102	0.054
D10	0.654	1.943	0.065	0.062	0.139	0.046
D11	0.730	1.916	0.033	0.057	0.092	0.044
D12	0.308	0.693	0.088	0.060	0.167	0.048
D13	0.896	1.822	0.039	0.059	0.118	0.050
D14	0.464	0.911	0.103	0.057	0.202	0.048
D15	0.982	1.945	0.101	0.062	0.203	0.048
D16	1.033	1.837	0.123	0.062	0.228	0.046
D17	1.126	1.867	0.050	0.059	0.145	0.044
D18	1.326	1.900	0.115	0.059	0.249	0.050
D19	1.296	1.740	0.155	0.060	0.298	0.046
D20	1.478	1.937	0.125	0.059	0.260	0.050
D21	1.469	1.908	0.156	0.062	0.303	0.052
D22	1.545	1.938	0.127	0.062	0.265	0.050
D23	1.499	1.882	0.171	0.059	0.316	0.056
D24	1.728	1.878	0.159	0.062	0.338	0.052
D25	1.891	1.866	0.175	0.064	0.375	0.052
D26	3.553	1.898	0.303	0.059	0.644	0.046
D27	3.657	1.941	0.327	0.064	0.694	0.050
D28	4.098	1.960	0.170	0.055	0.498	0.044
D29	4.164	1.930	0.418	0.060	0.821	0.050
D30	4.503	1.844	0.460	0.062	0.911	0.050
D31	5.396	1.856	0.519	0.059	1.066	0.050
WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.850	1.774	4.109	0.064	8.571	0.056	5.957

NAF

Table 20: Experimental Results Obtained by Running NAF Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.006	2.015	0.000	0.013	0.000	0.003
D2	0.003	0.587	0.000	0.012	0.000	0.003
D3	0.008	1.478	0.000	0.013	0.000	0.003
D4	0.014	1.996	0.000	0.009	0.000	0.003
D5	0.034	2.024	0.000	0.012	0.000	0.003
D6	0.060	1.944	0.001	0.012	0.000	0.004
D7	0.231	1.972	0.003	0.012	0.001	0.006
D8	0.323	1.975	0.003	0.012	0.001	0.008
D9	0.527	1.936	0.006	0.013	0.001	0.011
D10	0.671	1.996	0.005	0.013	0.001	0.013
D11	0.726	1.906	0.008	0.013	0.002	0.014
D12	0.388	0.873	0.006	0.013	0.002	0.016
D13	0.900	1.828	0.008	0.012	0.002	0.018
D14	0.554	1.089	0.008	0.012	0.002	0.018
D15	1.003	1.986	0.006	0.012	0.002	0.018
D16	1.045	1.858	0.007	0.012	0.002	0.020
D17	1.147	1.903	0.008	0.012	0.002	0.021
D18	1.360	1.947	0.009	0.013	0.003	0.024
D19	1.205	1.618	0.011	0.012	0.003	0.026
D20	1.468	1.924	0.010	0.010	0.003	0.026
D21	1.509	1.959	0.009	0.012	0.003	0.027
D22	1.548	1.941	0.010	0.010	0.003	0.028
D23	1.543	1.937	0.010	0.012	0.003	0.028
D24	1.787	1.942	0.012	0.012	0.003	0.031
D25	1.931	1.905	0.013	0.012	0.004	0.034
D26	3.677	1.964	0.024	0.012	0.006	0.061
D27	3.708	1.968	0.020	0.012	0.006	0.062
D28	4.199	2.008	0.022	0.012	0.007	0.068
D29	4.210	1.952	0.023	0.011	0.007	0.070
D30	4.518	1.850	0.028	0.012	0.008	0.079
D31	5.433	1.869	0.034	0.013	0.010	0.094

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
1.881	1.811	0.304	0.013	0.087	0.094	6.089

Pigz

Table 21: Experimental Results Obtained by Running Pigz Algorithm on Benchmark Datasets

DataSets	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.006	2.115	0.005	0.123	0.000	0.001
D2	0.002	0.416	0.008	0.103	0.000	0.001
D3	0.008	1.502	0.008	0.114	0.000	0.001
D4	0.015	2.125	0.012	0.122	0.000	0.001
D5	0.036	2.155	0.026	0.120	0.000	0.001
D6	0.063	2.043	0.050	0.125	0.000	0.001
D7	0.245	2.096	0.211	0.110	0.001	0.001
D8	0.348	2.129	0.277	0.120	0.002	0.001
D9	0.556	2.045	0.480	0.126	0.003	0.001
D10	0.716	2.129	0.541	0.122	0.003	0.001
D11	0.724	1.899	0.735	0.114	0.004	0.001
D12	0.317	0.712	0.712	0.113	0.003	0.001
D13	0.992	2.016	1.003	0.120	0.005	0.001
D14	0.485	0.953	0.797	0.121	0.005	0.001
D15	1.066	2.110	0.821	0.122	0.006	0.001
D16	1.144	2.035	1.136	0.118	0.006	0.001
D17	1.155	1.916	1.318	0.107	0.008	0.001
D18	1.423	2.039	1.275	0.115	0.008	0.001
D19	1.400	1.879	1.304	0.123	0.009	0.001
D20	1.570	2.057	1.262	0.119	0.008	0.001
D21	1.584	2.058	1.371	0.121	0.006	0.001
D22	1.641	2.058	1.328	0.115	0.008	0.001
D23	1.624	2.039	1.396	0.119	0.008	0.001
D24	1.890	2.055	1.566	0.121	0.009	0.001
D25	2.029	2.002	1.968	0.114	0.013	0.001
D26	3.816	2.039	3.431	0.115	0.022	0.001
D27	3.898	2.069	3.108	0.117	0.023	0.001
D28	4.481	2.143	3.341	0.122	0.025	0.001
D29	4.532	2.101	3.904	0.120	0.024	0.001
D30	4.759	1.949	4.926	0.115	0.027	0.001
D31	5.730	1.971	5.740	0.112	0.034	0.001
WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.985	1.899	44.060	0.126	0.270	0.001	7.492

PBzip2

Table 22: Experimental Results Obtained by Running PBzip2 Algorithm on Benchmark Datasets

DataSets (ID)	**CS** (Gigabytes)	**CR** (bits/base)	**CT** (Hours)	**CPM** (Gigabytes)	**DT** (Hours)	**DPM** (Gigabytes)
D1	0.006	2.139	0.000	0.128	0.000	0.067
D2	0.002	0.455	0.000	0.132	0.000	0.077
D3	0.007	1.308	0.000	0.138	0.000	0.085
D4	0.015	2.150	0.000	0.138	0.000	0.091
D5	0.036	2.162	0.001	0.139	0.000	0.091
D6	0.060	1.946	0.001	0.138	0.000	0.105
D7	0.251	2.149	0.002	0.139	0.001	0.107
D8	0.356	2.180	0.001	0.139	0.001	0.105
D9	0.567	2.085	0.006	0.144	0.002	0.111
D10	0.724	2.152	0.004	0.139	0.003	0.108
D11	0.790	2.073	0.007	0.141	0.003	0.106
D12	0.303	0.682	0.010	0.139	0.003	0.109
D13	0.992	2.017	0.007	0.139	0.004	0.114
D14	0.455	0.894	0.008	0.138	0.003	0.102
D15	1.089	2.156	0.007	0.141	0.004	0.106
D16	1.152	2.048	0.010	0.139	0.003	0.105
D17	1.245	2.065	0.011	0.139	0.004	0.107
D18	1.465	2.098	0.010	0.139	0.006	0.106
D19	1.196	1.605	0.012	0.139	0.006	0.113
D20	1.603	2.100	0.012	0.139	0.006	0.110
D21	1.637	2.125	0.013	0.141	0.004	0.105
D22	1.680	2.107	0.012	0.141	0.006	0.106
D23	1.657	2.080	0.012	0.139	0.006	0.105
D24	1.916	2.083	0.016	0.139	0.006	0.106
D25	2.091	2.063	0.016	0.139	0.006	0.115
D26	3.957	2.114	0.028	0.139	0.013	0.107
D27	4.006	2.126	0.028	0.139	0.013	0.110
D28	4.547	2.175	0.031	0.139	0.015	0.112
D29	4.664	2.162	0.033	0.139	0.016	0.117
D30	4.889	2.002	0.036	0.139	0.017	0.115
D31	5.881	2.023	0.045	0.139	0.020	0.117

WAvgCR (bits/base)	**AvgCR** (bits/base)	**TotalCT** (Hours)	**MaxCPM** (Gigabytes)	**TotalDT** (Hours)	**MaxDPM** (Gigabytes)	**CV** (%)
2.026	1.920	0.379	0.144	0.171	0.117	7.985

Brieflz

Table 23: Experimental Results Obtained by Running Brieflz Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.007	2.460	0.003	0.022	0.000	0.002
D2	0.002	0.481	0.004	0.022	0.000	0.002
D3	0.007	1.418	0.004	0.022	0.000	0.002
D4	0.017	2.468	0.005	0.022	0.000	0.002
D5	0.041	2.504	0.013	0.022	0.000	0.002
D6	0.069	2.207	0.023	0.022	0.001	0.002
D7	0.284	2.427	0.086	0.023	0.002	0.002
D8	0.405	2.480	0.124	0.022	0.003	0.002
D9	0.643	2.363	0.196	0.022	0.004	0.002
D10	0.796	2.367	0.253	0.022	0.005	0.002
D11	0.842	2.210	0.287	0.022	0.006	0.002
D12	0.333	0.749	0.363	0.022	0.004	0.002
D13	1.038	2.110	0.393	0.022	0.006	0.002
D14	0.504	0.990	0.413	0.022	0.005	0.002
D15	1.228	2.431	0.380	0.023	0.007	0.002
D16	1.216	2.163	0.446	0.023	0.008	0.002
D17	1.343	2.228	0.462	0.022	0.009	0.002
D18	1.643	2.353	0.532	0.022	0.009	0.002
D19	1.127	1.513	0.600	0.023	0.009	0.002
D20	1.767	2.315	0.561	0.022	0.010	0.002
D21	1.854	2.408	0.582	0.022	0.011	0.002
D22	1.872	2.348	0.586	0.022	0.012	0.002
D23	1.866	2.342	0.590	0.022	0.010	0.002
D24	2.144	2.331	0.696	0.022	0.013	0.002
D25	2.349	2.318	0.758	0.022	0.012	0.002
D26	4.431	2.367	1.424	0.022	0.024	0.002
D27	4.417	2.344	1.394	0.023	0.025	0.002
D28	1.566	0.749	1.630	0.023	0.015	0.002
D29	5.272	2.444	1.642	0.022	0.029	0.002
D30	5.424	2.221	1.821	0.022	0.029	0.002
D31	6.531	2.246	2.191	0.022	0.033	0.002

WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	%
2.100	2.076	18.462	0.023	0.301	0.002	10.191

Table 24: Experimental Results Obtained by Running Lzop Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.008	2.906	0.010	0.002	0.000	0.001
D2	0.003	0.591	0.003	0.002	0.000	0.001
D3	0.010	1.991	0.011	0.002	0.000	0.001
D4	0.020	2.905	0.024	0.002	0.000	0.001
D5	0.049	2.944	0.054	0.002	0.000	0.001
D6	0.086	2.757	0.096	0.002	0.001	0.001
D7	0.335	2.862	0.411	0.002	0.001	0.001
D8	0.477	2.923	0.599	0.002	0.002	0.001
D9	0.759	2.788	0.883	0.002	0.003	0.001
D10	0.979	2.910	1.121	0.002	0.004	0.001
D11	1.020	2.677	1.361	0.002	0.004	0.001
D12	0.433	0.975	0.479	0.002	0.003	0.001
D13	1.365	2.775	2.315	0.002	0.006	0.001
D14	0.658	1.294	0.750	0.002	0.003	0.001
D15	1.453	2.876	1.753	0.002	0.006	0.001
D16	1.575	2.801	2.615	0.002	0.005	0.001
D17	1.616	2.681	2.398	0.002	0.007	0.001
D18	1.953	2.798	2.444	0.002	0.005	0.001
D19	1.852	2.486	2.478	0.002	0.008	0.001
D20	2.135	2.798	2.463	0.002	0.006	0.001
D21	2.181	2.833	2.793	0.002	0.009	0.001
D22	2.235	2.803	3.151	0.002	0.008	0.001
D23	2.226	2.795	2.775	0.002	0.008	0.001
D24	2.584	2.809	3.223	0.002	0.009	0.001
D25	2.786	2.748	3.545	0.002	0.009	0.001
D26	5.247	2.803	6.773	0.002	0.014	0.001
D27	5.313	2.820	6.118	0.002	0.013	0.001
D28	2.275	1.088	2.534	0.002	0.012	0.001
D29	6.229	2.888	8.310	0.002	0.017	0.001
D30	6.506	2.664	8.131	0.002	0.026	0.001
D31	7.832	2.694	9.768	0.002	0.028	0.001

WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
2.559	2.538	79.389	0.002	0.217	0.001	11.263

LZ4

Table 25: Experimental Results Obtained by Running LZ4 Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.009	3.032	0.004	0.007	0.000	0.007
D2	0.003	0.597	0.001	0.006	0.000	0.005
D3	0.010	1.926	0.004	0.007	0.000	0.009
D4	0.021	3.057	0.007	0.007	0.000	0.007
D5	0.051	3.079	0.018	0.006	0.000	0.009
D6	0.087	2.789	0.030	0.007	0.000	0.009
D7	0.354	3.028	0.121	0.008	0.001	0.007
D8	0.504	3.083	0.177	0.008	0.001	0.009
D9	0.803	2.950	0.273	0.007	0.003	0.006
D10	1.028	3.055	0.364	0.008	0.002	0.009
D11	1.012	2.657	0.348	0.006	0.004	0.006
D12	0.439	0.988	0.136	0.007	0.000	0.009
D13	1.418	2.882	0.493	0.006	0.004	0.006
D14	0.664	1.305	0.210	0.006	0.003	0.009
D15	1.543	3.055	0.532	0.006	0.003	0.009
D16	1.640	2.916	0.568	0.007	0.003	0.006
D17	1.631	2.705	0.551	0.006	0.002	0.006
D18	2.058	2.948	0.739	0.006	0.004	0.009
D19	1.860	2.497	0.631	0.007	0.004	0.009
D20	2.250	2.948	0.781	0.007	0.005	0.009
D21	2.300	2.987	0.830	0.007	0.005	0.006
D22	2.356	2.955	0.812	0.007	0.004	0.009
D23	2.339	2.936	0.811	0.008	0.005	0.009
D24	2.696	2.931	0.944	0.007	0.004	0.006
D25	2.933	2.894	1.031	0.007	0.005	0.009
D26	5.531	2.955	1.996	0.006	0.008	0.009
D27	5.590	2.967	1.929	0.007	0.011	0.006
D28	2.140	1.023	0.692	0.006	0.009	0.009
D29	6.575	3.048	2.274	0.008	0.012	0.009
D30	6.824	2.794	2.419	0.006	0.016	0.009
D31	8.210	2.824	2.931	0.008	0.011	0.006
WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
2.669	2.639	22.657	0.008	0.127	0.009	12.231

SnZip

Table 26: Experimental Results Obtained by Running SnZip Algorithm on Benchmark Datasets

DataSets	**CS**	**CR**	**CT**	**CPM**	**DT**	**DPM**
	(Gigabytes)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)
D1	0.005	1.853	0.015	0.502	0.015	0.502
D2	0.002	0.371	0.021	0.502	0.021	0.502
D3	0.004	0.764	0.027	0.502	0.027	0.502
D4	0.013	1.817	0.037	0.502	0.037	0.502
D5	0.031	1.865	0.087	0.502	0.088	0.502
D6	0.040	1.301	0.162	0.502	0.164	0.502
D7	0.216	1.850	0.591	0.502	0.544	0.502
D8	0.290	1.778	0.765	0.502	0.759	0.502
D9	0.427	1.571	1.306	0.502	1.263	0.502
D10	0.170	0.506	1.433	0.502	1.384	0.502
D11	0.609	1.597	1.541	0.502	1.556	0.502
D12	0.158	0.356	1.702	0.502	1.728	0.502
D13	0.443	0.901	2.008	0.502	2.016	0.502
D14	0.255	0.502	1.972	0.502	1.998	0.502
D15	0.672	1.330	2.080	0.502	2.103	0.502
D16	0.522	0.928	2.292	0.502	2.314	0.502
D17	1.013	1.680	2.470	0.502	2.497	0.502
D18	1.140	1.633	2.993	0.502	2.918	0.502
D19	0.402	0.540	3.094	0.502	3.166	0.502
D20	1.013	1.327	3.275	0.502	3.205	0.502
D21	1.348	1.750	3.194	0.502	3.214	0.502
D22	1.087	1.363	3.392	0.502	3.461	0.502
D23	1.342	1.685	3.413	0.502	3.626	0.502
D24	1.448	1.574	4.089	0.502	4.122	0.502
D25	1.691	1.668	4.527	0.502	4.538	0.502
D26	3.116	1.664	8.334	0.502	8.243	0.502
D27	2.414	1.281	8.168	0.502	8.160	0.502
D28	2.193	1.049	9.042	0.502	9.053	0.502
D29	3.827	1.774	9.268	0.502	9.316	0.502
D30	3.786	1.550	10.429	0.502	10.214	0.502
D31	4.554	1.566	16.864	0.502	15.778	0.502

WAvgCR	**AvgCR**	**TotalCT**	**MaxCPM**	**TotalDT**	**MaxDPM**	**CV**
(bits/base)	(bits/base)	(Hours)	(Gigabytes)	(Hours)	(Gigabytes)	(%)
1.408	1.335	108.591	0.502	107.528	0.502	8.599

4 The Results of Deep Learning based Methods

Table 27: The CR (bits/base), CT (Hours), DT (Hours), CPM (GB), and DPM (GB) of Algorithms Colord, Cmix, LSTM-compressor, NNCP, and DZip on Benchmark Datasets D1-D5.

Metrics	DataSets	**Colord**	**Cmix**	**LSTM-compressor**	**NNCP**	**DZip**
CR (bits/base)	D1	**1.672**	1.833	7.754	1.884	1.869
	D2	0.360	**0.270**	0.426	0.328	0.494
	D3	0.876	**0.698**	8.221	0.799	0.755
	D4	**1.599**	1.825	8.290	1.890	1.880
	D5	1.914	**1.837**	8.019	1.838	1.840
CT (Hours)	D1	**0.002**	4.911	0.884	2.193	1.794
	D2	**0.004**	7.718	1.336	3.546	3.074
	D3	**0.003**	9.131	1.617	4.078	4.012
	D4	**0.003**	12.069	2.135	5.753	5.536
	D5	**0.004**	28.749	5.085	12.848	11.801
DT (Hours)	D1	**0.001**	4.933	0.874	2.319	1.136
	D2	**0.001**	7.385	1.345	3.287	1.735
	D3	**0.001**	9.368	1.613	4.081	2.983
	D4	**0.002**	12.269	2.102	5.379	2.924
	D5	**0.003**	29.596	5.044	12.878	6.954
CPM (GB)	D1	2.037	18.251	**0.003**	0.109	4.847
	D2	1.656	18.086	**0.003**	0.109	5.605
	D3	1.760	18.432	**0.003**	0.109	6.106
	D4	1.868	19.242	**0.003**	0.109	6.984
	D5	2.353	21.040	**0.003**	0.109	11.939
DPM (GB)	D1	1.922	18.243	**0.003**	0.109	3.715
	D2	1.632	18.085	**0.003**	0.109	3.853
	D3	1.649	18.438	**0.003**	0.109	3.820
	D4	1.665	19.237	**0.003**	0.109	4.038
	D5	1.688	21.041	**0.003**	0.109	4.837

Notes. The best results in the table are highlighted in boldface. For deep learning-based algorithms Cmix[27], LSTM-compressor[28], NNCP[29], and DZip[30], in addition to CPU memory, they also require CUDA memory. Here, we only recorded the CPU memory usage..

References

[1] ZPAQ. *ZPAQ Official Website*. http://mattmahoney.net/dc/zpaq.html.

[2] LZMA. *LZMA Official Website*. https://tukaani.org/lzma/.

[3] Glen G. Langdon Jr. "An Introduction to Arithmetic Coding". In: *IBM J. Res. Dev.* 28.2 (1984), pp. 135–149.

[4] Jorma Rissanen and Glen G. Langdon Jr. "Universal modeling and coding". In: *IEEE Trans. Inf. Theory* 27.1 (1981), pp. 12–22.

[5] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression". In: *IEEE Trans. Inf. Theory* 23.3 (1977), pp. 337–343.

[6] 7-Zip. *7-Zip Official Website*. https://www.7-zip.org/.

[7] John G. Cleary and Ian H. Witten. "Data Compression Using Adaptive Coding and Partial String Matching". In: *IEEE Trans. Commun.* 32.4 (1984), pp. 396–402.

[8] Alistair Moffat. "Implementing the PPM data compression scheme". In: *IEEE Trans. Commun.* 38.11 (1990), pp. 1917–1921.

[9] John G. Cleary and W. J. Teahan. "Unbounded Length Contexts for PPM". In: *Comput. J.* 40.2/3 (1997), pp. 67–75.

[10] PBzip2. *SRA database growth*. https://www.zlib.net/pigz/.

[11] Gzip. *Gzip Official Website*. http://www.gzip.org/.

[12] Zlib. *Zlib Official Website*. https://zlib.net/.

[13] PBzip2. *PBzip2 Official Website*. https://linux.die.net/man/1/pbzip2.

[14] Bzip2. *PBzip2 Official Website*. https://linux.die.net/man/1/pbzip2.

[15] Michael Burrows. "A block-sorting lossless data compression algorithm". In: *SRS Research Report* 124 (1994).

[16] Ziya Arnavut, David Leavitt, and Meral Abdulazizoglu. "Block Sorting Transformations". In: *Data Compression Conference, DCC 1998, Snowbird, Utah, USA, March 30 - April 1, 1998*. IEEE Computer Society, 1998, p. 524. URL: https://doi.org/10.1109/DCC.1998.672232.

[17] David A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101.

[18] XZ. *XZ Official Website*. https://github.com/tukaani-project/xz.

[19] Brotli. *Brotli Official Website*. https://github.com/google/brotli.

[20] Ilya Grebnov. *BSC Official Website*. https://github.com/IlyaGrebnov/libbsc.

[21] Zstd. *Zstd Official Website*. https://facebook.github.io/zstd/.

[22] LZ4. *LZ4 Official Website*. https://github.com/lz4/lz4.

[23] Lizard. *Lizard Official Website*. https://github.com/inikep/lizard.

[24] Brieflz. *Brieflz Official Website*. https://github.com/jibsen/brieflz.

[25] Lzop. *Lzop Official Website*. https://www.lzop.org/.

[26] Kubo Takehiro. *SnZip Official Website*. https://github.com/kubo/snzip.

[27] Cmix. *Cmix Official Website*. https://github.com/byronknoll/cmix.

[28] LSTM-compressor. *Official*. https://github.com/byronknoll/lstm-compress.

[29] NNCP. *NNCP Official Website*. https://bellard.org/nncp/.

[30] M Goyal, K Tatwawadi, S Chandak, et al. "DZip: improved general-purpose loss less compression based on novel neural network modeling". In: *2021 Data compression conference (DCC)*. IEEE. 2021, pp. 153–162.

[31] Qingxi Meng et al. "Reference-free lossless compression of nanopore sequencing reads using an approximate assembly approach". In: *Scientific Reports* 13.1 (2023), p. 2082.

[32] G Dufort y Álvarez, G Seroussi, P Smircich, et al. "ENANO: Encoder for NANOpore FASTQ files". In: *Bioinformatics* 36.16 (2020), pp. 4506–4507.

[33] Marek Kokot et al. “CoLoRd: compressing long reads”. In: *Nature methods* 19.4 (2022), pp. 441–444.

[34] Eugene W Myers. “The fragment assembly string graph”. In: *Bioinformatics* 21.suppl_2 (2005), pp. ii79–ii85.

[35] Heng Li. “Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences”. In: *Bioinformatics* 32.14 (2016), pp. 2103–2110.

[36] Sergey Koren et al. “Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation”. In: *Genome research* 27.5 (2017), pp. 722–736.

[37] D Lee and G Song. “FastqCLS: a FASTQ compressor for long-read sequencing via read reordering using a novel scoring model”. In: *Bioinformatics* 38.2 (2022), pp. 351–356.

[38] D Lan, R Tobler, Y Souilmi, et al. “Genozip: a universal extensible genomic data compressor”. In: *Bioinformatics* 37.16 (2021), pp. 2225–2230.

[39] S Chandak, K Tatwawadi, I Ochoa, et al. “SPRING: a next-generation compressor for FASTQ data”. In: *Bioinformatics* 35.15 (2019), pp. 2674–2676.

[40] Shubham Chandak, Kedar Tatwawadi, and Tsachy Weissman. “Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis”. In: *Bioinformatics* 34.4 (2018), pp. 558–567.

[41] D Pratas, AJ Pinho, and PJSG Ferreira. “Efficient compression of genomic sequences”. In: *2016 Data compression conference (DCC)*. IEEE. 2016, pp. 231–240.

[42] D Pratas, M Hosseini, and AJ Pinho. “GeCo2: An optimized tool for lossless compression and analysis of DNA sequences”. In: *Practical Applications of Computational Biology and Bioinformatics, 13th International Conference*. Springer. 2020, pp. 137–145.

[43] M Silva, D Pratas, and A Pinho. “Efficient DNA sequence compression with neural networks”. In: *GigaScience* 9.11 (2020), giaa119.

[44] K Kryukov, MT Ueda, S Nakagawa, et al. “Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences”. In: *Bioinformatics* 35.19 (2019), pp. 3826–3828.

[45] AJ Pinho and D Pratas. “MFCompress: a compression tool for FASTA and multi-FASTA data”. In: *Bioinformatics* 30.1 (2014), pp. 117–118.

[46] Armando J Pinho, Diogo Pratas, and Sara P Garcia. “GReEn: a tool for efficient compression of genome resequencing data”. In: *Nucleic acids research* 40.4 (2012), e27–e27.

[47] Li H. *Sra-Tools Official Website*. `https://github.com/ncbi/sra-tools`.