**Abstract:**

Background: Genomes within the same species reveal large similarity, exploited by specialized multiple genome compressors. The existing algorithms and tools are however targeted at large, e.g., mammalian, genomes, and their performance on bacteria strains is rather moderate.

Results: In this work, we propose MBGC, a specialized genome compressor making use of specific redundancy of bacterial genomes. Its characteristic features are finding both direct and reverse-complemented LZ-matches, as well as a careful management of a reference buffer in a multi-threaded implementation. Our tool is not only compression efficient, but also fast.

On a collection of 168,311 bacterial genomes, totalling 587\,GB, we achieve the compression ratio around the factor of 1265, and the compression (resp. decompression) speed around 1580\,MB/s (resp. 780\,MB/s) using 8 hardware threads, on a computer with a 14-core~/~28-thread CPU and a fast SSD, being almost 3 times more succinct and over 6 times faster in the compression than the next best competitor.

**Response to Reviewers:**

Dear Editor,

We submit a final version of our manuscript:

"MBGC: Multiple Bacteria Genome Compressor" by Szymon Grabowski and Tomasz M. Kowalski.

The changes are:
* the GigaDB DOI is cited under "Availability of Supporting Data" and included in the References,
* references are all augmented with DOIs,
* bio.tools ID and SciCrunch.org RRID are now included in the "Availability of source code and requirements" section.
Additionally, all the color marking (red), previously introduced for Reviewers' convenience, is removed from the manuscript.

Yours sincerely,
Szymon Grabowski (on behalf of both co-authors)

| Additional Information: |  |
|-------------------------|--|
| **Question** | **Response** |
| Are you submitting this manuscript to a special series or article collection? | No |
| **Experimental design and statistics** | Yes |
| Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our Minimum Standards Reporting Checklist. Information essential to interpreting the data presented should be made available in the figure legends. |  |
| Have you included all the information requested in your manuscript? |  |
| **Resources** | Yes |
| A description of all resources used, including antibodies, cell lines, animals and software tools, with enough information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite Research Resource Identifiers (RRIDs) for antibodies, model organisms and tools, where possible. |  |
| Have you included the information requested as detailed in our Minimum Standards Reporting Checklist? |  |
| **Availability of data and materials** | Yes |
| All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using |  |
a unique identifier in the references and in the “Availability of Data and Materials” section of your manuscript.

Have you met the above requirement as detailed in our Minimum Standards Reporting Checklist?
MBGC: Multiple Bacteria Genome Compressor

Szymon Grabowski\textsuperscript{1,\ast,†} and Tomasz M. Kowalski\textsuperscript{2,\ast,†}

\textsuperscript{1}Institute of Applied Computer Science, Lodz University of Technology, Poland
\textsuperscript{\ast} {sgrabow,tkowals}@kis.p.lodz.pl
\textsuperscript{†} Contributed equally.

Abstract

Background Genomes within the same species reveal large similarity, exploited by specialized multiple genome compressors. The existing algorithms and tools are however targeted at large, e.g., mammalian, genomes, and their performance on bacteria strains is rather moderate.

Results In this work, we propose MBGC, a specialized genome compressor making use of specific redundancy of bacterial genomes. Its characteristic features are finding both direct and reverse-complemented LZ-matches, as well as a careful management of a reference buffer in a multi-threaded implementation. Our tool is not only compression efficient, but also fast. On a collection of 168,311 bacterial genomes, totalling 587 GB, we achieve the compression ratio around the factor of 1265, and the compression (resp. decompression) speed around 1580 MB/s (resp. 780 MB/s) using 8 hardware threads, on a computer with a 14-core / 28-thread CPU and a fast SSD, being almost 3 times more succinct and over 6 times faster in the compression than the next best competitor.

Key words: Algorithms, Data compression, Multiple genome compression, FASTA, Pathogens

Background

Genome compression is a fairly old research topic, dating back to mid-1990s [1]. It was soon realized that even sophisticated techniques for compressing a single genome, e.g., [2], cannot offer much higher compression ratios than simple packing of DNA symbols into 2 bits per each (see also the recent experimental comparison [3]). The interest of researchers thus shifted into relative compression of a genome given a reference [4, 5, 6, 7, 8], typically representing the same species, or compression of a given collection of genomes without an external reference [9, 10, 11]. Some of those proposals apply quite advanced techniques (e.g., GDC 2 [11], GeCo3 [12]), while other use rather simple input preprocessing followed by a general-purpose backend compressor, like 7zip in DELIMINATE [13] or zstd in NAF [14]. For example, GDC 2 uses two-pass LZ77 matching and the matches in the latter pass can be built of several matches found in the former pass, to obtain unsurpassed compression ratios on large human genome collections (e.g., the ratio of around 9,500 on 1092 human diploid genomes). GeCo3 [12] combines the power of neural networks with specific DNA models, but its compression on a 2–4 gigabyte genomic collections already takes hours. Allowing mismatches (mutations) in matches, leveraging thus a generalized notion of a standard LZ–match, proved successful in MemRGC [8], a relative compressor for a single genome.

The abundance of full genomes available in major repositories, like NCBI or 1KGP, in recent years poses a challenge to compress them efficiently, preferably combining high compression ratios, fast compression and decompression, and reasonable memory requirements. In this work, we focus on the compression of bacterial genomes (without an external reference), for which existing genome collection compressors are not appropriate from algorithmic or technical reasons (e.g., ignoring reverse-complemented matches, slow compression of long DNA sequences interspaced with EOL symbols, lack of N symbol support or constraints concerning the number of sequences in a single FASTA file). We note that there exist also other possible aspects of compressors (or compression-based tools), e.g., random access support [15, 9, 16] or searching directly in the compressed data, also in an approximate manner [17]. For more references, see the survey [18].

Compiled on: December 14, 2021.
Draft manuscript prepared by the author.
Key Points

- Bacterial genomes are highly similar, due to repeating direct and reverse-complemented substrings.
- Our bacterial genome compressor, MBGC, achieves compression ratios up to above 1000, much higher than its competitors.
- A careful multi-threaded implementation allows to reach (de)compression speed above 1 GB/s on a mid-end workstation.

Analyses

For the experiments we took a large collection of 168,311 bacterial genomes in the FASTA format from the NCBI Pathogen Detection project, and four 1024-genome subsets of it, each representing a single species (except for a joint subset with E. coli and Shigella genomes). The proposed MBGC and other compressors were tested on a Linux (Debian) machine equipped with a 14-core Intel Core i9-10940X 3.3 GHz CPU, 128 GB of DDR4-RAM (CL 16, clocked at 2666 MHz) and a fast SSD (ADATA 2 TB M.2 PCIe NVMe XPG SX8200 Pro). MBGC is written in C++ and was compiled with gcc 10.2.1. The disk cache of-Line (EOL) symbols were removed from the DNA strings in the input files prior to the experiment. As a side note, we point out that MBGC accepts EOL symbols in the input, but does not to hamper their compression in any way. First, the End-Of-Line (EOL) symbols were removed from the DNA strings in our experiments, which can be explained by running 12 blocks of (up to) 2 GB each in parallel.

We point out that for the purpose of testing general-purpose tools (zstd, BSC and 7zip) we applied a unified strategy not to hamper their compression in any way. First, the End-Of-Line (EOL) symbols were removed from the DNA strings in the input files prior to the experiment. As a side note, we point out that MBGC accepts EOL symbols in the input, but does not preserve them in the decompressed output (it uses no EOLs in those strings by default or can insert EOLs in regular gaps in DNA strings, as specified by the user). Second, zstd and BSC work with a single file input (and output) and for this reason we combined the input into a TAR archive (the preprocessing time for the compression process and the postprocessing time for the decompression process were not included).

Preliminary experiments (with 1024 genome collections) show that on the original data (i.e., with EOLs preserved) 7zip needs about 40% more time to compress and its compression ratio is worse by a factor of 2–3. The respective losses are even greater for zstd (around 3–7 in the compression ratio, and 2 in the compression time with regard to the stronger mode). Such striking differences are however understandable; there are many long LZ-matches in our data, which are broken in the decompression) in our experiments, which can be explained by running 12 blocks of (up to) 2 GB each in parallel.

We point out that for the purpose of testing general-purpose tools (zstd, BSC and 7zip) we applied a unified strategy not to hamper their compression in any way. First, the End-Of-Line (EOL) symbols were removed from the DNA strings in the input files prior to the experiment. As a side note, we point out that MBGC accepts EOL symbols in the input, but does not preserve them in the decompressed output (it uses no EOLs in those strings by default or can insert EOLs in regular gaps in DNA strings, as specified by the user). Second, zstd and BSC work with a single file input (and output) and for this reason we combined the input into a TAR archive (the preprocessing time for the compression process and the postprocessing time for the decompression process were not included).

Preliminary experiments (with 1024 genome collections) show that on the original data (i.e., with EOLs preserved) 7zip needs about 40% more time to compress and its compression ratio is worse by a factor of 2–3. The respective losses are even greater for zstd (around 3–7 in the compression ratio, and 2 in the compression time with regard to the stronger mode). Such striking differences are however understandable; there are many long LZ-matches in our data, which are broken in “random” positions with the EOL characters.

It may be interesting to check the impact of reverse-complement matches on the MBGC performance. It is significant indeed; according to our preliminary experiments, on C. jejuni and L. monocytogenes the compression ratio with RC-matches turned off deteriorates roughly by a factor between 1.1 and 1.7 in the default mode.

Throughout all the presented experiments (except for those
presented in Fig. 3, in MBGC and “nrib” scenarios) the input data are in the uncompressed (FASTA) format. Still, MBGC can read gzipped FASTA and we briefly checked how it affects overall performance. The gzipped stream is decompressed with the aid of libdeflate (https://github.com/ebiggers/libdeflate), a library for fast whole-buffer Deflate-based decompression (and compression as well, but we use it only for reading). On the individual species collections the compression time gets slightly better (e.g., by even 21% for S. enterica), with 12.5% speedup for the whole genome collection (all numbers with respect to the default mode of MBGC). The compression ratio varies a little (due to unpredictable access to genomes with the worker threads), usually below 1%.

Fig. 1 shows the compression ratio and compression speed with varying the number of threads from 1 to 28. The speed does not improve with more than 6 threads (but perhaps it would with even more efficient disk I/O). The compression ratio is rather unaffected for E. coli and S. enterica, but using already more than 1 thread for C. jejuni and L. monocytogenes yields a few percent compression loss. For C. jejuni the gap is as large as about 15% when the number of threads grows from 1 to (the default) 8. On the other hand, using 8 threads is about 3–4 times faster than 1 thread in the compression for all four datasets and for this reason we find the compression loss in half of the cases rather acceptable.

Finally, in Fig. 2 we can see how the compression ratio and compression times change when more and more genomes are given as the input. The number of threads was set to 8 (default). As expected, the compression time grows roughly linearly (note the X-axis scale), but the compression ratio improves, as for further genomes more similar “pieces” can be found in the already processed collection (or, to be more precise, in the currently used REF sequence). The only exception is E. coli, where after processing about 2,000 genomes the compression ratio first deteriorates somewhat and then no longer improves. This can be easily explained by the heterogeneity of this dataset, which comprises both E. coli and (closely related to E. coli, but different) Shigella genomes.

For a separate experiment, we took two non-bacterial genome collections, S. cerevisiae and S. paradoxus (Table 4). We didn’t expect MBGC to be competitive here, and indeed, GDC 2 and 7z are superior in the compression ratio but MBGC remains the second fastest (after zstd –3) tool in the compression process while still maintaining a relatively high compression ratio. A better overall choice is, however, GDC 2, with a significantly higher compression ratio and being only slightly slower in the compression than MBGC max on S. cerevisiae. On the other hand, the compression speed difference is more than 6-fold, in favor of MBGC max, in case of S. paradoxus. In decompression, zstd is the fastest, followed by GDC 2 and 7z, and then by NAF and MBGC. BSC and HRCM are more than twice slower in the decompression than MBGC. Of these two, HRCM is a better pick due to higher compression ratio and relatively fast compression.

In the supplementary material we also present compression results for a small collection of human genomes (hg16, …, hg19). Although these kinds of data are not the target of MBGC, our tool performs satisfactorily here as well, with quite competitive compression ratios and speed.

### Potential implications

Our experiments with MBGC show that the genomes of some bacteria species can be collectively compressed by a factor exceeding 1000, at the (de)compression speed over 1 GB/s (as shown on the total collection of 168k pathogen individuals). This may be an argument for replacing the dominating gzip compression format with a much more resource-effective solution in DNA repositories, both for the end user (i.e., faster dissemination of genomic data) and the data resource management (e.g., easier backup).

A slightly less obvious, but still promising application, could be using the proposed format for rapid download. To this end, the genomes selected by a user to download could then be lumped together and compressed by a factor, say, between 10 and 100 (depending on the count and similarity of the datasets of choice), which is likely to offset the cost of the compression process. It is not clear if and how caching compressed groups of genomes downloaded together could improve this process, yet this possibility and resulting tradeoffs seem worth exploring.

Fig. 3 presents a combined measure expressed as the total time to transfer (download) the entire collection of our test genomes. Each bar consists of three parts, the compression time (at our test machine), the transfer time (assuming the network connection link of 10 Mbit/s or 100 Mbit/s, on the left and right figure, respectively), and the decompression time (at the same test machine). The used compression switches are: NAF –3 –long=31, pigz –6, mbgc –c 1 (default). The input for MBGC are the original gzip files (as provided in NCBI). The input FASTA files for pigz were stripped of EOL symbols prior to the compression. The “tat” bars basically correspond to trans-

---

**Figure 1.** Relative compression ratios and times in the function of the number of threads, with respect to the default mode of MBGC. The left (resp. right) Y axes are related to relative compressed ratios (resp. compression times). On both (left and right) Y axes smaller is better.
mitting raw FASTA files, where the compression phase is data tarring (merging), and the decompression phase is data un-tarring. The “ncbi” bars correspond to the gzip archives in the NCBI repository, where the compression time comprises only tarring the data (so, it is some lower bound estimation). Clearly, MBGC has a huge edge over the competitors, and only NAF comes relatively close with the faster network connection. Note also that even with a faster connection the gzip-based approaches (bars “ncbi” and “pigz”, which is multithreaded gzip implementation) are more than an order of magnitude slower than MBGC. The gaps are generally greater with a slower connection, and for the transfer of 10 Mbit/s the advantage of MBGC over NAF is more than 3-fold and by a factor of almost 6 for the transfer of ALL.

Methods

There is significant redundancy in bacterial genomes which cannot be fully exploited using existing multiple genome compressors. The standard approach of finding repetitions between the currently processed genome and a reference genome (or possibly all previously processed genomes), and encoding them as LZ-phrases of the form (offset, length), is only moderately successful. We found out that many strings repeat as reverse-complements of corresponding strings from other genomes, a phenomenon known, but surprisingly rarely handled earlier (the COMRAD tool being an exception [22]).

It is also beneficial not to limit the reference to one, or a few, previous genome(s), but to allow finding matches occurring almost anywhere earlier. This, however, requires a potentially unbounded memory buffer. We mitigate this problem with building the reference string, i.e., a reservoir for possible matches, in an incremental manner, appending only blocks which are “new enough”, that is, containing a relatively large fraction of DNA subsequences not seen before. This (general) approach, i.e., building “a dictionary of repeats”, is known in the context of relative genome compression, see, e.g., [9] and [23].

As the key ideas of our solution, Multiple Bacteria Genome Compressor (MBGC), are already sketched, now we present the algorithm in detail.
Basic algorithm

The main stages of MBGC compression are the contig matching process and the backend compression of matching products. Below we focus on explaining the former, essential stage.

The goal is to compress the sequence of genomes \( G_1, \ldots, G_n \) in the FASTA format. The genomes consist of one or many contigs (by a contig, throughout the paper, we mean a sequence in the FASTA file). At the start the reference string \( REF \) is initialized with \( \mathcal{r}(G_1) \), where \( \mathcal{r}(\cdot) \) stands for the reverse complement of the passed string. MBGC also stores a literal buffer, which is initialized with \( REF \) (but not its reverse complement). During the compression process, a hash table of fixed size (e.g., \( 2^{25} \) slots) is maintained, and the pairs of the form \((h, pos)\) are inserted into it, where the positions \( pos \) are taken from \( REF \) accessed sparsely, with a stride of 16 symbols, and \( h \) are the hash values of corresponding \( k \)-mer seeds taken from the sampled positions. A collision on the hash \( h \) overwrites the previous value associated with it.

In the following steps the genomes \( G_2, \ldots, G_n \) are taken one by one and LZ-matches of the form \((offset, length)\), where \( offset \) is the position of \( REF \) where a match of length \( length \) begins, are sought. The contigs in the current genome are processed in their original order. If a match is not found for the given position (note that such a check takes a constant time, due to the extremely simple hash table organization), we move to the next position in the current contig, etc., and once we have a (tentative) match, we verify its \( k \) symbols and try to extend it maximally in both directions (with a restriction that matches cannot cross contig boundaries). The left extension of the current match is allowed to “swallow” the (whole) previous match(es). Surprisingly, this little idea is a powerful optimization trick which improves the compression ratio sometimes by more than 50% on our datasets, and is also moderately beneficial for the compression speed, as there are significantly fewer LZ-matches for further encoding. To make this effect even stronger (by up to a few percent), the next position after a match is decreased by \( m \) (which is 16 by default). Using such a “skip margin” in some cases allows to find longer matches.

Finally, the symbols between matches are added to the literal buffer. At this point, we can define the strategy for augmenting the \( REF \) string depicted in Fig. 4. Once we are at the end of a contig, the portion of its symbols not covered with matches is checked; if it is large enough (exceeds \( 1/4 \) of the contig length, where \( u = 192 \) by default), the \( REF \) string is appended with the contig and its reverse complement. The rationale is that contigs too similar to some parts of \( REF \) are almost completely redundant and thus do not contribute enough to facilitate compression, but increase the memory requirement. This design decision was indeed very successful, as in our test data the string \( REF \) together with the concatenated literals often took less than 2% of the input. If, however, the contigs to compress are not similar enough to the previous ones, the \( REF \) string grows quickly and may reach its limit, which depends on the number of genomes in the collection and the size of the first genome (details in the supplementary material).

From this point on, the \( REF \) string works like a circular buffer, i.e., instead of being appended it is being overwritten from the starting position.

The resulting streams of match data (offsets, lengths), literals, header and filename data, and flags are compressed with LZMA and PPMd, using a well-known open-source software development kit (LZMA SDK).

For easier understanding of the MBGC internals, we created an example (see Fig. 5 in the supplementary material). Moreover, the last section of the supplementary material covers the details of backend compression of the streams resulting from the matching stage.

MBGC in the max mode

The description above corresponds to the single-threaded version of our algorithm. MBGC is, however, multi-threaded. The max MBGC’s mode (invoked as mbgc -c 3), with preference to the compression ratio rather than compression speed, does not use multithreading except for parallel input, backend compression and possibly gzip decompression; to understand such a design decision, see Fig. 1 and the related discussion.

We note that in the max mode the initialization of \( REF \) with \( G_1 \) is not required. Since matching is sequential, it can be started from \( G_1 \) instead of \( G_2 \) (even with an empty \( REF \) sequence). \( G_1 \) will be used to extend \( REF \) before matching the remaining genomes.

As multi-threaded matching implementation uses more memory, upper-bounding the reference buffer by \( 2^{25} \) bytes in the default mode helps to reduce the memory consumption during the compression of larger collections. On the other hand, in the max mode the buffer is allowed to grow up to \( 2^{25} \) bytes, which is beneficial for the compression ratio.

The last major difference between MBGC compression modes concerns backend compression. To optimize the performance in the default mode, the most time consuming of the resulting streams (i.e., match data and literals) are broken down into blocks and compressed in parallel, sacrificing however some compression ratio.

Multi-threading

MBGC makes use of the producer–consumer dataflow pattern. Assuming \( t \) worker threads, we have at most \( t - 1 \) producers and at least one consumer for the compression. The producers decompress and handle the input (gzip) files in parallel and store them in buffers (if the input file is uncompressed, the gzip decompression phase is simply skipped); each producer can handle up to 32 files (genomes) in its buffer. The consumer parses headers and contigs, and performs the actual compression (maintaining the hash table, finding LZ-matches, etc.). Once a producer fills up its buffer, it switches to compress the next unprocessed genome (entering a temporary consumer mode), which serves as a simple load balancing technique.

When a genome is fully encoded, the \( REF \) sequence is prolonged with the relevant contigs; updates to \( REF \) are performed in a critical section, preserving the original genome order (via a queuing mechanism). Let us explain this issue in more detail. We take care that the area of \( REF \) in which a worker looks for matches is not overwritten with newer contigs by other workers. To this end, when a worker \( W \) begins its job, it marks a guard position in the \( REF \) which prevents other workers from
overwriting REF beyond this position until W terminates processing a current genome. It might mean that some contigs cannot be written to REF and are thus ignored. Fortunately, in our experiments this detrimental effect hampers the compression ratio rather negligibly. When the buffer of a producer is not full, the producer again fills up its buffer by reading and processing the input data, and the consumer proceeds to compress new genomes.

Availability of source code and requirements

- Project name: MBGC: Multiple Bacteria Genome Compressor
- Project home page: https://github.com/kowallus/mbgc
- Operating system(s): Linux
- Programming language: C++
- Other requirements: C++14 standard or higher, cmake 3.4 or higher
- License: e.g. GNU GPL v3.0
- biotools ID: mbgc
- RRID:SCR_021875

Availability of supporting data and materials

The pathogen data sets supporting the results of this article are available in the US National Center for Biotechnology Information repository: https://www.ncbi.nlm.nih.gov/pathogens. The yeast datasets (S. cerevisiae and S. paradoxus) genomes were taken from Sanger Institute repository: ftp://ftp.sanger.ac.uk/pub/users/dmc/yeast/latest/.

All benchmark data are available online: http://coach.kis.p.lodz.pl/mbgc-datasets/.

Snapshots of our code and other data further supporting this work are openly available in the GigaScience repository, GigaDB [24].

Declarations

List of abbreviations

1KGP: 1000 Genomes Project; BSC: Block Sorting Compressor; COMRAD: COMpression using RedundAncy of Dna; EOL: End-Of-Line; GCC: GNU Compiler Collection; GDC2: Genome Differential Compressor 2; HRCM: Hybrid Referential Compression Method; MBGC: Multiple Bacteria Genome Compressor; MuGI: Multiple Genome Index; NAF: Nucleotide Archival Method; MBGC: Multiple Bacteria Genome Compressor; MuGI: Multiple Genome Index; NAF: Nucleotide Archival Format; NCBI: National Center for Biotechnology Information; LZ: Lempel-Ziv; LZMA: Lempel-Ziv-Markov chain-Algorithm; PPfM: Prediction by Partial Matching (variant by Dmitry Shkarin).

Funding

This work was partially supported by the Faculty of Electrical, Electronic, Computer, and Control Engineering, Lodz University of Technology, as a statutory activity (both authors).

Author’s Contributions

S.G. developed the overall conception and participated in the design of the work and in drafting the manuscript. T.M.K. participated in the design of the work, implemented the tool and conducted all major experiments, and participated in drafting the manuscript. Both authors reviewed and approved the final manuscript.

Acknowledgements

Not applicable.

References

1. Grumbach S, Tahi F. Compression of DNA sequences. In: Proc. Data Compression Conference, IEEE; 1993. p. 340–350. https://doi.org/10.1109/DCC.1993.253115.

2. Cao MD, et al. A simple statistical algorithm for biological sequence compression. In: Proc. Data Compression Conference, IEEE; 2007. p. 43–52. https://doi.org/10.1109/DCC.2007.7.

3. Kryukov K, et al. Sequence Compression Benchmark (SCB) database—A comprehensive evaluation of reference-free compressors for FASTA-formatted sequences. GigaScience 2020;9(7):1–12. https://doi.org/10.1093/gigascience/giaa072.

4. Christley S, et al. Human genomes as email attachments. Bioinformatics 2009;25(2):274–275. https://doi.org/10.1093/bioinformatics/btn582.

5. Pavlichen D, et al. The human genome contracts again. Bioinformatics 2013;29(17):2199–2202. https://doi.org/10.1093/bioinformatics/btt362.

6. Ochoa I, et al. iDoComp: a compression scheme for assembled genomes. Bioinformatics 2015;31(3):626–633. https://doi.org/10.1093/bioinformatics/btu698.

7. Yao H, et al. HRCM: An Efficient Hybrid Referential Compression Method for Genomic Big Data. BioMed Research International;2019:3108950. https://doi.org/10.1155/2019/3108950.

8. Liu Y, et al. Allowing mutations in maximal matches boosts genome compression performance. Bioinformatics 2020;36(18):4675–4681. https://doi.org/10.1093/bioinformatics/btaa572.

9. Deorowicz S, Grabowski S. Robust relative compression of genomes with random access. Bioinformatics 2011;27(21):2979–2986. https://doi.org/10.1093/bioinformatics/btr505.

10. Wandelt S, Leser U. FRESCO: Referential compression of highly similar sequences. IEEE/ACM Transactions on Computational Biology and Bioinformatics 2013;10(5):1275–1288. https://doi.org/10.1109/tcbb.2013.122.

11. Deorowicz S, et al. GDC 2: Compression of large collections of genomes. Sci Rep 2015;5:11565. https://doi.org/10.1038/srep11565.

12. Silva M, et al. Efficient DNA sequence compression with neural networks. GigaScience 2020;9(11). https://doi.org/10.1093/gigascience/giaa119, giaa119.

13. Mohammed MH, et al. DELIMINATE—a fast and efficient method for loss–less compression of genomic sequences.
| Table 1. Compression results – collections of 1024 genomes. |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|                | HRCM            | BSC             | 7z              | zstd –3         | NAF –3          | NAF –19         |
|                 | -p -b2047       | -md4g           | –long=31        | –long=31        | –long=31        | –long=31        |
| C. jejuni (1.78 GB) |
|                 | ratio           | ctime (1)        | dtime (2)       | cmem (3)        | dmem (4)        |                 |
|                 | 15.0            | 40.2             | 2.70 (3)        | 1.28 (1)        | 8.81 (2)        |                 |
|                 | 196.3           | 38.0             | 1064.5 (2)      | 9.00 (2)        | 11.79 (2)       |                 |
|                 | 27.0            | 10.6             | 123.0 (3)       | 4.95 (3)        | 131.1 (2)       |                 |
|                 | (2) 1.76        | (3) 9.00         | (2) 18.00       | (2) 1.92        | (2) 1.90        |                 |
|                 | 0.28            | 8.81             | 1.79            | 1.76            | 1.76            |                 |
|                 | 1.88            | 24.04            | 45.98           | 2.31            | 2.30            |                 |
|                 | 0.66            | 21.48            | 4.82            | 2.15            | 2.21            |                 |
| E. coli (4.87 GB) |
|                 | ratio           | ctime (1)        | dtime (2)       | cmem (3)        | dmem (4)        |                 |
|                 | 158.2           | 166.9            | 377.5           | 357.8           | 495.2           |                 |
|                 | 308.3           | 127.4            | 2228.7          | 10.0 (1)        | 20.8 (2)        |                 |
|                 | 28.9            | 42.4             | 6.5             | 2.9 (3)         | 12.6 (2)        |                 |
|                 | (2) 1.88        | (3) 24.04        | (2) 45.98       | (3) 2.30        | (3) 2.65        |                 |
|                 | 0.66            | 21.48            | 4.82            | 2.15            | 2.21            |                 |
| L. monocyt. (3.09 GB) |
|                 | ratio           | ctime (1)        | dtime (2)       | cmem (3)        | dmem (4)        |                 |
|                 | 39.3            | 70.4             | 2.688           | 82.9            | 131.1           |                 |
|                 | 225.5           | 123.5            | 1838.1          | 4.5 (2)         | 13.8 (2)        |                 |
|                 | 26.0            | 36.7             | 4.0             | 2.1 (3)         | 7.6 (2)         |                 |
|                 | (2) 1.79        | (3) 15.39        | (2) 31.82       | (3) 2.31        | (3) 2.63        |                 |
|                 | 0.30            | 15.30            | 3.07            | 2.15            | 2.15            |                 |
| S. enterica (5.2 GB) |
|                 | ratio           | ctime (1)        | dtime (2)       | cmem (3)        | dmem (4)        |                 |
|                 | 268.4           | 130.0            | 472.4           | 427.9           | 547.5           |                 |
|                 | 308.2           | 130.6            | 2350.3          | 6.1 (2)         | 20.4 (2)        |                 |
|                 | 28.2            | 43.0             | 5.9             | 3.0 (3)         | 12.8 (2)        |                 |
|                 | (2) 1.91        | (3) 25.67        | (2) 49.45       | (3) 2.30        | (3) 2.64        |                 |
|                 | 0.48            | 25.68            | 5.15            | 2.15            | 2.17            |                 |

The rows “ratio” show the ratio of the input to the output size. Compress / decompress times (as “ctime” / “dtime”) are given in seconds, memory usages (“cmem” / “dmem”) given in GB (G = 10^9). The best three results in a row are marked with a number in parentheses. HRCM is single-threaded (except for the latter phase where it invokes 7zip), BSC uses 12 threads, 7zip (up to) 6 threads, zstd 14 threads and MBGC 8 threads.

---

Sequence analysis. Bioinformatics 2012;28(19):2527–2529. [https://doi.org/10.1093/bioinformatics/bts467](https://doi.org/10.1093/bioinformatics/bts467).

14. Kryukov K, et al. Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences. Bioinformatics 2019;35(19):3826–3828. [https://doi.org/10.1093/bioinformatics/btz344](https://doi.org/10.1093/bioinformatics/btz344).

15. Kuruppu S, et al. Relative Lempel–Ziv Compression of Genomes for Large-Scale Storage and Retrieval. In: Chávez E, Lonardi S, editors. String Processing and Information Retrieval – 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11–13, 2010. Proceedings, vol. 6393 of Lecture Notes in Computer Science Springer; 2010. p. 420–426. [https://doi.org/10.1007/978-3-642-16321-0_20](https://doi.org/10.1007/978-3-642-16321-0_20).

16. Belazzougui D, et al. Block Trees. Journal of Computer and System Sciences 2021;117:1–22. [https://dx.doi.org/10.1016/j.jcss.2020.11.002](https://dx.doi.org/10.1016/j.jcss.2020.11.002).

17. Rahn R, et al. Journaled string tree—a scalable data structure for analyzing thousands of similar genomes on your laptop. Bioinformatics 2014;30(24):3499–3505. [https://doi.org/10.1093/bioinformatics/btu438](https://doi.org/10.1093/bioinformatics/btu438).

18. Gagie T, Puglisi S. Searching and Indexing Genomic Databases via Kernelization. Front Bioeng Biotechnol 2015;3. [https://dx.doi.org/10.3389/fbioe.2015.00012](https://dx.doi.org/10.3389/fbioe.2015.00012).

19. Kuhle A, et al. Efficient Construction of a Complete Index for Pan-Genomics Read Alignment. J Comput Biol 2020;27(4):500–513. [https://dx.doi.org/10.1089/cmb.2019.0309](https://dx.doi.org/10.1089/cmb.2019.0309).

20. Sherman RM, Salzberg SL. Pan-genomics in the human genome era. Nature Reviews Genetics 2020;21:243–254. [https://dx.doi.org/10.1038/s41576-020-0210-7](https://dx.doi.org/10.1038/s41576-020-0210-7).

21. Danek A, et al. Indexes of Large Genome Collections on a PC. PLoS ONE 2014;9(10):1–12. [https://dx.doi.org/10.1371/journal.pone.0109384](https://dx.doi.org/10.1371/journal.pone.0109384).

22. Kuruppu S, et al. Iterative Dictionary Construction for Compression of Large DNA Data Sets. IEEE ACM Trans Comput Biol Bioinform 2012;9(1):137–149. [https://dx.doi.org/10.1109/TCBB.2011.82](https://dx.doi.org/10.1109/TCBB.2011.82).

23. Kuruppu S, et al. Reference Sequence Construction for Relative Compression of Genomes. In: SPIRE, vol. 7024 of Lecture Notes in Computer Science Springer; 2011. p. 420–426. [https://dx.doi.org/10.1007/978-3-642-24583-1_41](https://dx.doi.org/10.1007/978-3-642-24583-1_41).

24. Grabowski S, Kowalski TM, Supporting data for “MBGC: Multiple Bacteria Genome Compressor” GigaScience Database; 2021. [http://dx.doi.org/10.5524/100967](http://dx.doi.org/10.5524/100967).
Table 2. Compression results – large species collections.

|                | BSC -p | 7z  | zstd -3  | NAF -3   | NAF -19  | MBGC     | MBGC     |
|----------------|--------|-----|----------|----------|----------|----------|----------|
|                | -b2047 | -md4g| -long=31 | -long=31 | -long=31 | default  | max      |
| C. jejuni      | 69.7   | 164.9| 74.9     | 137.1    |          | (1) 176.6| (2) 412.5| (3) 450.6|
| C. jejuni      |         |      |          |          |          | (1) 92.7 | (2) 102.3|          |
| E. coli        | 241.3  | 129.3| (3) 127.3| 280.3    | 250.9    | (1) 78.5 | (2) 81.3 |          |
| E. coli        |         |      |          |          |          | (1) 26.8 | (2) 28.6 |          |
| L. monocyt.    | 128.84 | 122.54| (1) 2.32 | (1) 3.31 | (1) 2.66 | 8.78     | 7.22     |
| L. monocyt.    |         |      |          |          |          | (1) 2.70 | (2) 2.69 |          |
| S. enterica    | 129.21 | 34.59| (1) 2.15 | (1) 3.10 | (1) 3.20 | 5.62     | 5.02     |
| S. enterica    |         |      |          |          |          | (2) 3.36 |          |          |

The rows “ratio” show the ratio of the input to the output size. Compress / decompress times (as “ctime” / “dtime”) are given in seconds, memory usages (“cmem” / “dmem”) given in GB (G = 10^9). The best three results in a row are marked with a number in parentheses. BSC uses 12 threads, 7zip (up to) 6 threads, zstd 14 threads and MBGC 8 threads.

Table 3. Compression results – mixed species collections.

|                | BSC -p | 7z  | zstd -3  | NAF -3   | NAF -19  | MBGC     | MBGC     |
|----------------|--------|-----|----------|----------|----------|----------|----------|
|                | -b2047 | -md4g| -long=31 | -long=31 | -long=31 | default  | max      |
| 168,311 genomes| 5824.0 | 140902.0| (2) 966.1| 2287.3   | 7100.0   | (1) 1266.6| (3) 1414.4|
| 4 × 1024 genomes| 129.79 | 34.59| (1) 2.15 | (2) 4.51 | (3) 5.3 | 8.70 | 10.73 |

The rows “ratio” show the ratio of the input to the output size. Compress / decompress times (as “ctime” / “dtime”) are given in seconds, memory usages (“cmem” / “dmem”) given in GB (G = 10^9). The best three results in a row are marked with a number in parentheses. BSC uses 12 threads, 7zip (up to) 6 threads, zstd 14 threads and MBGC 8 threads.
Table 4. Compression results on non-bacterial genome collections, *S. cerevisiae* and *S. paradoxus*.

|                    | S. cerevisiae (39 genomes, 486 MB) | S. paradoxus (36 genomes, 429 MB) |
|--------------------|------------------------------------|-----------------------------------|
|                    | ratio | ctime | dtime | cmem | dmem | ratio | ctime | dtime | cmem | dmem |
| GDC 2              |        | 109.8 | 4.12  | (1)  | 0.57 | (2)  | 0.52  | (2)  | 0.14 | (2)  |
| HRCM               |        | 78.8  | 7.13  |      |      | 52.6 | 8.30  |      |      |      |
| BSC – p – b2047    |        | 52.9  | 10.76 |      |      | 33.8 | 9.59  |      |      |      |
| 7z – md4g          |        | 316.38| (1)  | 0.73 | 4.92 | (3)  | 0.82  | 21.92| (3)  | 0.17 |
| zstd – 3 – long=31 |        | 45.8  | (1)  | 0.98 | (1)  | 0.54 | (1)  | 0.49 |      |      |
| NAF – 3 – long=31  |        | 67.0  | (3)  | 2.80 | 1.02 | (3)  | 0.63  | (3)  | 0.49 |      |
| MBGC default       |        | 30.5  | (3)  | 0.87 | (3)  | 0.40 | (3)  | 0.49 | (3)  | 0.43 |
| MBGC max           |        | 30.5  | (3)  | 0.87 | (3)  | 0.40 | (3)  | 0.49 | (3)  | 0.43 |

The columns "ratio" show the ratio of the input to the output size. Compress / decompress times (as "ctime" / "dtime") are given in seconds, memory usages ("cmem" / "dmem") given in GB (G = 10^9). The best three results in a column are marked with a number in parentheses. HRCM is single-threaded (except for the latter phase where it invokes 7zip), BSC uses 12 threads, 7zip (up to) 6 threads, zstd 14 threads and MBGC 8 threads.
Dear Editor,

We submit a final version of our manuscript:

"MBGC: Multiple Bacteria Genome Compressor"
by
Szymon Grabowski and Tomasz M. Kowalski.

The changes are:
* the GigaDB DOI is cited under "Availability of Supporting Data" and included in the References,
* references are all augmented with DOIs,
* bio.tools ID and SciCrunch.org RRID are now included in the "Availability of source code and requirements" section.

Additionally, all the color marking (red), previously introduced for Reviewers' convenience, is removed from the manuscript.

Yours sincerely,
Szymon Grabowski (on behalf of both co-authors)