Briefings in Bioinformatics, 00(00), 2020, 1–8

doi: 10.1093/bib/bbaa368
Problem Solving Protocol

Pyfastx: a robust Python package for fast random access to sequences from plain and gzipped FASTA/Q files

Lianming Du, Qin Liu, Zhenxin Fan, Jie Tang, Xiuyue Zhang, Megan Price, Bisong Yue and Kelei Zhao

Corresponding authors: Kelei Zhao, Institute for Advanced Study, Chengdu University, Chengdu 610106, China. Tel: +86-28-84216035; Fax: +86-28-84333218; Email: zhaokelei@cdu.edu.cn. Bisong Yue, Key Laboratory of Bio-Resources and Eco-Environment, Ministry of Education, College of Life Science, Sichuan University, Chengdu 610065, China. Tel: +86-28-85416928; Fax: +86-28-85416928; Email: bsyue@scu.edu.cn

Abstract

FASTA and FASTQ are the most widely used biological data formats that have become the *de facto* standard to exchange sequence data between bioinformatics tools. With the avalanche of next-generation sequencing data, the amount of sequence data being deposited and accessed in FASTA/Q formats is increasing dramatically. However, the existing tools have very low efficiency at random retrieval of subsequences due to the requirement of loading the entire index into memory. In addition, most existing tools have no capability to build index for large FASTA/Q files because of the limited memory. Furthermore, the tools do not provide support to randomly accessing sequences from FASTA/Q files compressed by gzip, which is extensively adopted by most public databases to compress data for saving storage. In this study, we developed pyfastx as a versatile Python package with commonly used command-line tools to overcome the above limitations. Compared to other tools, pyfastx yielded the highest performance in terms of building index and random access to sequences, particularly when dealing with large FASTA/Q files with hundreds of millions of sequences. A key advantage of pyfastx over other tools is that it offers an efficient way to randomly extract subsequences directly from gzip compressed FASTA/Q files without needing to uncompress beforehand. Pyfastx can easily be installed from PyPI (https://pypi.org/project/pyfastx) and the source code is freely available at https://github.com/lmdu/pyfastx.

Key words: FASTA; FASTQ; random access; sequence retrieval; sequence parser

Lianming Du, PhD, is an assistant professor at the Institute for Advanced Study, Chengdu University, Chengdu, China. His research interest is focused on bioinformatics and integrated genomics.
Qin Liu, PhD, is a lecturer at the College of Life Sciences and Food Engineering, Yibin University, Yibin, China.
Zhenxin Fan, PhD, is an associate professor at the Key Laboratory of Bio-Resources and Eco-Environment, Ministry of Education, College of Life Science, Sichuan University, Chengdu, China.
Jie Tang, PhD, is an associate professor at the Institute for Advanced Study, Chengdu University, Chengdu, China.
Xiuyue Zhang, PhD, is an associate professor at the Key Laboratory of Bio-Resources and Eco-Environment, Ministry of Education, College of Life Science, Sichuan University, Chengdu, China.
Megan Price, PhD, is a lecturer at the Key Laboratory of Bio-Resources and Eco-Environment, Ministry of Education, College of Life Science, Sichuan University, Chengdu, China.
Bisong Yue, PhD, is a professor at the Key Laboratory of Bio-Resources and Eco-Environment, Ministry of Education, College of Life Science, Sichuan University, Chengdu, China.
Kelei Zhao, PhD, is a professor at the Institute for Advanced Study, Chengdu University, Chengdu, China.
Submitted: 1 March 2020; **Received (in revised form):** 30 October 2020

Introduction

Although a variety of data types have been developed with innovations of omics sequencing technologies, dealing with sequence data represented in structured formats remains the core issue in bioinformatics analysis [1]. Among biological data formats, FASTA is the most common file format for nucleotide and protein sequences, while FASTQ is the most ubiquitous file format for sequencing read data [2]. The FASTA format was originally invented as the input format for the FASTA sequence alignment tool [3]. As a simple extension to the FASTA format, FASTQ can store both nucleotide sequence and its corresponding quality scores [4]. Although FASTA/Q formats similarly suffer from the absence of explicit definitions, they have evolved into the *de facto* standard to exchange sequence data between bioinformatics tools.

Due to the rapid proliferation of high-throughput sequencing technologies, the amount of sequence data being deposited into public databases and accessed in FASTA/Q formats is increasing dramatically. Thus, development of powerful and efficient tools for parsing FASTA/Q format files can greatly facilitate transforming sequence data into biological knowledge. Currently, many tools have been developed to manipulate FASTA/Q formatted files and can be divided into two main categories. The first category comprises tools that can only parse sequences in order such as HTSeq [5], seqmagick (https://github.com/fhcrc/seqmagick), fasta_utilities (https://github.com/jimhester/fasta_utilities), seqtk (https://github.com/lh3/seqtk), FASTX-Toolkit (http://hannonlab.cshl.edu/fastx_toolkit), and fqtools [6]. The second category encompasses tools that have the capability to randomly access sequences such as Biopython [7], BioPerl [8], samtools [9], pysam [9], seqkit [10], pyfasta (https://github.com/brentp/pyfasta) and pyfaidx [11]. The tools in the second category are generally time and memory-efficient when extracting specified subsequences and randomly sampling sequences without scanning the entire sequence file from start to end.

Samtools is the first tool support for random access to sequences by establishing an index file that contains the name, the number of bases, byte offset and line length of each sequence [9]. Once an index file is generated, it can be reused for retrieval of sequences in the future to save time. Pysam, pyfaidx and seqkit generate compatible index files with samtools, while pyfasta, biopython and bioperl generate self-defined index files that are different from samtools. According to the sequence location in the index file, these tools are capable of quickly seeking a sequence start position and reading a given number of bytes. Except for biopython and bioperl, the other tools need to keep index into memory when building index and retrieving sequences, which severely limits the capability of parsing large FASTA/Q files containing hundreds of millions of sequences. To date, only samtools, biopython and bioperl have the capacity to randomly access reads from FASTQ files. However, these tools do not provide support for randomly accessing sequences from gzip compressed files. Although samtools enables random access to a bgzip compressed file, gzip is still the most popular compression tool employed by public databases such as NCBI, Ensembl and UCSC genome browser.

Here, we present pyfastx, a robust and versatile tool with high time- and memory-efficient random access to sequences from both plain and gzipped FASTA/Q files containing large sequence records. Pyfastx is also a fast parser for sequential iterating over sequence records from FASTA/Q files. We developed pyfastx as a

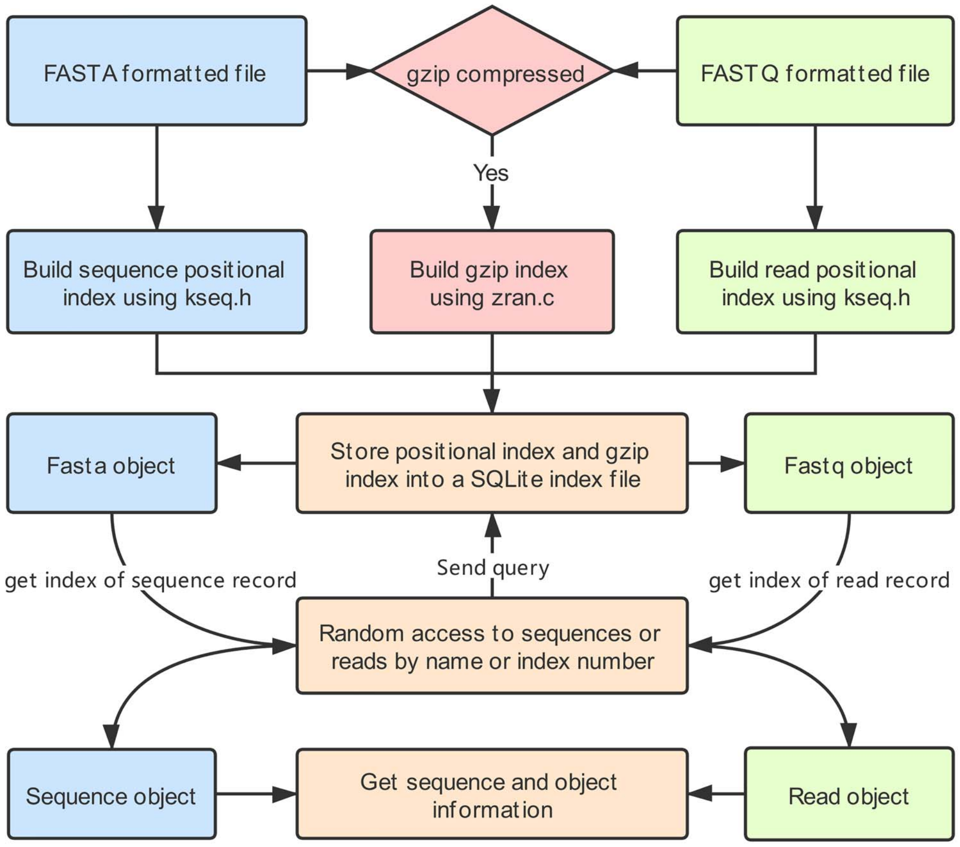
FASTA formatted file
gzip compressed
FASTQ formatted file
Yes
Build sequence positional index using kseq.h
Build gzip index using zran.c
Build read positional index using kseq.h
Fasta object
Store positional index and gzip index into a SQLite index file
Fastq object
get index of sequence record
Send query
get index of read record
Random access to sequences or reads by name or index number
Sequence object
Get sequence and object information
Read object

Figure 1. The overview of pyfastx workflow.

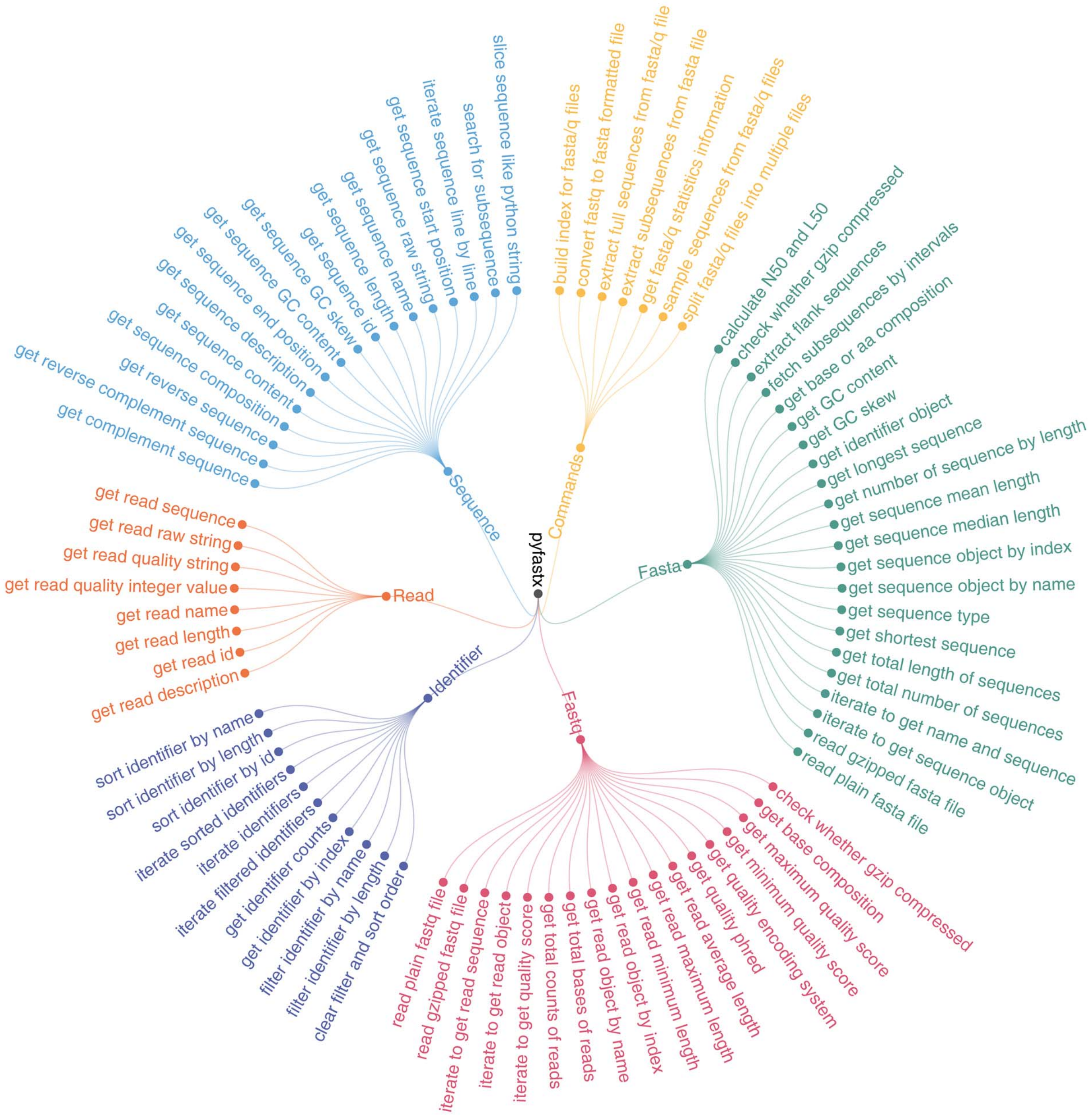
pyfastx
Sequence
slice sequence like python string
search for subsequence
iterate sequence line by line
get sequence start position
get sequence raw string
get sequence name
get sequence length
get sequence id
get sequence GC skew
get sequence GC content
get sequence end position
get sequence description
get sequence content
get sequence composition
get reverse sequence
get reverse complement sequence
get complement sequence
Commands
build index for fasta/q files
convert fastq to fasta formatted file
extract full sequences from fasta/q file
extract subsequences from fasta file
get fasta/q statistics information
sample sequences from fasta/q files
split fasta/q files into multiple files
Fasta
calculate N50 and L50
check whether gzip compressed
extract flank sequences
fetch subsequences by intervals
get base or aa composition
get GC content
get GC skew
get identifier object
get longest sequence
get number of sequence by length
get sequence mean length
get sequence median length
get sequence object by index
get sequence object by name
get sequence type
get shortest sequence
get total length of sequences
get total number of sequences
iterate to get name and sequence
iterate to get sequence object
read gzipped fasta file
read plain fasta file
Read
get read sequence
get read raw string
get read quality string
get read quality integer value
get read name
get read length
get read id
get read description
Identifier
sort identifier by name
sort identifier by length
sort identifier by id
iterate sorted identifiers
iterate identifiers
iterate filtered identifiers
get identifier counts
get identifier by index
filter identifier by name
filter identifier by length
clear filter and sort order
Fastq
read plain fastq file
read gzipped fastq file
iterate to get read sequence
iterate to get read object
iterate to get quality score
get total counts of reads
get total bases of reads
get read object by name
get read object by index
get read minimum length
get read maximum length
get read average length
get quality phred
get quality encoding system
get minimum quality score
get maximum quality score
get base composition
check whether gzip compressed

Figure 2. Pyfastx structure, features and functions.

user-friendly Python package due to the increasing popularity of Python as an ideal language for developing bioinformatics applications [12]. Additionally, pyfastx is designed to have excellent compatibility and allows parsing of non-standard FASTA files with different line lengths in an individual sequence. Pyfastx also supplies command-line tools for users to extract subsequences, split FASTA/Q files and randomly sample sequences.

Materials and methods

Index design

An index file containing the start offset of each sequence should be built to enable random access to a sequence. Thus, the desired sequence can be quickly found by seeking the predefined offset. The FASTA and FASTQ index files generated by pyfastx contain nine and six columns, respectively, and these columns in both files are distinctly different from columns in index files created by other tools (Supplementary Figure S1). In general, the line length of a sequence should also be included in the index file to facilitate the calculation of the start offset of the subsequence. The biopython index file does not include line length information and therefore has no capability to retrieve the subsequence in a memory efficient way. Whereas the index file of pyfastx holds more information, making it more compatible to deal with non-standard FASTA formatted files and more flexible when extracting sequences and description information.

Implementation

Pyfastx was implemented in C language and developed as a Python package depending on the zlib library (https://www.zli b.net) and SQLite3 database (https://www.sqlite.org/index.html). The workflow of pyfastx is shown in Figure 1. Pyfastx adopted kseq.h extracted from the prevalent klib library (https://github.com/attractivechaos/klib) to build positional index. The index of each sequence was stored in an SQLite database file with .fxi extension rather than a flat text file to avoid loading index into memory when the sequence file is repeatedly opened. To accelerate access to sequence or read information stored in the database file, a unique search index was created on sequence or read name column that might substantially increase the index file size. If the FASTA/Q files are gzip compressed, pyfastx will utilize the zran.c module from the indexed_gzip project (https://github.com/pauldmccarthy/indexed_gzip) to build an index of seek points, allowing random reading of sequences in a gzip compressed file.

Benchmark

We evaluated the performance of pyfastx for index building, random access and sequence iteration by monitoring the running time and memory usage. We used pyfastx v0.7.0, biopython v1.74, bioperl v1.7.7, samtools v1.9, pysam v0.15.3, pyfasta v0.5.2, pyfaidx v0.5.8 and seqkit v0.13.2 for performance comparison. We downloaded 15 genome FASTA files from the NCBI assembly repository with total bases varying from 12 MB to 32 GB and sequence counts ranging from 6 to 5,449,423 (Supplementary Table S1). We also downloaded 5 FASTQ files from the NGDC database [13] with file size ranging from 3.6 to 95.52 GB and read counts ranging from 15.82 million to 392.21 million (Supplementary Table S2). Benchmark tests were performed on a Centos server with Intel(R) Xeon(R) CPU E5–2620 v4 @ 2.10GHz, 64GB RAM and Python 3.6.6. More detailed information about the server is listed in Supplementary Table S3. The running time and peak memory usage for each tool were measured by using linux built-in 'time' command located under/usr/bin folder with -f '%e %M' option. Each test was executed three times, and the average elapsed time and peak memory were calculated as the final benchmarking result. In addition, we used all tools to build the index for the UniParc database, a large FASTA file downloaded from UniProt (https://www.uniprot.org/downloads) with a size of 106 GB, containing more than 305 million sequences. All the scripts used for performing the benchmark are freely available at https://github.com/lmdu/pyfastx/tree/master/benchmark.

Results and discussion

Structure and features

Pyfastx has five objects that were implemented in pure C language, these being Fasta, Fastq, Sequence, Read and Identifier. The functions of these objects are depicted in Figure 2. The Fasta and Fastq objects are the main components responsible for building index and random access to sequences from plain or gzipped FASTA/Q files. These two objects behave like Python list and dict, allowing users to obtain a Sequence and Read object by index or name. The Sequence object behaves like a Python string that can be sliced to obtain subsequences and holds substantial useful information such as name, description, GC content, nucleotide sequence and reverse complement sequence. The Read object allows users to obtain read sequences and quality scores. Both Sequence and Read objects allow users to extract the raw string of a sequence as it appears in the FASTA/Q file. The Identifier object facilitates users to filter and sort sequences

Table 1. Comparison of pyfastx with other tools in the aspect of features and functions

Tools	Language	URL	Random access	Format support			Command line tools			
				FASTA	FASTQ	Gzip	File split	Sequence extract	Sequence sample	FASTQ to FASTA
pyfasta	Python	https://github.com/brentp/pyfasta	Yes	Yes	No	No	Yes	Yes	No	No
pyfaidx	Python	https://github.com/mdshw5/pyfaidx	Yes	Yes	No	No	Yes	Yes	Yes	No
pysam	Python, C	https://github.com/pysam-developers/pysam	Yes	Yes	Yes[a]	No	No	No	No	No
biopython	Python	https://biopython.org/	Yes	Yes	Yes	No	No	No	No	No
pyfastx	Python, C	https://github.com/lmdu/pyfastx	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
samtools	C	http://www.htslib.org/	Yes	Yes	Yes	No	No	Yes	No	No
seqkit	GO	https://bioinf.shenwei.me/seqkit/	Yes	Yes	Yes[a]	Yes[b]	Yes	Yes	Yes	Yes
HTSeq	Python	https://github.com/simon-anders/htseq	No	Yes	Yes	No	No	No	No	No
seqmagick	Python	https://fhcrc.github.io/seqmagick/	No	Yes	Yes	No	No	No	No	Yes
fasta_utilities	Perl	https://github.com/jimhester/fasta_utilities	No	Yes	Yes	No	Yes	Yes	Yes	Yes
FASTX-Toolkit	Perl	http://hannonlab.cshl.edu/fastx_toolkit/	No	Yes	Yes	No	Yes	No	No	Yes
seqtk	C	https://github.com/lh3/seqtk	No	Yes	Yes	Yes[b]	No	Yes	Yes	Yes
fqtool	C	https://github.com/alastair-droop/fqtools	No	No	Yes	Yes[b]	No	Yes	No	Yes
Bioperl	Perl	https://bioperl.org/	Yes	Yes	Yes	No	No	No	No	No

[a]The tool can parse the FASTQ file, but does not allow random access to reads.
[b]The tool can read sequences in the gzip compressed file from start to end, but does not allow random access to sequences from the gzip compressed file.

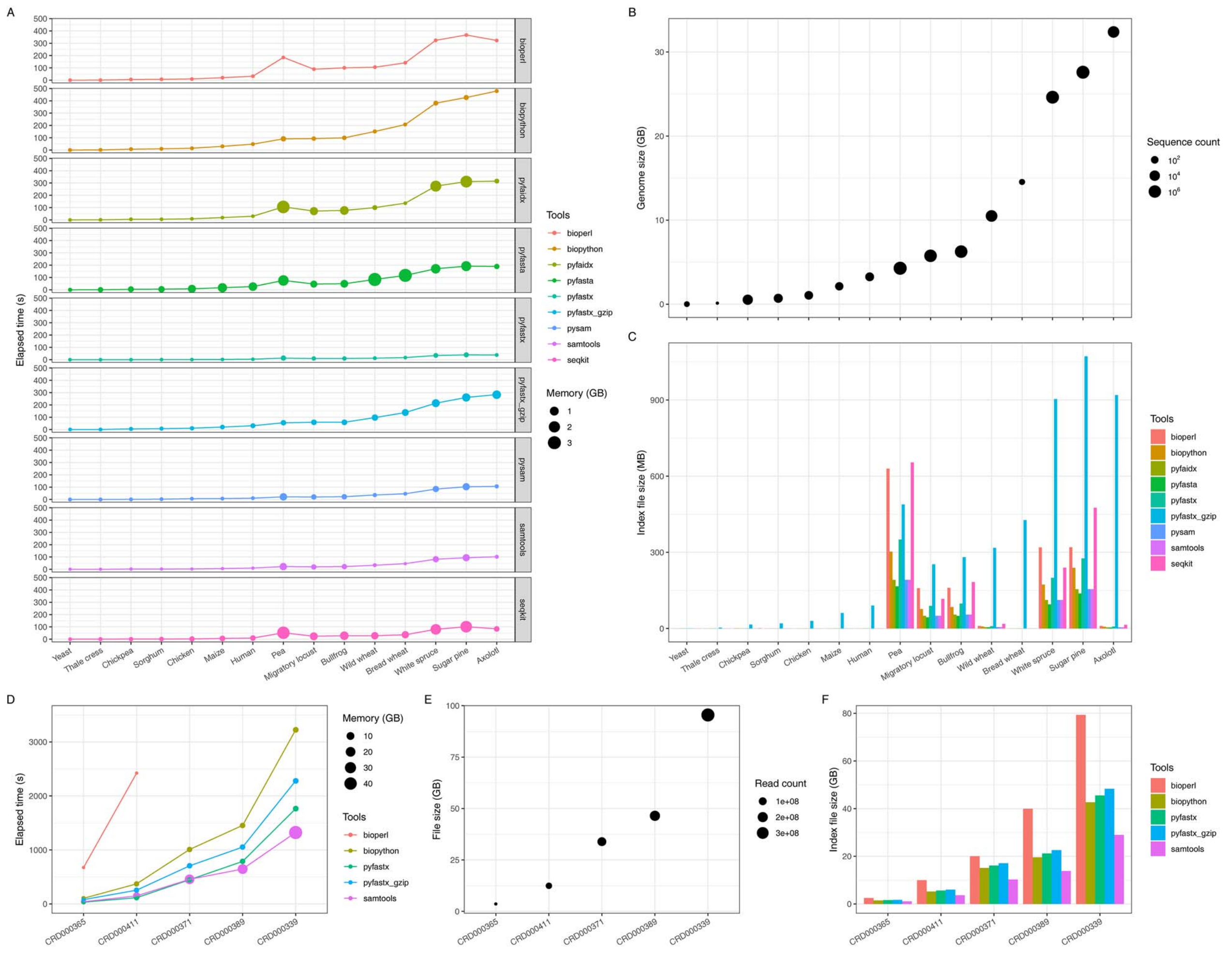

Figure 3. Performance comparison of index building. (A) The elapsed time and peak memory of each tool for building FASTA index. (B) The genome size and sequence count of tested FASTA files. (C) The size of FASTA index files for each tool. (D) The elapsed time and peak memory of each tool for building FASTQ index. The missing dots indicate that its elapsed time > 3500 seconds. (E) The file size and read count of tested FASTQ files. (F) The size of FASTQ index files for each tool.

in the FASTA file by name and length. Additionally, command-line tools released with pyfastx assist users to split FASTA/Q files, sample sequences from FASTA/Q files, extract subsequences and convert FASTQ to FASTA format.

We have compared the features of pyfastx to other tools. The results indicated that pyfastx is sufficiently advantageous to replace existing tools when dealing with sequence files (Table 1). Pyfastx provides support for random access to sequences from both FASTA and FASTQ formats. Despite pysam and seqkit being able to read sequences from gzip compressed FASTA/Q files, they have no capability of building index for random access. As shown in Table 1, pyfastx is the only tool that supports random access to sequences from gzip compressed files. Moreover, it offers commonly used command-line tools for users to manipulate plain and gzipped FASTA/Q files.

Index building assessment

Figure 3 and Supplementary Table S4 illustrate the performance of tested tools in terms of peak memory, elapsed time and index file size for all datasets. Although all the tested tools exhibited similar performance for small genomes, noticeable differences have been observed between tested tools for large genomes (Figure 3A and B). Pyfastx performed extremely well for time and memory usage, and was even slightly faster than pysam and samtools. Bioperl and biopython consumed the most time, whereas pyfaidx, pyfasta and seqkit consumed the most amount of memory. Pyfastx and biopython employed SQLite3 to store index while bioperl used Berkeley DB. The generated index was directly written into the database file instead of retaining it in memory, which vastly reduced memory consumption. Despite more time and memory consumed to build the index for the gzip compressed version of genomes, pyfastx was still faster than bioperl and biopython. Similarly, the sizes of generated index files were also remarkably different when these tools were used to process large genomes (Figure 3B and C). The sizes of index files generated by pyfastx were larger than samtools, but smaller than bioperl and seqkit, whereas, the sizes of index files generated by pyfastx for compressed genomes were much larger than those for the uncompressed ones (Figure 3C). The index file size was closely related to the number of records and length of name in the sequence file, and for the compressed file, it was also related to file size due to the requirement of storing a seek point index. Nevertheless, the disk space usage of the pyfastx index file is still acceptable to all modern desktop computers (e.g. maximum size ∼1 GB).

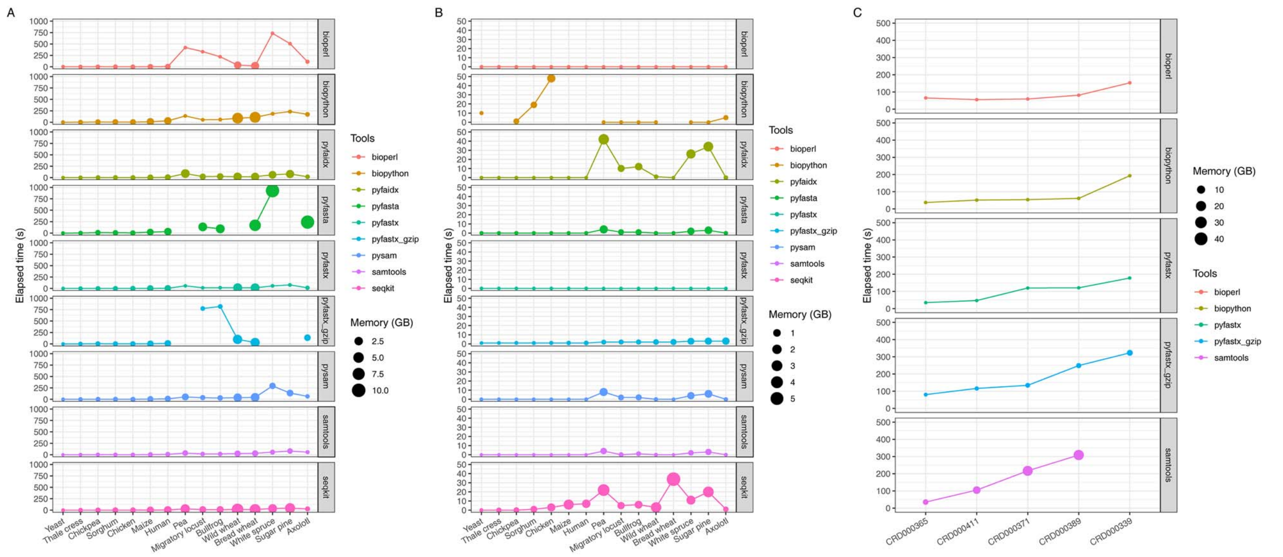

Figure 4. Performance comparison of random access. (A) The elapsed time and peak memory of each tool for random access to 30% of sequences from FASTA files. The missing dots indicate that its elapsed time > 1000 seconds. (B) The elapsed time and peak memory of each tool for random access to 1000 subsequences with length of 1 Kb from FASTA files. The missing dots indicate that its elapsed time > 50 seconds. (C) The elapsed time and peak memory of each tool for random access to 10 000 reads from FASTQ files. The missing dots indicate that its elapsed time >500 seconds.

All tools were then employed to build index for the UniParc database file to evaluate the capacity of processing the large sequence file. Pyfastx only used <3 GB memory to successfully build the index, while it took three times longer for the compressed version (Supplementary Table S5). Pyfaidx, pyfasta and seqkit consumed almost all of the memory and were manually terminated after running for >10 hours, even though samtools consumed more than 30 GB of memory. Although bioperl and biopython were also memory efficient, pyfastx was more than 3 times faster than biopython and 14 times faster than bioperl. The index file size of pyfastx was larger than biopython and samtools, but smaller than bioperl.

Currently, only pyfastx, biopython, bioperl and samtools support random access to reads from FASTQ files. Figure 3D demonstrates the performance of building the index. Bioperl was not completely included in the figure since it consumed more than 32 hours to build the index for the largest FASTQ file (Supplementary Table S6). Pyfastx was slightly slower than samtools, but samtools consumed too much memory with a maximum of 46 GB. Although biopython was memory efficient, it took twice as long as pyfastx and was slower than pyfastx when processing compressed files. Unfortunately, the sizes of index files generated by pyfastx were much larger than samtools with a maximum ratio of 47.68% of the original file (Figure 3E and F). Additionally, the index file size of the compressed file can be larger than the size of the original file (Supplementary Table S6). However, it is worthwhile to sacrifice storage space to reduce memory usage when considering pyfastx's suitability for use on desktop computers and servers.

Random access evaluation

After building the index, all tools were applied to randomly extract 30% of sequences from all genomes to evaluate the performance of random access. As illustrated in Figure 4A and Supplementary Table S7, pyfastx and other tools gave nearly the same performance for processing smaller genomes. For larger genomes, pyfastx was comparable with samtools but had a higher performance than the other tools. However, for some gzip compressed large genomes, pyfastx consumed much more time than processing a plain format file. Due to the default use of two threads, seqkit was even faster than samtools, but it consumed much more memory (Supplementary Table S7). We also assessed the performance of retrieval of subsequences by using these tools to randomly extract 1000 subsequences of length 1 Kb from all genomes. As shown in Figure 4B and Supplementary Table S8, pyfastx performed excellently with extremely low time and memory consumption. Unexpectedly, bioperl was comparable with pyfastx while biopython consumed more time to process some genomes. Biopython cannot directly extract subsequences due to index design defect, where it has to acquire the whole specified sequence and then slice the subsequence. We also assessed the performance of retrieval of reads from FASTQ files between pyfastx, bioperl, biopython and samtools. These tools were used to randomly extract 10 000 reads from FASTQ files. The time and memory usage of samtools increased considerably with the size of FASTQ files, reaching a maximum memory usage of 47 GB for the largest FASTQ file (Supplementary Table S9). Pyfastx, bioperl and biopython were time- and memory efficient (Figure 4C). Although bioperl and biopython were slightly faster than pyfastx, pyfastx still had better performance when considering the time consumption of index building.

Sequence iteration measurement

In practice, iterating over sequence records from FASTA/Q files is a very important function for processing genomic data and has a wide range of applications. We compared the sequence iteration performance of pyfastx to other Python packages and bioperl. Pyfastx provides two traversal modes, one is with index that allows the users to access more sequence information, the other is without index and provides only the name and sequence, and quality for the FASTQ file. As shown in Figure 5A and Supplementary Table S10, the performance

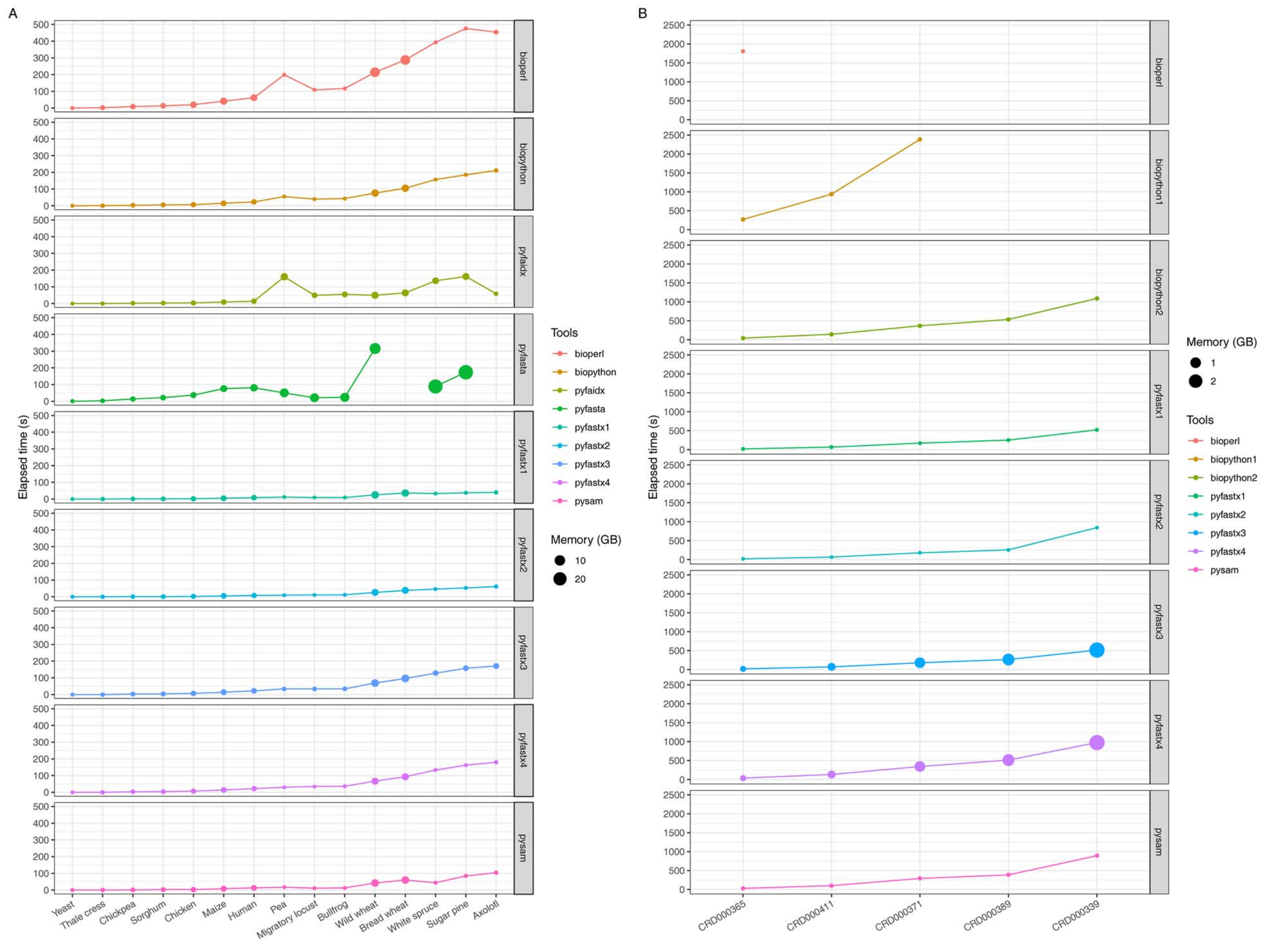

Figure 5. Performance comparison of sequence iteration. (A) The elapsed time and peak memory of each tool for iterating over sequences from FASTA files. The missing dots indicate that its elapsed time > 500 seconds. (B) The elapsed time and peak memory of each tool for iterating over reads from FASTQ files. Biopython1: iterating over reads by using SeqIO; biopython2: iterating over reads by using FastqGeneralIterator; pyfastx1: iterating over reads from plain FASTQ file with index; pyfastx2: iterating over reads from plain FASTQ file without index; pyfastx3: iterating over reads from gzip compressed FASTQ file with index; pyfastx4: iterating over reads from gzip compressed FASTQ file without index. The missing dots indicate that its elapsed time > 2500 seconds.

of these two modes were comparable and higher than the performance of the other Python packages. Although pyfastx consumed more time to traverse the gzip compressed FASTA file, it was still much faster than bioperl and comparable with biopython in processing plain FASTA files. As depicted in Figure 5B and Supplementary Table S11, it was clear that pyfastx consumed more memory to process gzip compressed FASTQ files, while its speed was faster than other tools. Biopython also provides two ways to iterate over reads from a FASTQ file, one is FastqGeneralIterator, and the other is SeqIO. Although FastqGeneralIterator was much faster than SeqIO, it was still slower than pyfastx. It is worth noting that the consumption times of pyfastx for processing plain and gzip compressed FASTQ files were very similar.

Conclusion

In this study, we have presented pyfastx, which is a time and memory efficient Python package for randomly accessing sequences from FASTA/Q files. Compared to existing tools, pyfastx exhibited the best performance for index building, sequence retrieval and sequence iterating. Pyfastx also has the capability to build index for large FASTA/Q files containing hundreds of millions of sequences with low memory consumption. To our knowledge, pyfastx is the first tool that provides support for randomly accessing sequences from gzip compressed FASTA/Q files. Although pyfastx was developed as a Python package, it also offers command-line tools for users to handle common manipulations of FASTA/Q files. We anticipate that pyfastx may greatly increase the efficiency of newly developed bioinformatics tools for sequence analysis.

Data Availability

The FASTA files and FASTQ files underlying this study were downloaded from NCBI assembly repository and NGDC database. The accession number of each dataset can be found in Supplementary Table S1 and S2. The source code of pyfastx is deposited in a Github repository (https://github.com/lmdu/pyfastx).

Key Points

- Pyfastx is the first tool that supports random access to sequences from gzip compressed FASTA/Q files.
- Pyfastx has the highest performance for building index and extracting sequences.
- Pyfastx has the capability to process extremely large FASTA/Q files containing thousands of millions of sequences with extremely low memory consumption.
- Pyfastx is a Python package and can be easily integrated into biological applications, database or genome browsers to extract subsequences.
- Pyfastx provides common command-line tools for users to manipulate FASTA/Q files.

Supplementary Data

Supplementary data are available online at *Briefings in Bioinformatics*.

Acknowledgements

We would like to greatly appreciate Heng Li for developing kseq and Paul McCarthy for developing indexed_gzip. We would also like to thank pyfastx community users for providing feature requests and reporting bugs.

Funding

The Sichuan Science and Technology Program (2020YJ0490) and Sichuan Association for Science and Technology (2018RCTJ).

References

1. Kucherov G. Evolution of biosequence search algorithms: a brief survey. *Bioinformatics* 2019;**35**(19):3547–52.
2. Zhang H. Overview of sequence data formats. *Methods Mol Biol* 2016;**1418**:3–17.
3. Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A* 1988;**85**: 2444–8.
4. Cock PJ, Fields CJ, Goto N, *et al*. The sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res* 2010;**38**(6):1767–71.
5. Anders S, Pyl PT, Huber W. HTSeq–a python framework to work with high-throughput sequencing data. *Bioinformatics* 2015;**31**(2):166–9.
6. Droop AP. Fqtools: an efficient software suite for modern FASTQ file manipulation. *Bioinformatics* 2016;**32**(12):1883–4.
7. Cock PJ, Antao T, Chang JT, *et al*. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics* 2009;**25**(11):1422–3.
8. Stajich JE, Block D, Boulez K, *et al*. The Bioperl toolkit: Perl modules for the life sciences. *Genome Res* 2002;**12**(10):1611–8.
9. Li H, Handsaker B, Wysoker A, *et al*. The sequence alignment/map format and SAMtools. *Bioinformatics* 2009;**25**(16):2078–9.
10. Shen W, Le S, Li Y, *et al*. SeqKit: a cross-platform and ultrafast toolkit for FASTA/Q file manipulation. *PLoS One* 2016;**11**(10):e0163962.
11. Shirley MD, Ma Z, Pedersen BS, *et al*. Efficient "pythonic" access to FASTA files using pyfaidx. *Peer J Prepr* 2015;**3**: e970v1.
12. Ekmekci B, McAnany CE, Mura C. An introduction to programming for bioscientists: a python-based primer. *PLoS Comput Biol* 2016;**12**(6):e1004867.
13. National Genomics Data Center Members and Partners. Database resources of the National Genomics Data Center in 2020. *Nucleic Acids Res* 2020;**48**(D1):D24–33.