Exploring Zstandard user-provided dictionary compression for FASTA files

Michael A. Persico[1]

1 Department of Biology, Concordia University, Montreal, Quebec, Canada,

☯These authors contributed equally to this work.

¤Current Address: Dept/Program/Center, Institution Name, City, State, Country

†Deceased

¶Membership list can be found in the Acknowledgments sections

Abstract

Zstandard (Zstd) represents a universal, lossless data compression standard and implementation that is highly configurable and is aimed at coupling high compression ratios with fast compression/decompression performance. Previous studies have paired specific Zstd configurations with various file formats in bioinformatics to reduce total data volume. This paper presents a primitive "training mode" model, written in the Julia programming language, wherein a custom compression dictionary is generated from a sample FASTA set in order to explore further compression improvements and compare them to the compression performance of Xz, Zlib, Bzip2, and Lz4 universal compressors. Zstd dictionary-based compression did not yield significant compression gains ($P < 0.05$ in many compressor contrast scenarios using Tukey's HSD(honestly significant difference) test) and had performed worse than alternative compressor types. As an *E.coli* training dataset was used to generate a dictionary to compress randomized data, and with the abundance of options available for manually tuning Zstd's parameters, further compression improvements may be possible.

Introduction

The explosion of biological data has represented a significant topic of research, with a number of challenges presented over subsequent generations of technological development in regards to the management of the increasing volume and complexity of data[1, 2]. In response, emerging trends in data management have lead to the development of novel, scalable methods for the efficient transmission and storage of large amounts of data[3]. With a potentially exponential quantity of files, datasets, and other data resources to be handled, data compression represents a method for reducing overall resource size by encoding the original data into a compact form, thus helping to ease storage requirements [4]. Research into data compression in the context of biological data began to pick up near the turn of the 21st century as universal compression algorithms at the time were not considered ideal for compressing DNA or RNA sequence data well, which led to the introduction of purpose-built algorithms that addressed the unique peculiarities of genomic data[5]. At the same time, new file formats were introduced, either text-based or binary-based, for more accurate structuring and representation of biological data, complementing new software tools[6, 7].

The FASTA file format is a legacy of the original FASTA program for finding sequence similarities with a query sequence[6]. Each file can possess multiple sequences, each paired with a description line distinguished by a ">" symbol followed by arbitrary text, usually a name and/or summary description, on the same line. It is a commonly supported file format in bioinformatics and has been the target for optimized data compressors with competing claims for performance. The DELIMINATE lossless algorithm was first proposed in 2012, wherein header and sequence data are separated into DELIM-1 and DELIM-2 variants and a two-phase process is pursued involving delta encoding, progressive elimination of nucleotide characters, and 7-Zip archiving[8]. The claims of better compression/decompression performance of FASTA files were soon rivaled by the introduction of the MFCompress tool, again separating headers and sequence data but instead relying on probabilistic models to encode the data[9], which was then countered by the Nucleotide Archival Format, a novel file format noteworthy in this context for the inclusion of a Zstandard compression step[10].

IETF RFC 8878, introduced by engineers at Facebook, defines Zstandard as a lossless data compression/decompression format[11]. It is often abbreviated as "Zstd", though such can also refer to Facebook's own implementation of the algorithm written mostly in C[12]. Content is sliced and packaged into "frames" that are independent of one another defined as either compressed data Zstandard frames or Skippable frames containing custom user metadata[11]. Zstd's backbone is the use of Finite State Entropy and Huffman entropy encoding schemes that replace data with coded forms independently of the medium[13, 14], with the former compressing all symbols, though header information is first encoded by the latter[11]. Zstd, for small data compression improvements, also functions as a dictionary coder, meaning that although the algorithm is universal in the sense of being applicable to a number of both text-based and binary-based data files, it can be optimized for compacting characteristic data by "training" Zstd with a collection of sample files to build a set of common patterns that allow for substitutions when compressing/decompressing, allowing for further gains for similar data[15].

Common bioinformatics file formats often include a set structure with the express purpose of representing specific kinds of biological data composed of repeating elements, as is the case with FASTA files with either nucleotide or amino acid sequences. In this work is described a pipeline for building Zstd dictionaries via FASTA datasets along with a comparison of Zstd compression/decompression performance with that of several alternative lossless compressors for select datasets.

Materials and methods

All resources for the paper are included in the public Github repository at github.com/M-PERSIC/Persico2022.git.

The model contains two pipelines, one for Zstd custom dictionary generation and another for generating transcoded compression data using Bzip2, Xz, zLib, and Zstd compressors, with a general overview in Fig 1. Julia was chosen as there is package support for working with sequence data as well as for trivial transcoding of compressed or decompressed data via TranscodingStreams.jl wherein compression algorithm implementations are loaded as codecs[16]. A list of all direct dependencies can be found in the repository's Project.toml file. The Julia implementation operates by first unloading a tarball containing the set of FASTA training files into a Julia artifact, or life-cycled datastore. The files are plugged into Zstd's training mode, via the Zstd command line interface –train option, to generate a custom dictionary specific to that

dataset that is then plugged again into the CLI for data compression. Random FASTA data is generated using BioSequences.jl[17] and FASTX.jl[18] APIs which is then transcoded into compressed formats via the CodecBzip2.jl[19], CodecXz.jl[20], CodecZlib.jl[21], and CodecZstd.jl[22] codecs for comparative analysis. Both dictionary-based and default Zstd compression are applied to determine if the former results in a significant reduction of overall data size.

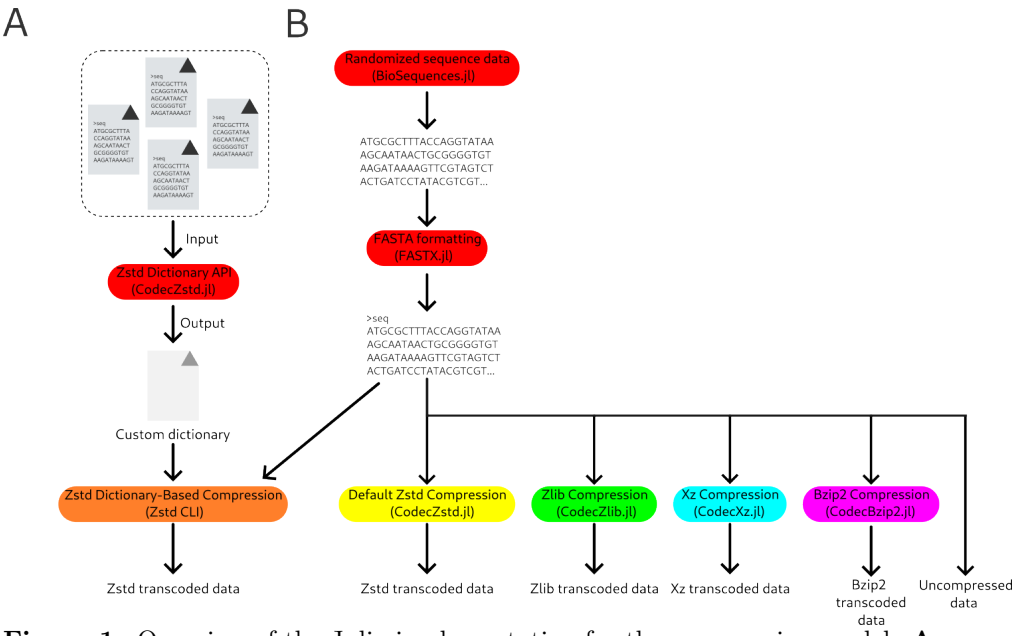

Figure 1. Overview of the Julia implementation for the compression model. **A**, Custom dictionary generation via an Julia artifact of FASTA files plugged into the Zstd Dictionary API. **B**, Transcoding randomly generated FASTA formatted data using TranscodingStreams.jl[16] compression codecs.

Results

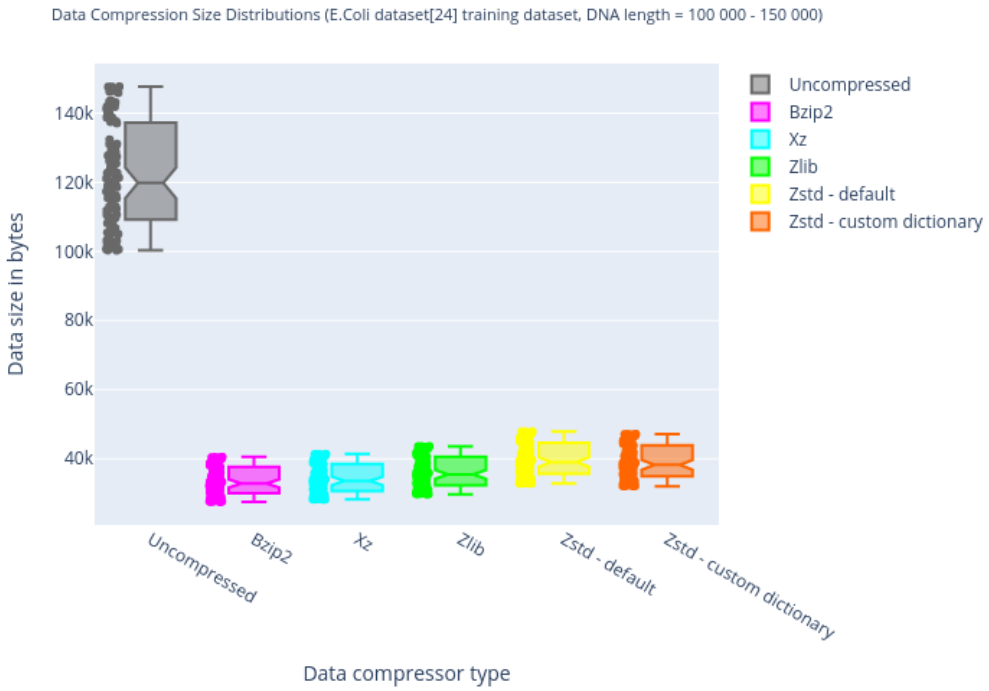

Figure 2. Average FASTA formatted data size, in bytes, following Bzip2, Xz, Zlib, default Zstd and dictionary-based Zstd compression. *Escherichia coli* MS 200-1 (NZ_ADUC00000000) artifact for Zstd dictionary generation[23]. Randomized DNA sequences within the range between 100 000 to 150 000 base pairs.

Contrasts	Adjusted p-value
Uncompressed-Bzip2	4.7907544598047e-10
Xz-Bzip2	0.983193159332035
Zlib-Bzip2	0.141907170385555
Zstd_default-Bzip2	1.05980452302923e-7
Zstd_dict-Bzip2	0.00000411265863342614
Xz-Uncompressed	4.7907544598047e-10
Zlib-Uncompressed	4.7907544598047e-10
Zstd_default-Uncompressed	4.7907544598047e-10
Zstd_dict-Uncompressed	4.7907544598047e-10
Zlib-Xz	0.493429064052453
Zstd_default-Xz	0.00000437865753477595
Zstd_dict-Xz	0.000113838138698363
Zstd_default-Zlib	0.00850774203086213
Zstd_dict-Zlib	0.0670995548100523
Zstd_dict-Zstd_default	0.984494758977982

Figure 3. Tukey's HSD test for determining statistical significance between the compression performance, based on compressed data size, of Bzip2, Xz, Zlib, Zstd with default parameters, and Zstd with a custom dictionary based on the E.coli training dataset[23].

The Tukey's HSD test paints a picture of the statistical significance of the means of the compressed set of FASTA formatted data between each compressor type. As expected, uncompressed data size and compressed data size differ significantly, with no compressed FASTA file above more than 60 kilobytes based on Fig 2. With dictionary-based compression applied, Zstd performs similarly to Zstd compression with default parameters ($P > 0.05$) and, based on the trends of the boxplot and the Tukey's test results, in almost every instance aside from default Zstd compression, dictionary-based compression performed worse than the competing compressor types.

Discussion

As stated in the Results section, Zstd dictionary-based compression results were surprising in the sense that, at least with the configuration (particular *E.coli* dataset, specific number of randomized FASTA files, completely randomized DNA, RNA, or amino acid sequences), it performed either on par, as was the case with default Zstd compression, or worse than the alternative compressors. The provided FASTA training dataset was retrieved from a set of *E.coli* MS 200-1 strain genomic scaffolds from a whole genome shotgun run[23]. Sampling a single organism may allow for additional compression/decompression improvements as the sequence data could possess repeating patterns unique to that individual's or species' genome that are recognized and added to the Zstd dictionary during training. The FASTA files themselves may bear similarity with one another in the same set, such as each file being of similar length. Simultaneously, the Zstd dictionary may not achieve comparable compression/decompression performance when attempted on more dissimilar sequence data, which might explain the observed results as randomized sequences were generated for compression performance analysis. Portability of Zstd dictionaries, in the sense of being universally applicable across all sequence data, could be weighed against generating dedicated dictionaries for specific datasets depending on the storage requirements. All other Zstd parameters were kept default, though they can be manually altered, including advanced compression options like compression job size or selecting from 22 predefined compression levels[12].

Zstd's multifarious design warrants further research in how tuning the implementation by altering available options (block size, compression level,...) might yield more significant compression performance. The development of novel data toolsets have utilized Zstd in unique ways. Aside from the Nucleotide Archival Format, FASTAFS was recently introduced as a filesystem in userspace (FUSE) toolkit wherein FASTA files are pooled into a virtualization abstraction layer that archives the set of data synced with its metadata, generating a high-level interface for accessing and storing FASTA data[24]. The zstd-seekable compression library is an embedded component that keeps the FASTAFS archive compressed for storage reduction purposes, and it represents an alternative compressed data format that deviates from Zstandard, albeit in a compatible manner, that allows for partial decompression of subranges of data[12]. This is achieved via independently compressed/decompressed Zstd frames, which FASTAFS exploits for fast random access to FASTA files by decompressing only data of interest. Further areas of research for general lossless data compression in the context of biological data could include exploring other properties like compression/decompression speed performance, meaning how fast data transcoding occurs, or exploring proof-of-concept algorithms that attempt novel strategies.

Supplementary Material

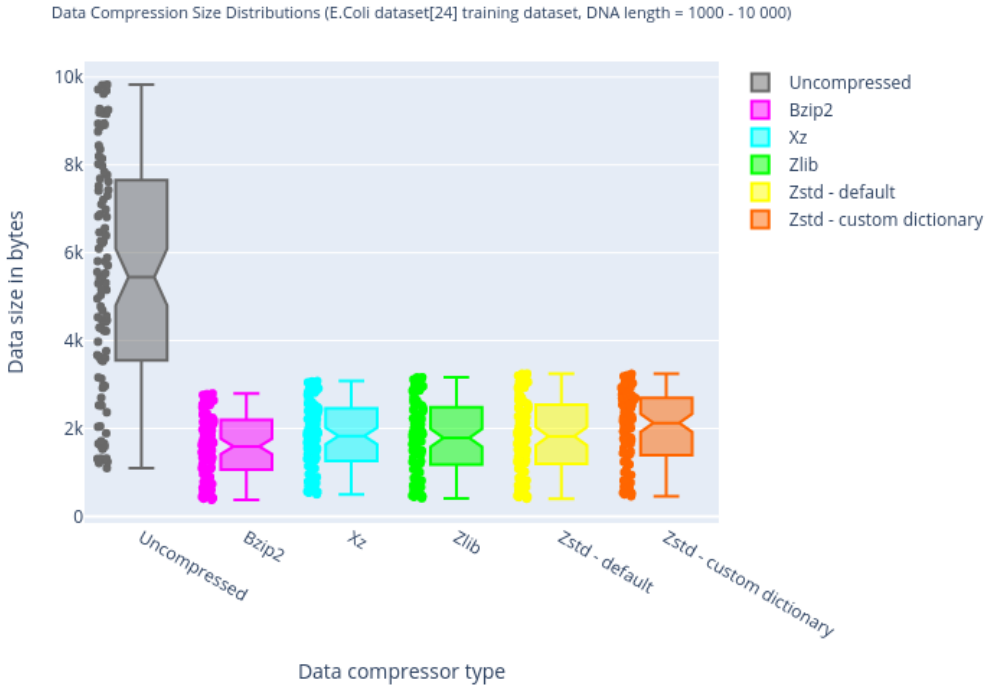

Figure 4. Average FASTA formatted data size, in bytes, following Bzip2, Xz, Zlib, default Zstd and dictionary-based Zstd compression. *Escherichia coli* MS 200-1 (NZ_ADUC00000000) artifact for Zstd dictionary generation. In this instance, randomized DNA sequences were generated within a shorter range between 100 000 to 150 000 base pairs. The discrepancy between uncompressed and compressed data is less pronounced than in Fig 2, though no compressed FASTA file exceeds more than four kilobytes in size.

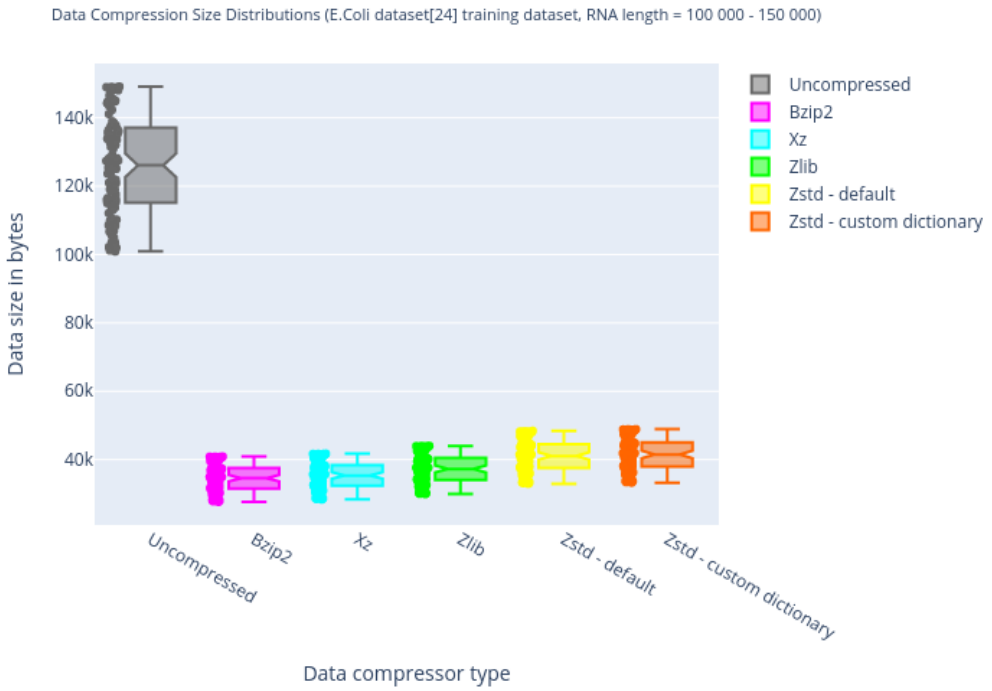

Figure 5. Average FASTA formatted data size, in bytes, following Bzip2, Xz, Zlib, default Zstd and dictionary-based Zstd compression. *Escherichia coli* MS 200-1 (NZ_ADUC00000000) artifact for Zstd dictionary generation. In this instance, randomized RNA sequences were generated within the range between 100 000 to 150 000 base pairs. The results visually remain similar to that of Fig 2, although the compression performance of dictionary-based Zstd compression appears to have slightly regressed, perhaps due to some improvements in data compression gained via dictionary use no longer being possible due the difference in sequence type (substitution of thymine with uracil).

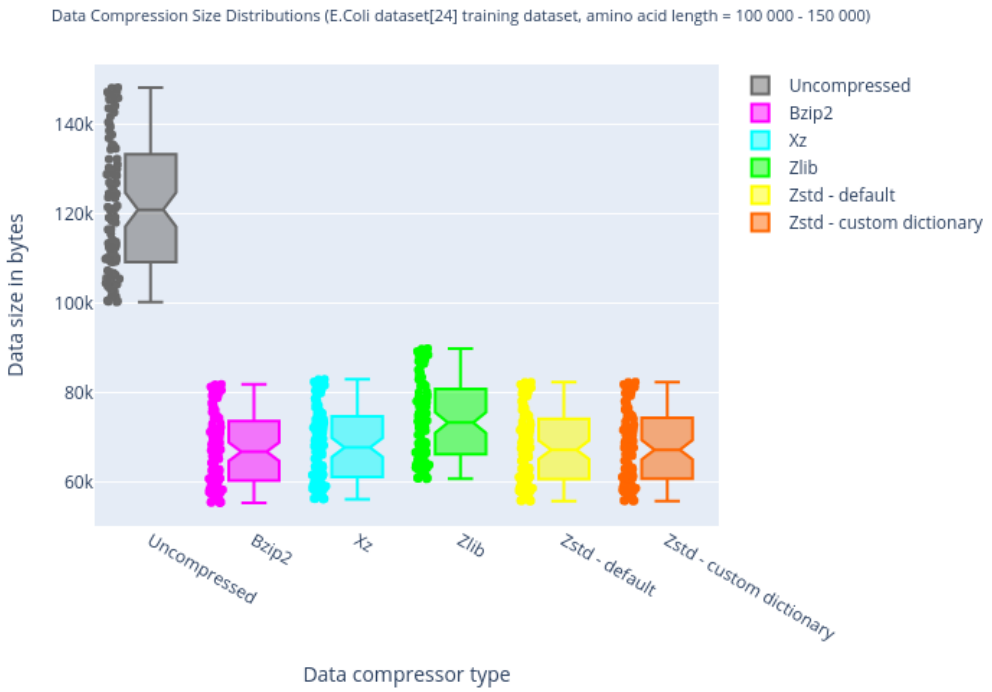

Figure 6. Average FASTA formatted data size, in bytes, following Bzip2, Xz, Zlib, default Zstd and dictionary-based Zstd compression. *Escherichia coli* MS 200-1 (NZ_ADUC00000000) artifact for Zstd dictionary generation. In this instance, randomized amino acid sequences were generated within the range between 100 000 to 150 000 codons. Zstd default and dictionary-based compression performance remain similar, it being assumed that the more complex sequence type (20 possible codons versus four possible nucleotides per position) may have erased dictionary compression improvements trained from a DNA sequence dataset.

Acknowledgments

I would like to express my sincere gratitude to Professor David Walsh for aiding me throughout the project and for teaching the bioinformatics course; My friends and family that supported me throughout my studies; The Julia community for their support before and throughout the writing of this paper; Mark Kittisopikul for developing the CodecZstd.jl #dictionary_and_parameters branch for Zstd dictionary functions; The Quarto developers for producing the Quarto publishing system[25] along with the PLOS template used for this paper; Luca Di Maio and contributors to the Distrobox tool for the ease of setting up the containerized development environments[26]; Maximiliano Sandoval and contributors to the Citations app for managing the paper's bibliography[27], and all other persons that had indirectly assisted through their programs and research.

Conflict of interest

The author declares no conflict of interest.

References

1. D'Argenio V. The high-throughput analyses era: are we ready for the data struggle? High-throughput. 2018;7(1):8.
2. Li Y, Chen L. Big biological data: challenges and opportunities. Genomics, proteomics & bioinformatics. 2014;12(5):187.
3. Sais M, Rafalia N, Abouchabaka J. Intelligent Approaches to Optimizing Big Data Storage and Management: REHDFS system and DNA Storage. Procedia Computer Science. 2022;201:746–751.
4. Jayasankar U, Thirumal V, Ponnurangam D. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. Journal of King Saud University-Computer and Information Sciences. 2021;33(2):119–140.
5. Grumbach S, Tahi F. A new challenge for compression algorithms: genetic sequences. Information processing & management. 1994;30(6):875–886.
6. Lipman DJ, Pearson WR. Rapid and sensitive protein similarity searches. Science. 1985;227(4693):1435–1441.
7. Mills L. Common file formats. Current protocols in bioinformatics. 2014;45(1):A–1B.
8. Mohammed MH, Dutta A, Bose T, Chadaram S, Mande SS. DELIMINATE—a fast and efficient method for loss-less compression of genomic sequences: Sequence analysis. Bioinformatics. 2012;28(19):2527–2529.
9. Pinho AJ, Pratas D. MFCompress: a compression tool for FASTA and multi-FASTA data. Bioinformatics. 2014;30(1):117–118.
10. Kryukov K, Ueda MT, Nakagawa S, Imanishi T. Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences. Bioinformatics. 2019;35(19):3826–3828.
11. Collet Y, Kucherawy M. RFC8478: Zstandard compression and the application/zstd media type; 2021.

12. Facebook. Facebook/ZSTD: Zstandard - fast real-time compression algorithm;. Available from: https://github.com/facebook/zstd.

13. Ezhilarasan M, Thambidurai P, Praveena K, Srinivasan S, Sumathi N. A new entropy encoding technique for multimedia data compression. International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007). 2007;doi:10.1109/iccima.2007.123.

14. Lu ZM, Guo SZ. Chapter 1 - Introduction. In: Lu ZM, Guo SZ, editors. Lossless Information Hiding in Images. Syngress; 2017. p. 1–68. Available from: https://www.sciencedirect.com/science/article/pii/B9780128120064000012.

15. Kanda S, Morita K, Fuketa M. Practical String Dictionary Compression Using String Dictionary Encoding. In: 2017 International Conference on Big Data Innovations and Applications (Innovate-Data); 2017. p. 1–8.

16. Sato K. JuliaIO/TranscodingStreams.jl: Simple, consistent interfaces for any codec.;. Available from: https://github.com/JuliaIO/TranscodingStreams.jl.

17. Ward SJ, Nissen J. Biojulia/biosequences.jl: Biological sequences for the julia language;. Available from: https://github.com/BioJulia/BioSequences.jl.

18. Ward S, Nissen J. Biojulia/fastx.jl: Parse and process fasta and FASTQ formatted files of biological sequences.;. Available from: https://github.com/BioJulia/FASTX.jl.

19. Sato K. Juliaio/codecbzip2.jl: A Bzip2 codec for transcodingstreams.jl.;. Available from: https://github.com/JuliaIO/CodecBzip2.jl.

20. Sato K. Juliaio/CodecXz.jl: An XZ codec for TranscodingStreams.jl.;. Available from: https://github.com/JuliaIO/CodecXz.jl.

21. Sato K. Juliaio/CodecZlib.jl: Zlib codecs for TranscodingStreams.jl.;. Available from: https://github.com/JuliaIO/CodecZlib.jl.

22. Sato K. Juliaio/CodecZstd.jl: A ZSTD codec for transcodingstreams.jl.;. Available from: https://github.com/JuliaIO/CodecZstd.jl.

23. Escherichia coli MS 200-1, Whole Genome Shotgun sequencing project - nucleotide - NCBI;. Available from: https://www.ncbi.nlm.nih.gov/nuccore/NZ_ADUC00000000.1.

24. Hoogstrate Y, Jenster G, van de Werken HJ. FASTAFS: File system virtualisation of random access compressed FASTA files. 2020;doi:10.1101/2020.11.11.377689.

25. Allaire JJ, Teague C, Scheidegger C, Xie Y, Dervieux C. Quarto; 2022. Available from: https://github.com/quarto-dev/quarto-cli.

26. Di Maio L. Distrobox;. Available from: https://github.com/89luca89/distrobox.

27. Sandoval M. Citations – apps for Gnome;. Available from: https://apps.gnome.org/app/org.gnome.World.Citations/.