FQbin: a compatible and optimized format for storing and managing sequence data

Darío Guerrero-Fernández, Rafael Larrosa, and M. Gonzalo Claros*

University of Málaga, Plataforma Andaluza de Bioinformática-SCBI,
C/ Severo Ochoa 34, 29590 Málaga, Spain
{dariogf,rlarrosa,claros}@uma.es,
http://www.scbi.uma.es

Abstract. Existing hardware environments may be stressed when storing and processing the enormous amount of data generated by next-generation sequencing technology. Here, we propose FQBIN, a novel and versatile tool in C for compressing, storing and reading such sequencing data in a new and FASTA/FASTQ-compatible format that outperforms the existing proposals. It is based on the general-purpose ZLIB library and offers up to 10X compression. The compressed file is read and decompressed up to 3X faster than a FASTQ file is read, and a nearly 'instant' random access to every entry in the FQBIN container is allowed. Fast file reading is maintained even in shared storage environments, where different processes are simultaneously accessing the same FQBIN file. Slow networks can take even more advantage from FQBIN.

Keywords: FastQ, compression, zLib, NGS, pipeline, workflow

1 Introduction

The advent of next-generation sequencing (NGS) techniques enabled the generation of an overwhelming and ever growing amount of information in short periods of time [5]. This may saturate existing hardware environments; the main computational concerns is related to CPU time required to process such amount of data, storage capabilities, and data transmission over not so fast networks. In fact, storage systems are the real bottleneck in relation to NGS data processing [5].

Compression is obtained when sequences and the accompanying quality values (QVs) [6] are transformed in the more compact FASTQ format [3]. Due to the 4-nucleotide nature of DNA sequences, it is immediate to think about a better compression based on two-bit conversion storage [1]. This solution is adequate for compressing files containing alignments and mapping, but does not fit when more information, such as QVs, is to be gathered with sequences. DSRC algorithm [4] is worth mentioning, although only compresses FASTQ files, since it is appropriate for most genomics approaches. It uses an internal block structure to

* Corresponding author

provide random access and decrease the file size from four to six times. There are other compressing methods that do not offer sufficient compression rates, or require a great amount of CPU time for decompression and loading every time the data are accessed [9]. Therefore, when one wants to compress data containing sequences, QVs and something else (e.g., metadata, flowgrams, images), compression tools such as DNACompress [1], PBAT, DNAZip, SlimGene, MZip, ReCoil, or SpeedGene [9], to cite a few, are not appropriate. Finally, binary SFF (Standard Flowgram Format) files contain sequence IDs, sequences, base call QVs, flowgram, and can contain information on how sequences need to be clipped; unfortunately, the final file size does not present any advantage.

Since FASTA, QUAL and FASTQ formats are widely used in bioinformatics, there is no option to think about a complete transition to a new, incompatible format. Therefore, here it is described FQBIN, a file container and a set of command line tools in C that provide—using a general purpose compression algorithm—a compressed, FASTA/FASTQ-compatible format that improves compression size and sequence access compared to other algorithms.

2 Methods

FQBIN is based on the compression library ZLIB (`http://zlib.net/`) wrapped in a shared library with a set of C command line tools for compression, decompression and random access. It is additionally available as a Ruby gem.

2.1 FQbin container format

FQBIN container gathers individual FASTA, QUAL and EXTRAS fields compressed in separate chunks for each sequence. Compressed chunks are then saved to disk interleaved with a header field for each sequence that will facilitate the random access to any sequence. The simplified scheme of the FQBIN container is shown in Fig. 1. It starts with a *file header*, in which the first 4 bytes define its variable length, then a variable string containing a format identifier, and then version and subversion fields to deal with future upgrades. Next, the first *compressed block* of data contains a maximum of 10 000 sequence records, compressed as a unique ZLIB stream. Every *sequence record* in the compressed block contains a sequence header and the remaining data (base calls, QV, EXTRAS). The *sequence header* starts with 4 bytes that define its variable length. This is followed by four string-fields indicating the name (that serves as an identifier [ID] for random access), the sequence length, the number of QVs, if any, and the length of EXTRAS, if available. Once the block is full of records, the stream is closed and a new one is created. Organization in blocks requires decompression only of a single block—instead of the whole file—to gain access to a particular sequence, saving time and disk usage. As will be explained later (section 3.3), this separation in blocks also serves as a kind of firewall against data corruption. Compatibility with current and legacy software is guaranteed since the FQBIN contents can be streamed to another program that reads FASTA/QUAL and FASTQ formats (see section 2.4 below).

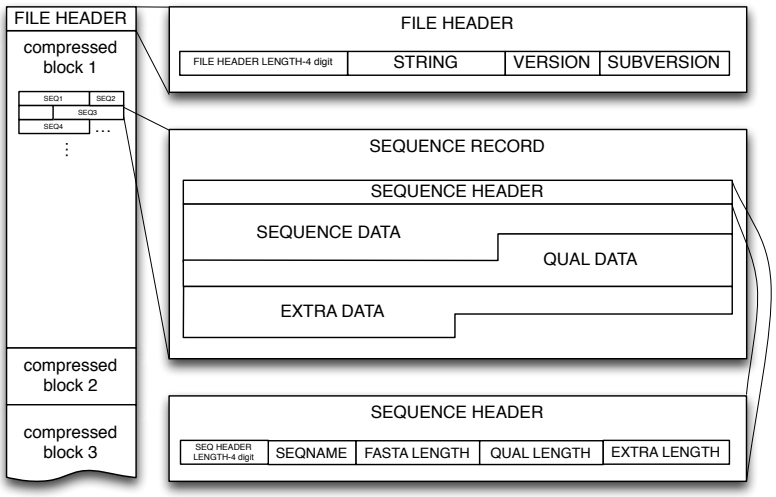

Fig. 1. Scheme of FQBIN container. The general format structure is shown on the left. Detailed description of every block is on the right. See text for details

2.2 Random access

FQBIN container and tools assure a fast random access to sequences by their ID by means of two external and regenerable index and hash files (see section 2.4). Accessing to a random sequence means finding the ID in the small, alphabetically ordered, *hash file* to get its position in the compressed index file. The *index file* is accessed directly at the position of the ID, which indicates the location of the sequence information corresponding to such ID within the main FQBIN container. Finally, the compressed block containing the target ID is read and inflated in RAM, and sequence, QVs and EXTRAS are directly retrieved. Therefore, only a small portion of data is loaded to get access to the sequence, and only a small RAM portion is used even if the file size is much bigger than the RAM size. Moreover, the uncompressed sequence is dumped to `stdout` in FASTA/FASTQ format, avoiding the need of further sequence readers.

2.3 Simplifying quality values

Since QVs are quite repetitive in the useful sequence [10], an additional QV compression step was included. The following rules are applied to all QVs of every sequence, and only the optional steps involve some shallow loss of QV information.

- Conversion of QV numerical data into FASTQ characters when original data are given as FASTA + QUAL files.
- Discretization (optional): QV range (usually 1-40 in NGS) is divided into intervals of customizable length; QVs within each interval are replaced by

their corresponding discretized QV in order to reduce complexity and promote deeper compression by zLib. The discretization formula is $QV_{discretized}[i] = trunc(QV[i]/customized\_length) \times customized\_length$.
- Filtering (optional): since the use of QV in NGS post-processing is usually impractical [2, 10], only low QVs could have some interest in order to trim low quality base calls. Therefore, QVs qualified as good (usually QV $\geq$ 20; the cutoff is customizable) are replaced by this cutoff value.
- Simplification of repeated QVs: when the resulting QVs of a sequence have all the same value, they are stored as one single value. At decompression, when a single QV is read, it is repeated the number of times indicated by the sequence length.

2.4 Main FQbin command-line tools

`idx_fqbin` is for re-creation of index file. `hash_fqbin` is for re-creation of hash file. `read_fqbin` is for random access to a sequence. A complete FQbin container can be constructed as `mk_fqbin -e extra_information.fasta -o outputfile.fqbin inputfile.fastq`. Loading of a complete FQbin file and sending every sequence to Blast to be compared against `blast_database` can be commanded as `iterate_fqbin -F file.fqbin | blastn -db blast_database`. Other software that do not accept pipelining can be used by means of named pipes as follows:

```
mk_fifo nam_pipe1                    # creating the named pipe
bowtie2 index nam_pipe1 &            # bowtie2 uses the pipe
iterate_fqbin file.fqbin > nam_pipe1 # sending data into the pipe
rm nam_pipe1                         # deleting the named pipe
```

2.5 Tests

Three different classes of sequences were used: one Illumina dataset SRR314795 (21,908,723 reads, 3.9 GB), one Roche 454/FLX+ SRR073389 (811,509 reads, 0.9 GB) and one Fasta file containing sequences without QVs of the first 10 human chromosomes (1.6 GB: AC_000133.1 to AC_000142.1). FQbin was compared against general compression algorithms, such as Zip (widely used), gZip (a wrapper for zLib), BSC (a high performance file compressor based on lossless, block-sorting data compression algorithms [7]) and DSRC (the better published compressor for FastQ data [4]). All tests have been performed on a quad-core iMac at 2.8 GHz with 8 GB of RAM, using a shared storage mounted with samba through a 1 Gbit/s or 100 Mbit/s Ethernet network. Time measurements were performed with the Unix `time` command.

3 Results and Discussion

It will be demonstrated that FQbin is a robust development that provides an appropriate, compressed format for NGS, and a nearly instant access to any sequence at any moment. Getting the whole dataset from the FQbin container is faster any other possibility.

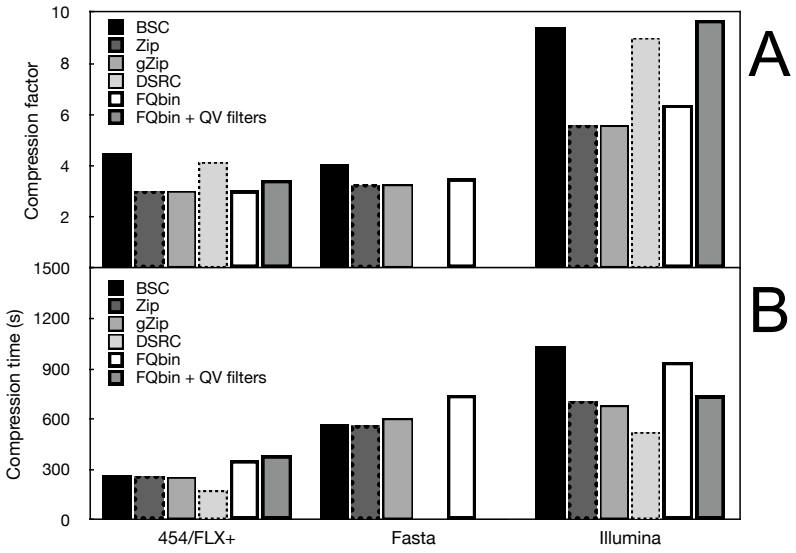

Fig. 2. Compression capabilities of FQBIN compared to other compression algorithms. A: Compression factor with respect to the uncompressed file size calculated as original file size divided by the compressed file size (greater is better). B: Time elapsed for file compression (greater is worse).

3.1 Compression capabilities

When comparing FQBIN with other general and specific compressors (Fig. 2), differences in compression factor become more remarkable with the biggest file (Illumina data, 3.9 GB), with BSC, FQBIN and DSRC providing the better compression factors in all cases (Fig. 2A). It can be seen that as the file size increases, compression factor of FQBIN begins to outperform the other compressors. BSC compares well with FQBIN in compression factor, but compression time is clearly increasing for big files (Fig. 2B), suggesting that it is quite sensitive to the original file size. DSRC does not outperform FQBIN compression of Illumina data and is not able to manage FASTA files, but is the fastest compressor when FASTQ files are involved. GZIP, whose compression engine is also ZLIB, does not supersede FQBIN, although it is a little bit faster during compression. FQBIN with QV filtering is faster and provides higher compression factor with Illumina data than FQBIN without filtering; this confirms the repetitive nature of most QVs in this file, illustrating the advantages of QV filtering [10]. In literature, SpeedGene claims for a compression factor ranging from 16 to several hundreds [9], but it cannot be compared with FQBIN since SpeedGene compresses based on a sequence reference for genome-wide association studies, while FQBIN is focused on storing the whole original data. For genomic data, compression factor is more limiting than compression time, since files will be compressed only once.

Therefore, results in Fig. 2 show that FQBIN presents a good balance between compression factor and compression time, and demonstrate that it compares well with other compression algorithms, being the best for Illumina data.

3.2 Reading compressed files

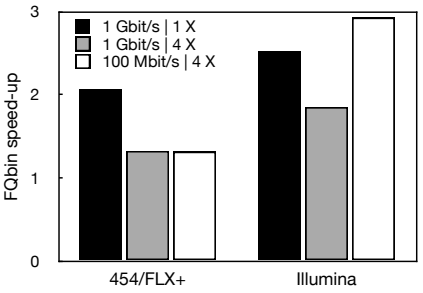

Fig. 3. Whole-file reading speed-up (greater is better) due to FQBIN with respect to the FASTQ reader when 454/FLX+ and Illumina files are read disabling the filesystem cache on networks of 1 Gbit/s and 100 Mbit/s. "1 X" means that only one process is reading the file; "4 X" means that four different processes are accessing the same file at the same time.

Although the file size of a FQBIN container is shorter than the original FASTQ file (Fig. 2), its more complex content (Fig. 1) could make one think that reading the complete set of sequences could take longer from the FQBIN container than from a simple, uncompressed FASTQ file. An optimized script in C was therefore written for time-efficient reading of FASTQ files (Guerrero-Fernández, unpublished results). It should be noted that files compressed using ZIP, GZIP, BSC or DSRC can only be converted to FASTQ files that the optimized script must read. Therefore, if FQBIN outperforms this script in reading flat FASTQ files, it will be more advantageous than any other compression algorithm. Using the same files as above, FQBIN reads the whole 454/FLX+ file in 26 s and Illumina file in 87 s, being 2-3 times faster than the specific FASTQ reader (Fig. 3) and DSRC (63 s and 230 s, respectively). This behavior was also remarkable when four concurrent processes were reading the same file (Fig. 3, '4 X' columns), particularly when big files are read under slow networks (100 Mbit/s): loading the Illumina data decreased from 3329 s (FASTQ) to 1137 s (FQBIN), providing an speed-up of 2.9X. Efficient reading of FQBIN is the result of reading a smaller file and fast data inflation in RAM. In conclusion, FQBIN is not only saving disk space, but also saving accessing time to the file, speeding-up the sequence loading; all together allow a more optimized use of computing resources.

Compressors compared to FQBIN in Fig. 2 require the inflation of the complete file to retrieve a single sequence, but DSRC and FQBIN can access to any particular sequence entry without reading or inflating the whole file. The efficiency in random access can then be assessed comparing the time elapsed in reading the first and the last entry of the file (Table 1). As expected, the sequential process for the FASTQ file takes longer as the file size increases, while reading the last entry was 29X (454/FLX+) to 8.8X (Illumina) faster for FQBIN simply using the index file. Since DSRC behaves better reading the last sequence, the hash index file was included to provide a nearly 'instant' access to any entry in FQBIN, that is, a speed-up ranging from 86X (454/FLX+) to 415X (Illumina).

Moreover, random access of FQBIN is driven by ID, while DSRC use the ordinal position, making it less useful.

In conclusion, FQBIN is also an appropriate format for storing and quick accessing sequences, since data-loading process of one sequence (Table 1) or the whole file (Fig. 3) takes less time than using any other sequence compressor.

Table 1. Accessing time for the first and the last entries of a file

Tested format	454/FLX+	Illumina
FASTQ 1st	0.19	0.14
FASTQ last	37.08	176.30
DSRC 1st	1.36	0.74
DSRC last	1.06	0.79
FQBIN 1st	0.24	0.25
FQBIN last	1.27	19.94
FQBIN hashed 1st	0.219	0.242
FQBIN hashed last	0.429	0.423

3.3 Robustness of FQbin container

When copying files, data are read from the original disk, stored in RAM, and later written down to disk. Since RAM capacity of current personal computers is in constant increase, and most of them use RAM without error-correcting code (ECC), RAM corruption probability is not negligible. Moreover, managing NGS data involves files of several gigabytes, which is also increasing the probability of data corruption. Although this concern is usually absent when computers with ECC memory are involved, personal computers are always operating on several steps of NGS managing. Users are not usually alerted of file corruption during copying, but since FQBIN is based on ZLIB, and ZLIB uses an internal CRC (checking redundancy code) for integrity verification, FQBIN notifies the user when some corruption occurred while copying, allowing users to discard the corrupted file and copying again the original file.

Single-bit corruption of text files does not hinder its reading, even though a minor, unnoticed change in a sequence can lead to undesired side-effects. The same corruption in regular binary files (e.g. compressed or SFF files) can make the whole file unrecoverable. This drawback was minimized in FQBIN by storing compressed data in several independent blocks (Fig. 1): corruption of one block does not affect the other blocks, and most sequences can be recovered.

4 Conclusions

The FQBIN shared library in C offers the following advantages: (i) it provides storage of sequences and/or QVs and/or EXTRAS in the same file, expanding its use beyond the nucleotide sequences. In fact, EuroPineDB database [8] uses the same library to retrieve contigs within large ACE files. (ii) It provides direct-data-retrieving functions supporting input/output for FASTA, QUAL and FASTQ formats, which extends its use from crude FASTQ reads to curated FASTA sequences. Additionally, it allows the compressed information to be easily pipelined from/to other programs, assuring compatibility with existing software without any recoding. (iii) It improves the storage and the analysis of NGS data in current hardware environments. (iv) The file size decrease compares well with other

compression algorithms and formats, improving it as the file size increases, especially in slow networks. (v) It reads the same amount of data in less time than others and provides nearly 'instant' access to any sequence by ID, regardless its position in the container.

Acknowledgement

The authors gratefully acknowledge the computer resources and technical support provided by the Plataforma Andaluza de Bioinformática of the University of Málaga, Spain. This study was supported by grants from the Spanish MICINN (BIO2009-07490) and Junta de Andalucía (P10-CVI-6075), as well as institutional funding to the research group BIO-114.

References

1. Xin Chen, Ming Li, Bin Ma, and John Tromp. Dnacompress: fast and effective dna sequence compression. *Bioinformatics*, 18(12):1696–8, Dec 2002.
2. Manuel Gonzalo Claros, Rocío Bautista, Darío Guerrero-Fernández, Hicham Benzerki, Pedro Seoane, and Noé Fernández-Pozo. Why assembling plant genome sequences is so challenging. *Biology*, 1:439–459, 2012.
3. Peter J A Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic Acids Res*, 38(6):1767–71, Apr 2010.
4. Sebastian Deorowicz and Szymon Grabowski. Compression of dna sequence reads in fastq format. *Bioinformatics*, 27(6):860–2, Mar 2011.
5. Editorial. Prepare for the deluge. *Nat Biotechnol*, 26(10):1099, Oct 2008.
6. B Ewing and P Green. Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome Res*, 8(3):186–94, Mar 1998.
7. P. Fenwick. *Proceedings of the 19th Australasian Computer Science Conference*, chapter Block sorting text compression, pages 1–10. University of Melborune, 1996.
8. Noé Fernández-Pozo, Javier Canales, Darío Guerrero-Fernández, David P Villalobos, Sara M Díaz-Moreno, Rocío Bautista, Arantxa Flores-Monterroso, M Ángeles Guevara, Pedro Perdiguero, Carmen Collada, M Teresa Cervera, Alvaro Soto, Ricardo Ordás, Francisco R Cantón, Concepción Avila, Francisco M Cánovas, and M Gonzalo Claros. Europinedb: a high-coverage web database for maritime pine transcriptome. *BMC Genomics*, 12:366, 2011.
9. Dandi Qiao, Wai-Ki Yip, and Christoph Lange. Handling the data management needs of high-throughput sequencing data: Speedgene, a compression algorithm for the efficient storage of genetic data. *BMC Bioinformatics*, 13:100, 2012.
10. Raymond Wan, Vo Ngoc Anh, and Kiyoshi Asai. Transformations for the compression of fastq quality scores of next-generation sequencing data. *Bioinformatics*, 28(5):628–35, Mar 2012.