Google Summer of Code 2014 Proposal:

Addition of a Lazy Loading Sequence Parser to Biopython's SeqIO Package

Evan Parker

Introduction:

One of the core motivations for the production and maintenance of Biopython, and indeed all Open Bioinformatics Foundation projects, is the reduction of code duplication by scientists performing common tasks. It follows that the sequence parsing module SeqIO is an important component of Biopython since one of the most often repeated tasks is parsing flat file sequence data. While most use-cases have adequate performance using a straightforward implementation of sequence parsing, a border case that is not currently addressed is the efficient parsing of large sequences. The current parsers in SeqIO load sequences either as a generator in the case of SeqIO.parse(), or as a custom dictionary in the case of SeqIO.index(). For the largest individual sequences this is not an efficient strategy since parsing the entire sequence may entail loading millions of base pairs into memory. Many of these may not be required for the task at hand.

A potentially better strategy when parsing the largest sequences would be to lazily load only the portion of the file required. Using slice notation, a user will be able to request only the segment of the sequence required for the task. A lazy loading parser would walk the line between increased transaction overhead and decreased memory requirements but in many cases the new parser would improve overall performance. Especially in the use case of large sequences I am confident that overall performance will increase as unnecessary parsing and storage can be eliminated or the necessary parsing can be streamlined into the a comprehensive analysis pipeline. For single threaded processes lazy loading sequence data can mean faster access to intermediate results and for parallel processes lazy loading can lead to even greater performance increases than possible currently.

Project Goals:

- Implement location aware parsers for sequential sequence file formats that lazily load called sequence portions. Parsers should cover, at minimum, FASTA, GenBank, Embl, Swiss, UniProt-XML, and tab formats.
- Implement a lazy loading SeqRecord proxy class that can be returned by the SeqIO function; these objects will accept slice notation to invoke the lazy loading parsers and return the requested sequences.
- Profile the new code to identify trouble spots and to make explicit the use cases where lazy loading improves performance.
- Document the new functionality.

Implementation:

The implementation of this functionality will reuse as much code as possible from the format-specific parsers included in the SeqIO package and elsewhere in Biopython. As a necessary step to implement lazy loading

parsers, a proxy class, hereon referred to as SeqRecProxy, will be used to initialize the actual lazy loading and grant access to SeqRecord objects. Figures 2 and 3 indicate how a lazy lading SeqRecord proxy would alter the current workflow.

The lazy loading of FASTA data is an illustrative example of the most trivial implementation of parsing a flat file with sequential unannotated sequences. The implementation of FASTA parsing will be comprised of two new generators in SeqIO/FastaIO.py. The first will yields SeqRecProxy objects, the second will be a file seeking simple parser that used by the SeqRecProxy. The added parsing functionality will be accessed with a Boolean "lazy=True" keyword argument in either index() or parse(). The *lazy* kwarg will default to False preventing the added functionality from disrupting any existing programs.

SeqIO.parse() and SeqIO.index(), when called with "lazy=True", will return SeqRecProxy objects. On instantiation of the SeqRecProxy the lazy loading parser will read the first two lines of the first FASTA entry. The first line will be completely parsed to form the 'id', 'name', and 'description' used in all derived SeqRecord objects. The second line will be parsed to determine the number of residues encoded per line; while the recommended length of FASTA sequence lines is 80 characters, implementations differ depending on source so recognition of this number will assist in seeking to the correct file location. The column width is then stored as an instance attribute of the SeqRecProxy. Finally, when passed slice notation, the lazy loading parser's __getitem__() function will seek to the correct file line and sequentially extract the requested residues.

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
```

Figure 1: this partial FASTA record uses the standard line length of 80 characters for sequence information

Iterating heavily annotated formats like GenBank or SwissProt flat files will be a more difficult task. For most file formats one cannot assume that annotations are 100% ordered, thus true-lazy loading of annotations will suffer from low performance as annotation data is reparsed multiple times. Detailed pre-parsing and indexing would make sense if millions of reads are expected, but I expect that in most use cases, the process of indexing would actually detract from overall performance. Lacking format-enforced order in the annotations, a tradeoff must be made between pre-parsing the annotations to deliver from memory, or re-parsing the annotations on every call. My proposed solution would be for SeqRecProxy to have an *@property* decorated annotations() method that would parse all annotations. Once annotations are parsed, SeqRecProxy will include appropriate annotations with slice retrieved SeqRecords.

Timeline

It should be noted that between the community bonding period and June 22, I will continue to have obligations due to my graduate studies and preparation for the American Society of Mass Spectrometry conference where I will be giving a talk. After this obligation, my time will be 100% dedicated to the GSOC project.

Community bonding:	Read Biopython documentation, prepare development environment and connect with my mentor. Work on a detailed specification along with the core class prototypes and tests that will need to be passed (working at 50% capacity due to ongoing graduate work)
May 19 -25	Write the core SeqRecProxy (working at 50% capacity due to ongoing graduate work)
May 26- 1	Write the lazy loading FASTA parser (working at 50% capacity due to ongoing graduate work)
June 2 – 8	Text and profile new code (working at 50% capacity due to ongoing graduate work)
June 9 - 15	Write parsers for other un-annotated file formats (working at 50% capacity due to ongoing graduate work)
June 16 - 22	Continue working on un-annotated file formats, write additional unit tests (working at 10% capacity due to American Society of Mass Spectrometry conference, this conference will mark the end of my pre-summer obligations).
June 23 - 29	Add lazy loading of sequence information to the Bio.GenBank.Scanner parser.
June 30- July 6	Add lazy loading of annotation information to Bio.Genbank.Scanner. Write unit tests.
July 7-13	Spend time profiling GenBank lazy loaders, optimize any slow code. Use lessons learned here while writing lazy loaders for other annotated formats.
July 14-20	Add lazy loading to Swiss formats
July 21-27	Add lazy loading to UniProt-XML formats
July 28 – Aug 3	Write additional unit tests for annotated formats from Swiss and Uniprot-XML
Aug 4- Aug 11	Continue cleaning up code, write documentation.

About me:

I am a third year PhD student in the Chemistry department of UC Davis. Programming in Python was a hobby during my undergraduate education and it has become a major portion of my graduate work. I work in the lab of Carlito Lebrilla and I write programs used internally for the interpretation of mass spectrometric data. I am currently working on improved scoring algorithms for glycopeptides fragmented by collision induced dissociation. In one of my recent projects I have assisted in the production of a peptidomics workflow by writing a proteolytic enzyme analysis script, this code was made public in preparation of publication and can be seen on my GitHub page[1]. I will be presenting my work on encoding binary information with a nontoxic polymer marker at the upcoming American Society of Mass Spectrometry meeting in June.

[1] https://github.com/eparker05/

Figure 2: The current API accepts a file handle along with the format and returns an iterable generator or a dictionary-like object. This can be used directly to access the SeqRecord. Subsequences retrieved via slice notation will also be SeqRecord types.

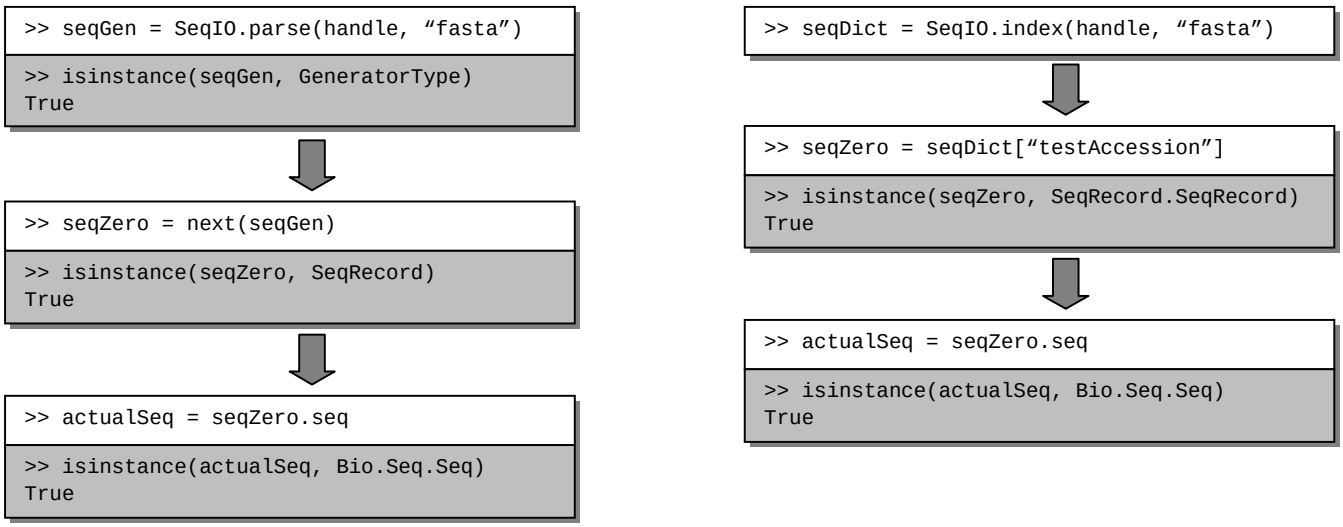

Figure 3: In the proposed project, lazy loading intermediates are returned by SeqIO parse and index functions. This intermediate can be used to access lazily loaded subsequences of the larger underlying sequence.