Pybedtools: a flexible Python library for manipulating genomic datasets and annotations

Ryan K. Dale¹*, Brent S. Pedersen² and Aaron R. Quinlan³ *
¹Laboratory of Cellular and Developmental Biology, National Institute of Diabetes and Digestive and Kidney Diseases, National Institutes of Health, Bethesda, MD 20892, ²Department of Medicine, University of Colorado, Denver, Anschutz Medical Campus, Aurora, CO 80045 and ³Department of Public Health Sciences, Center for Public Health Genomics, University of Virginia, Charlottesville, VA 22908, USA

Associate Editor: John Quackenbush

ABSTRACT

Summary: pybedtools is a flexible Python software library for manipulating and exploring genomic datasets in many common formats. It provides an intuitive Python interface that extends upon the popular BEDTools genome arithmetic utilities by providing a powerful and consistent framework for reproducible research. Here we introduce pybedtools, which extends the BEDTools (Quinlan and Hall, 2010) genome arithmetic utilities by providing a powerful interface combining the benefits of Python scripting and the BEDTools libraries. Using a simple syntax, it allows researchers to analyze datasets in BED (Kent et al., 2002), VCF (Danecek et al., 2011), GFF, BEDGRAPH (Kent et al., 2002) and SAM/BAM (Li et al., 2009) formats without the need for format conversion.

2 APPROACH

The pybedtools library allows one to manipulate datasets at both the file and individual feature level using the BedTool and Interval classes, respectively. It integrates high-level BEDTools programs through the Python subprocess module, and lower level BEDTools functionality by exposing a subset of BEDTools’ libraries. At the core of pybedtools is the BedTool class. Typically, a BedTool is initially created with a file name. BEDTools programs are then accessed as methods of BedTool objects (e.g. BedTool.intersect for the BEDTools program intersectBed) with arguments identical to the user’s installed version of BEDTools. However, in addition to passing filenames as in typical BEDTools command line usage, one may also pass collections of Interval objects which can be manipulated in Python on a feature-by-feature basis. Furthermore, BedTool methods return new BedTool instances, allowing users to chain many operations together in a fashion similar to the UNIX command line.

The pybedtools package provides a standardized interface to individual features in diverse genomics datasets, thus allowing one to iterate through datasets while accessing chromosome, start and stop coordinates with identical syntax, regardless of the underlying file format. This abstraction is made possible via Cython (http://cython.org, last accessed Aug 2011) which exposes the BEDTools file manipulation, feature parsing and overlap detection methods return new Interval objects (e.g. BedTool.intersect) with arguments identical to the user’s installed version of BEDTools. However, in addition to passing filenames as in typical BEDTools command line usage, one may also pass collections of Interval objects which can be manipulated in Python on a feature-by-feature basis. Furthermore, BedTool methods return new BedTool instances, allowing users to chain many operations together in a fashion similar to the UNIX command line.

3 APPLICATION

The pybedtools package employs a syntax that is intuitive to Python programmers. For example, given an annotation file of genes, hg19.gff, and a file containing relevant genetic variation, angus.bed, one can identify genes that contain SNPs with the following:

```
from pybedtools import BedTool
```
At this point, one can easily examine the genes that overlap SNPs:

```python
for g in genes_with_snps:
    print g.chrom, g.start, g.end, len(g)
```

or filter the results with simple boolean functions:

```python
def chrom_filter(g):
    return g.chrom == 'chr21'
```

The underlying BEDTools commands send their results to "standard output". To assist in managing intermediate files, `pybedtools` automatically saves these results as temporary files that are deleted when Python exits. Results can be explicitly saved with the `saveas()` method:

```python
subset = subset.saveas('chr21-genes-snps.gff')
```

Given a FASTA file of the genome, `hg19.fa`, sequences for this subset of genes can be retrieved and saved with:

```python
subset.sequence('hg19.fa')
```

One of the more powerful extensions provided by the `pybedtools` interface is the ability to mix file operations with feature operations in a way that makes otherwise difficult tasks very accessible with minimal code. For example, the following identifies the closest gene (within 5 kb) to each intergenic SNP:

```python
intergenic_snps = (snps - genes)
```

```python
for g in genes_with_snps:
    print g.chrom, g.start, g.end
```

Finally, although `nearby` represents results that are a composite of GFF and BED features (i.e., `genes` and `snps`), the operation that produced `nearby` was driven by the `gene` GFF file. Therefore `gene.name` is seamlessly extracted from the GFF "attributes" field.

`pybedtools` also allows one to integrate sequence alignments in the widely used SAM/BAM format into their analyses. The following example illustrates how one would use `pybedtools` to identify sequence alignments that overlap coding exons.

```python
reads = BedTool('reads.bam')
exons = BedTool('exons.bed')
exonic = reads.intersect(exons)
```

Alternatively, this analysis could be reduced to the following statement:

```python
exonic = BedTool('reads.bam').intersect('exons.bed')
```

Some BEDTools programs require files containing chromosome sizes. `pybedtools` handles these automatically with the `genome` keyword argument to methods that wrap such programs. For example, the following command creates a bedGraph file of read coverage for the hg19 assembly:

```python
bedgraph = reads.genome_coverage(genome='hg19', bg=True)
```

## 4 Conclusion

The `pybedtools` package provides a convenient and flexible interface to both the BEDTools command-line tools and efficient functions in the BEDTools C++ libraries. `pybedtools` simplifies complicated analyses by extending the functionality in BEDTools and by providing, to our knowledge, the first Python library offering a common interface for manipulating datasets in diverse formats. Other new functionality includes: set operations on multiple datasets using a simple, intuitive syntax, the ability to filter features and select specific columns or attributes, a unified interface to common attributes (e.g., chromosome, start, end, name and strand) from many file formats, and a documented command history. `pybedtools` provides researchers with a simple and efficient interface for exploring complex genomics datasets in widely used formats.

**Funding:** Intramural Program of the National Institute of Diabetes and Digestive and Kidney Diseases.

**Conflict of Interest:** none declared.

## References

1. 1000 Genomes Project Consortium et al. (2010) A map of human genome variation from population-scale sequencing. Nature, 467, 1061–1073.
2. Danacek,P. et al. (2011) The Variant Call Format and VCFTools. Bioinformatics, 27, 2156–2158.
3. ENCODE Project Consortium et al. (2011) A user’s guide to the encyclopedia of DNA elements (ENCODE). PLoS Biol, 9, e1000546.
4. Giancarli,B. et al. (2009) Galaxy: a platform for interactive large-scale genome analysis. Genome Res, 19, 1451–1455.
5. Kent,W. et al. (2002) The human genome browser at UCSC. Genome Res, 12, 996–1006.
6. Li,H. et al. (2009) The Sequence Alignment/Map format and SAMTools. Bioinformatics, 25, 257 – 260.
7. Quentin,A. and Hall.I. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics, 26, 341–342.
8. Shen.L. (2010) The case for cloud computing in genome informatics. Genome Biol, 11, 207.