Efficient "pythonic" access to FASTA files using pyfaidx

Matthew Shirley[1,2], Zhaorong Ma[3], Brent Pedersen[4], and Sarah Wheelan[1,2]

[1]**Department of Oncology, Johns Hopkins University School of Medicine, USA**
[2]**Center for Computational Genomics, Johns Hopkins University, USA**
[3]**SoftGenetics, LLC. 100 Oakwood Ave, Suite 350 State College, PA 16803, USA**
[4]**Eccles Institute of Human Genetics, University of Utah School Of Medicine, USA**

ABSTRACT

The `pyfaidx` Python module provides memory and time-efficient indexing, subsetting, and in-place modification of subsequences of FASTA files. `pyfaidx` provides Python classes that expose a dictionary interface where sequences from an indexed FASTA can be accessed by their header name and then sliced by position without reading the full file into memory. `pyfaidx` includes an extensive test suite to ensure correct and reproducible behavior. A command-line program (`faidx`) is also provided as an alternative interface, with significant enhancements to functionality, while maintaining full index file compatibility with `samtools`. The `pyfaidx` module is installable from PyPI (`https://pypi.python.org/pypi/pyfaidx`), and development versions can be found at Github (`https://github.com/mdshw5/pyfaidx`).

Keywords: fasta, python, api, bioinformatics

Please send correspondence to mdshw5@gmail.com
Software issues should be submitted to `http://github.com/mdshw5/pyfaidx/issues`

1 INTRODUCTION

The FASTA file specification was originally developed as the input format for the `FASTA` sequence alignment software (Pearson and Lipman, 1988). Subsequently, the FASTA file format has become a ubiquitous exchange format for single-letter alphabet biological sequences such as DNA, RNA, and protein. Commonly FASTA files contain multiple sequences, with each sequence having a uniform line length before wrapping to the beginning of a new line, and with sequence identifiers separating each sequence. Genome assemblies are commonly distributed as FASTA files, with each sequence entry representing either a contiguous assembled scaffold, or an entire chromosome.

Manipulation of sequences stored in a FASTA file can become problematic when the in-memory size of a sequence exceeds the physical memory available to a program. In such cases, it is common to break a sequence into smaller chunks and then apply a function to each of the smaller chunks in succession. Because many FASTA files are line-wrapped with a consistent number of characters per line, a line can provide a natural chunk size for reading a large sequence. While line-based iteration over a FASTA sequence can be memory efficient, many times random access to sub-sequences is desirable.

For the common case of accessing specific sub-sequences in a line-wrapped FASTA file, `samtools` (Li et al., 2009) established an indexing scheme that relies on consistent line-lengths within individual sequence entries, consistent ASCII line terminator characters (Gorn et al., 1963) and removal of trailing white-space and blank lines. When these conditions are met, an index file can be generated which maps sequence coordinates to byte offsets in the FASTA file, facilitating memory and time-efficient retrieval of sub-sequences without reading the entire FASTA file from start to end. The `pyfaidx` module provides Python interfaces to build and utilize this index using a "dictionary" interface and memory-efficient sequence slicing that follows Python conventions, as well as a stand-alone program (`faidx`) suitable for use by non-programmers. This allows Python users to quickly leverage well-tested, highly compatible, and efficient code that would otherwise be duplicated in many independent projects.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.970v1 | CC-BY 4.0 Open Access | rec: 8 Apr 2015, publ: 8 Apr 2015

2 METHODS

2.1 Installation

The `pyfaidx` module supports Python 2, Python 3 and PyPy. Installation from the Python Package Index (PyPI) is supported via `pip install pyfaidx`.

2.2 Fasta indexing

The FASTA index file (.fai extension) consists of five columns with rows containing values for every sequence in the FASTA file:

- sequence definition lines
- sequence length in characters excluding newlines
- the byte offset at start of the sequence
- the wrapped line length both with and without newlines

Both the `faidx` command-line utility and the `Fasta` class automatically generate this index file if it does not exist. Files with non-unique definition lines will raise an error. It is important to note that pyfaidx generates the FASTA index as a data stream, and therefore can index large FASTA files such as the NCBI non-redundant nucleotide database (Pruitt et al., 2005) without holding the entire index (or of course, the sequences) in memory, a shortcoming of other implementations such as texttttsamtools (Li et al., 2009). FASTA index files generated by `samtools`, the `faidx` utility, and the `pyfaidx` module are compatible and interchangeable.

2.3 Sequence retrieval

The `Fasta` class provides access to indexed FASTA files with an interface that acts as a Python "dictionary" object. FASTA definition lines are used as dictionary keys, and for any key the sequence is returned as a `Sequence` object. `Sequence` objects have attributes for the string representation of the requested sequence, along with a header specifying 1-based start and end genomic coordinates (Figure 1). Note that slicing indices are 0-based and that negative indices are relative to the end of the sequence, which is consistent with the indexing behavior of Python sequence types.

Figure 1. Dictionary lookup and string slicing methods using the `Fasta` class. Input lines are preceded with `>>>`.

```
>>> from pyfaidx import Fasta
>>> hg38 = Fasta('hg38.fa')
>>> hg38['chr1'][10000:10010]
>chr1:10001-10010
TAACCCTAAC
>>> hg38['chr1'][-1000000:-999990]
>chr1:247956423-247956432
GTGGGCTCTC
```

Several existing methods for random FASTA access are available, and fall into three categories:

1. `biopython` parses and reads the entire file into memory
2. `pyfasta` copies the FASTA file, removes sequence identifiers, and creates a proprietary index
3. `samtools`, `pysam`, and `pyfaidx` all generate a compatible index of the unmodified FASTA file

Of the preceding methods, `biopython`, `pyfasta` and `pyfaidx` are implemented in pure Python and require no external dependencies. `Samtools` and `pysam` require a `C` compiler and interface with `htslib`, and `pyfasta` operates most efficiently using the `Numpy` backend, which is also implemented in `C`. A comparison of these methods (Table 1) demonstrates that `pyfaidx` is approximately as fast as

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.970v1 | CC-BY 4.0 Open Access | rec: 8 Apr 2015, publ: 8 Apr 2015

pyfasta and much faster than calling htslib using pysam. Importantly, pyfaidx is the fastest method that leverages the memory-efficient and samtools compatible "*.fai" indexing scheme. Memory usage for pyfasta and biopython were significantly higher than pyfaidx.

Software	Init(index) (seconds)	Fetch 1kb sequence (microseconds)	Memory (max MB)
pyfaidx.Fasta (seek)	30.20	90.81	0.190
pyfaidx.Faidx (seek)	28.92	64.15	0.146
pyfasta.Fasta (numpy)	29.02	43.59	37.239
pyfasta.Fasta (seek)	27.18	160.94	37.239
Bio.SeqIO	29.35	4.58	2517.515
samtools faidx	20.08	168.11	NA
pysam.faidx	14.85	411.49	NA

Table 1. Benchmark of random 1000bp accesses to FASTA sub-sequences. Benchmarking was performed in triplicate on a 2.4GHz Haswell Core i5 with a solid state disk drive using Python 3.4, pyfaidx v0.3.9, biopython v1.64, numpy v1.9.1, pyfasta v0.5.2, and pysam v0.8.1. Average timings are reported. Memory usage is reported using tracemalloc, and usage for samtools and pysam was omitted due to difficulty profiling Python C-extensions. Benchmarks were performed using https://github.com/mdshw5/pyfaidx/blob/v0.3.9/scripts/benchmark.py

2.4 In-place sequence masking

"Masking" a FASTA file is a common step in many bioinformatics pipelines, used to indicate positions that are flagged for different treatment in downstream analysis. Certain characters or ranges of characters in FASTA file sequences are either replaced with a distinct character, or the case of the character is inverted. Existing FASTA masking tools (Quinlan and Hall, 2010) read a file from start to end, perform sequence masking, and then write the masked FASTA file to disk. This usually requires a list of regions sorted in the same order as the FASTA file, or that the regions are all stored in program memory. This approach is particularly inefficient for large FASTA files requiring a relatively small amount of masking. For this reason pyfaidx provides a "mutable" Fasta object for in-place modification of a FASTA file. The faidx utility also provides in-place masking capabilities that emulate the capabilities of bedtools maskfasta. In benchmarks against bedtools masking regions of low complexity (Li, 2014) faidx uses 3.2X less memory and runs in equal time with less CPU usage (Table 2).

Software	Memory (MB)	CPU (%)	Time (seconds)
bedtools	616	98	86
pyfaidx	194	70	88

Table 2. Benchmarking was performed in triplicate on a 2.4GHz Haswell Core i5 with a solid state disk drive using Python 3.4, pyfaidx v0.3.9, and bedtools v2.22.0.

2.5 Splitting FASTA to separate files by region

The faidx --split-files flag creates new output files for each region specified in either a bed file, or UCSC format "chr:start-end" (Figure 2).

2.6 Sequence retrieval using definition line fields

One common naming scheme for FASTA definition lines is to include information about the sequence such as accession number and a long description, in addition to a short identifier, such as a gene name. NCBI follows a convention (Madden, 2013) of separating fields in FASTA definition lines using | (pronounced "pipe"). Bioinformatics tools commonly use string comparison to determine if a feature maps to a reference sequence selected from a FASTA file or similar format. This requires pattern matching over special characters which is both inconvenient and error-prone. The faidx utility can split definition lines on a delimiter for retrieval of sequences by fields from such files (Figure 3). This may be used to quickly relabel sequences for downstream tools, or for sequence lookup using only a definition line field.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.970v1 | CC-BY 4.0 Open Access | rec: 8 Apr 2015, publ: 8 Apr 2015

Figure 2. Splitting FASTA sequences to individual files using `faidx` program. Input lines are prefixed with '$'.

```
$ faidx --split-files hg38.fa
$ ls chr*
chr10.fa  chr15.fa  chr1.fa   chr3.fa
chr8.fa   chr11.fa  chr16.fa  chr20.fa
...

$ faidx --split-files hg38.fa chr1:100-1000
$ ls chr*
chr1.100.1000.fa

$ faidx --split-files hg38.fa --bed regions.bed
$ ls chr*
chr8.50000.55000  chrX:800000-1000000
```

Figure 3. Retrieval of zebrafish protein sequence accession via the shell. Input lines are prefixed with '$'. Ellipses (...) indicate lines truncated for display purposes.

```
$ head -n3 zebrafish.fasta
>gi|54400524|ref|NP_001006011.1| pleckstrin...
MLESGVLKEGALEKRSDGLLQLWKKKRCVLTEDGLVLHPHKHH...
FTVVMSEGREIDFRCLQDEGWNAEITLRMVQYKNRQAILAVKS...
$ faidx -d '|' zebrafish.fasta NP_001006011.1
>NP_001006011.1
MLESGVLKEGALEKRSDGLLQLWKKKRCVLTEDGLVLHPHKHH...
FTVVMSEGREIDFRCLQDEGWNAEITLRMVQYKNRQAILAVKS...
```

3 CONCLUSIONS

The `pyfaidx` module provides a lightweight, easy to install, familiar and intuitive interface to FASTA files. Indexing, retrieval, and in-place file modification are implemented in a time and memory-efficient manner. `pyfaidx` is tested and supported under Linux, Mac OS, and Windows using Python 2.6, 2.7, 3.2, 3.3, 3.4, and PyPy, and is installable via `pip install pyfaidx`.

REFERENCES

Gorn, S., Bemer, R. W., and Green, J. (1963). American standard code for information interchange. *Communications of the ACM*, 6(8):422–426.

Li, H. (2014). Toward better understanding of artifacts in variant calling from high-coverage samples. *Bioinformatics*, 30(20):2843–2851.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079.

Madden, T. (2013). The BLAST sequence analysis tool.

Pearson, W. R. and Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448.

Pruitt, K., Tatusova, T., and Maglott, D. (2005). NCBI Reference Sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Res*, 33:D501–4.

Quinlan, A. R. and Hall, I. M. (2010). BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.970v1 | CC-BY 4.0 Open Access | rec: 8 Apr 2015, publ: 8 Apr 2015