Bioinformatics, 2023, **39(5)**, btad295
https://doi.org/10.1093/bioinformatics/btad295
Advance access publication 2 May 2023
Applications Note

OXFORD

Gene expression

BUSZ: compressed BUS files

Pétur Helgi Einarsson [1] **and Páll Melsted** [1,*]

[1]Faculty of Industrial Engineering, Mechanical Engineering, and Computer Science, University of Iceland, Reykjavík, Iceland

*Corresponding author. Faculty of Industrial Engineering, Mechanical Engineering, and Computer Science, University of Iceland, Bjargargata 1, 102 Reykjavík, Iceland. E-mail: pmelsted@hi.is

Associate Editor: Christina Kendziorski

Abstract

Summary: We describe a compression scheme for BUS files and an implementation of the algorithm in the BUStools software. Our compression algorithm yields smaller file sizes than gzip, at significantly faster compression and decompression speeds. We evaluated our algorithm on 533 BUS files from scRNA-seq experiments with a total size of 1TB. Our compression is 2.2× faster than the fastest gzip option 35% slower than the fastest zstd option and results in 1.5× smaller files than both methods. This amounts to an 8.3× reduction in the file size, resulting in a compressed size of 122GB for the dataset.

Availability and implementation: A complete description of the format is available at https://github.com/BUStools/BUSZ-format and an implementation at https://github.com/BUStools/bustools. The code to reproduce the results of this article is available at https://github.com/pmelsted/BUSZ_paper.

1 Introduction

The Barcode-Umi-Set (BUS) file format (Melsted et al. 2019) was designed to represent intermediate results from single-cell RNA-sequencing (scRNA-seq) experiments. The goal was to separate the process of alignment and downstream analysis allowing for rapid analysis and adaptation to different scRNA-seq technologies. The modular design of BUStools (Melsted et al. 2021) allows for using various sub-commands together to build pipelines tailored for various projects or technologies, which is achieved by streaming or storing intermediate BUS files.

The original BUS files are often smaller than the corresponding FASTQ files containing the original sequences, especially after sorting. The speed of analysis makes it convenient to store or archive the sorted BUS files for reproducibility or future analysis. The BUS format was designed for fast and modular processing, but still has room for compression. A simple method would be to use gzip or zstd (Collet and Kucherawy 2021) for compression, but the structured format of the data allows for tailoring the compression method to the needs of BUStools.

The first release of the Human Cell Atlas (The Tabula Sapiens Consortium 2022) contained over 500 000 cells. Advances in scRNA-seq technologies and increased sequencing throughput have already made it feasible for single labs to generated datasets of several million cells. Just the raw sequencing data can be estimated to be 2TB per 1 million cells. Thus, to contain the cost of storing and transferring BUS files we propose a compression method to efficiently compress and decompress BUS files and implement it into BUStools.

2 Methods

Technologies for scRNA-seq experiments vary in how they measure gene expression of isolated cells and capture information about individual molecules. The BUS file abstracts these technological differences by encoding the barcode of each cell and the unique molecular identifier (UMI) as short oligonucleotides encoded as integers. For each molecule, the corresponding read from the cDNA is not stored, but rather the transcript or set of transcripts it aligned to is stored as an equivalence class (EC). The EC corresponds to a set of transcripts, each set is encoded as a unique integer and the map is stored alongside of the BUS file.

A BUS file consists of a header followed by a sequence of BUS records (Melsted et al. 2019). Each BUS record occupies exactly 32 bytes and consists of six fields; barcode, UMI, EC, read count, flags, and padding. These fields are 8, 8, 4, 4, 4, 4 bytes in size, respectively. The barcodes and UMIs represent nucleotide sequences and are two-bit encoded, allowing them to be expressed as unsigned 64-bit integers. The fixed length format of BUS records allows for fast loading of BUS files and circumventing the need for parsing textual data.

The layout of records in BUS files is row-based, as shown in Fig. 1A. To compress the BUS file we use a columnar layout for the compression, where the columns correspond the fields of the BUS records, similar to the CRAM format (Fritz et al. 2011).

Our compression algorithm assumes a sorted input, which can be obtained using the `bustools sort` command. The input is sorted lexicographically by barcodes first, then by UMIs, and finally by the ECs. This can be seen as the first level of compression, as records with the same barcode, UMI,

Received: December 19, 2022. **Revised:** March 29, 2023. **Accepted:** April 28, 2023

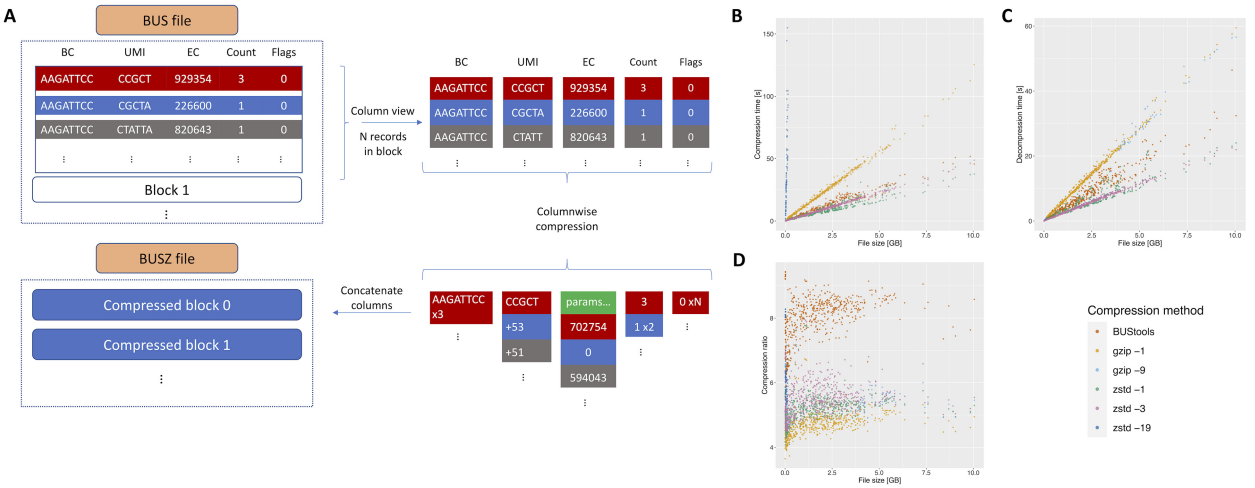

Figure 1. (A) A general outline of the compression scheme for BUS files. (B) Comparing compression time between BUStools, gzip, and zstd. The best (`gzip -9`) option is omitted due to extremely long compression time. (C) Comparing decompression time of the three methods. (D) Comparing compression ratios using the three methods.

and EC are merged. We split the input into blocks of N records (by default $N = 10\,000$). Within each block, the columns are compressed independently, each with a customized compression-decompression scheme (codec). The padding column is not included in the compressed file.

The compression scheme of each column is as follows. To simplify notation, we let RLE_0 denote run-length-encoding where zeros are compressed into runs whereas other values are left as-is, and FRLE_0 denote the Fibonacci encoding (Fraenkel and Klein 1996) of the output of RLE_0. Since Fibonacci encoding does not encode zeros, we increment each value by one before encoding. Similarly, we define RLE_1 and FRLE_1 for compressing runs of ones. This does not require incrementing the values.

When compressing the barcode column we expect that many BUS records originate from reads captured from the same cell, thus resulting in a repeat of barcodes. We take the differences of adjacent barcodes, which are non-negative since the barcode values are increasing. Since this list is expected to contain a run of zero differences, we encode the differences with FRLE_0.

The UMIs are encoded in a similar manner as the barcodes. However, they only increase within the same barcode so we must modify how the differences are computed. The difference is only taken of adjacent records from the same cell. Otherwise, the absolute value is used. We then continue with FRLE_0 on the modified differences.

The third column, containing the ECs, is not as well structured as the first two as it has higher entropy and is harder to compress (Supplementary Material). To compress this column we modified a variant of PForDelta (Zukowski et al. 2006) called NewPFD (Yan et al. 2009). This codec splits the list of ECs into sub-blocks of N_{PFD} consecutive values (by default $N_{\text{PFD}} = 512$). For each sub-block $\boldsymbol{B}$ we find two parameters, k and b, so that f (default $f = 0.9$) fraction of values fall in the interval $[k, k + 2^b - 1]$. We choose k to be the smallest value in the sub-block, whereas b is computed as the number of bits required to encode $f{\cdot}N_{\text{PFD}}$ of the values $(x - k)$, $x \in \boldsymbol{B}$. Any value requiring more than b bits is considered an exception and the remaining high bits are Fibonacci encoded. A complete description of the encoding is in the Supplementary Material.

The last two columns, count and flags, are compressed using *RLE* and Fibonacci encoding. The count values are strictly positive and are most often equal to one, so we use run-length encoding on ones (FRLE_1). The flags column contains mostly zeros so we use FRLE_0.

Each compressed block is preceded by a block header, containing the number of records contained in the block, as well as the size in bytes of the compressed block.

The compressed file starts with a header containing a fixed magic number to identify the file as a compressed BUS file, the header values of the input file, and the parameters used for the compression. Optionally, we output an indexing file, which contains the first barcode of each block and the size of the block, facilitating fast lookup of records for specific barcodes without decompressing the entire BUSZ file.

3 Results

To evaluate our compression algorithm, we measured the time it took to compress and decompress 533 BUS files collected as a part of a larger survey (Booeshaghi et al. 2022). The total size of the dataset is 1010 GB of BUS files. All experiments were performed using a single core on a Intel Xeon CPU E5-2697 2.7 GHz processor. File sizes ranged from 4MB to 10GB and were all sorted prior to compression. The full list of results and data used are given in the Supplementary Material. We compare the performance with gzip, using the fastest compression (`gzip -1`) and the best compression (`gzip -9`), and zstd using the fastest (`zstd -1`), default (`zstd -3`) and best compression (`zstd -19`). To ensure that disk access patterns and caching would not affect the results, we eliminate disk read latency by caching the input files before running each compression method. Since compressing the files using the best options (`gzip -9` and `zstd -19`) takes substantially more time than the other methods, we only used those options to compress the 100 smallest BUS files for both methods and the 50 largest for `gzip -9`. The results in Fig. 1 were acquired using a block size of

$N = 10\,000$ and a NewPFD sub-block size of $N_{\text{PFD}} = 512$. Our method performed consistently better than the two gzip methods and `zstd -19`, both in terms of speed and compression ratio. The `zstd -1` and `-3` methods were faster at both compression and decompression but resulted in larger compressed file sizes. For compression and decompression, all methods have a running time roughly linear in terms of input size up to 10GB. Fitting the data with a linear model shows a speedup of 2.2×, 55×, and 126× compared to gzip *-1*, *-9*, and `zstd -19`, respectively (Fig. 1B). The compression time of `zstd -1` and `-3` was 35% and 15% faster than BUStools. Similarly, BUStools' decompression time is 1.6× faster than both options for gzip (Fig. 1C). The zstd methods showed consistently faster decompression speeds, 1.5×, 1.6×, and 1.2× for `zstd -1`, `-3`, and `-19`. The compression ratio is the ratio of the uncompressed size to the compressed size of the file and we estimate it using a linear model and find a compression ratio of 8.3, 4.9, 5.4, 5.3, 5.5, and 5.8 for BUStools, `gzip -1`, `-9`, `zstd -1`, `-3`, and `-19`, respectively (Fig. 1D).

Together these results show that the compression scheme, as implemented in BUStools, is significantly faster and shows better compression than gzip alone. zstd with the best compression is too slow, but the default (`-3`) and fast (`-1`) options are fast for compression and decompression. The compression speed is 194 MB/s for BUStools versus 222–262 MB/s for the two faster zstd options, however the resulting compressed file is 50% larger when using zstd. This shows that the BUStools compression method is superior for reducing the cost of storage and results in higher throughput of data transfer due to its smaller file size.

Acknowledgements

We thank Ángel Gálvez-Merchán and A. Sina Booeshaghi for help with benchmarking. Atli Fannar Franklín worked on an initial prototype of the software. Lior Pachter provided valuable feedback on the design of the compression and use cases.

Supplementary data

Supplementary data is available at *Bioinformatics* online.

Conflict of interest

None declared.

Funding

This work was supported by the Icelandic Research Fund Project [218111-051].

Data availability

All data used is publicly available, a list of accession ID is given in the Supplementary Material.

References

Booeshaghi AS, Hallgrímsdóttir IB, Gálvez-Merchán Á *et al.* Depth normalization for single-cell genomics count data. *bioRxiv*, 2022. https://doi.org/10.1101/2022.05.06.490859.

Collet Y, Kucherawy M. Zstandard Compression and the 'application/zstd' Media Type. RFC 8878. 2021. https://www.rfc-editor.org/rfc/rfc8478.

Fraenkel AS, Klein ST. Robust universal complete codes for transmission and compression. *Discrete Appl Math* 1996;**64**:31–55.

Fritz MH-Y, Leinonen R, Cochrane G *et al.* Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res* 2011;**21**:734–40.

Melsted P, Booeshaghi AS, Liu L *et al.* Modular, efficient and constant-memory single-cell RNA-seq preprocessing. *Nat Biotechnol* 2021;**39**:813–8.

Melsted P, Ntranos V, Pachter L *et al.* The barcode, UMI, set format and BUStools. *Bioinformatics* 2019;**35**:4472–3.

The Tabula Sapiens Consortium. The tabula sapiens: a multiple-organ, single-cell transcriptomic atlas of humans. *Science* 2022;**376**:eabl4896.

Yan H, Ding S, Suel T. Inverted index compression and query processing with optimized document ordering. In: *Proceedings of the 18th International Conference on World Wide Web, WWW '09*. New York, NY, USA. Association for Computing Machinery, 2009, 401–410.

Zukowski M, Heman S, Nes N *et al.* Super-scalar RAM-CPU cache compression. In: *22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, GA, USA. 2006, 59–59.