BioCantor: a Python library for genomic feature arithmetic in arbitrarily related coordinate systems

Pamela H. Russell[1] and Ian T. Fiddes [1]

[1] Inscripta, Inc., Boulder, CO 80301

Abstract

Motivation

Bioinformaticians frequently navigate among a diverse set of coordinate systems: for example, converting between genomic, transcript, and protein coordinates. The abstraction of coordinate systems and feature arithmetic allows genomic workflows to be expressed more elegantly and succinctly. However, no publicly available software library offers fully featured interoperable support for multiple coordinate systems. As such, bioinformatics programmers must either implement custom solutions, or make do with existing utilities, which may lack the full functionality they require.

Results

We present BioCantor, a Python library that provides integrated library support for arbitrarily related coordinate systems and rich operations on genomic features, with I/O support for a variety of file formats.

Availability and implementation

BioCantor is implemented as a Python 3 library with a minimal set of external dependencies. The library is freely available under the MIT license at https://github.com/InscriptaLabs/BioCantor and on the Python Package Index at https://pypi.org/project/BioCantor/. BioCantor has extensive documentation and vignettes available on ReadTheDocs at https://biocantor.readthedocs.io/en/latest/.

Introduction

The term “genomic feature arithmetic” refers to coordinate operations on representations of genomic features such as genes, transcripts or non-coding elements. Examples of feature arithmetic operations include coordinate conversion between coordinate systems, binary set theoretic operations such as intersection or union of features, and unary operations such as iterating over windows of a feature or reversing the strand of a feature. A variety of computational tools exist that support feature arithmetic operations, both as command line utilities (Bedtools[1]; BEDOPS [2]) and software libraries (Pybedtools [3]; PyRanges [4]; the GenomicFeatures[5] BioConductor package). However, no library exists that supports rich feature arithmetic operations across arbitrarily related coordinate systems: for example, a series of nested coordinate systems including exon-relative, transcript-relative, and chromosome-relative coordinates.

Here we present BioCantor, a Python library implementing a rich set of feature arithmetic operations including many not supported by other packages (**Table 1**). BioCantor facilitates importing and exporting common annotation file formats into a simple genome annotation data model, supports arbitrarily related coordinate systems and abstracts coordinate conversion from the user.

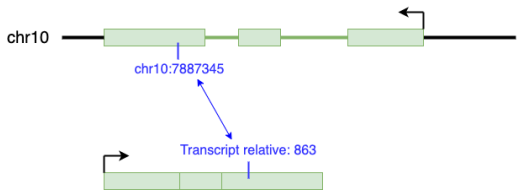
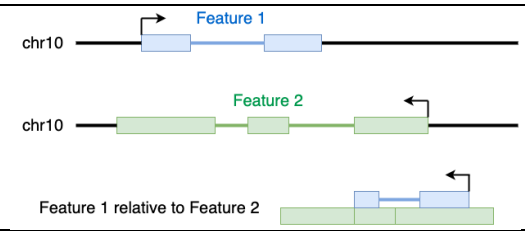
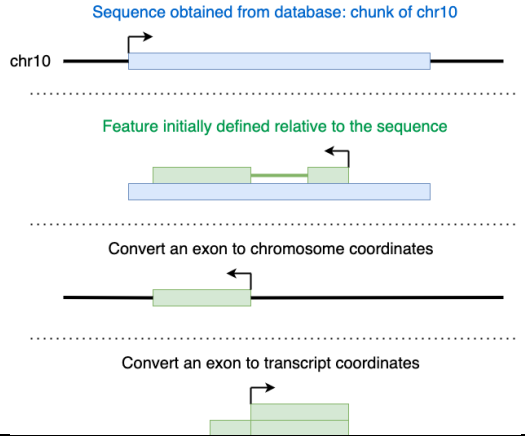
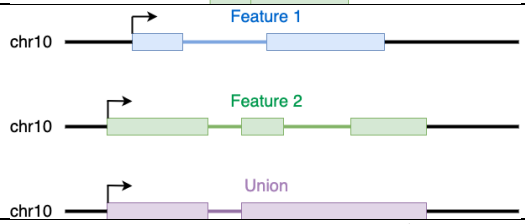
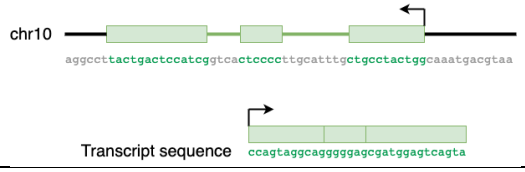
Example operation	Diagram	BioCantor statement(s)	Corresponding functionality available in existing tools
Convert between chromosome position and transcript relative position		feature.parent_to_relative_pos(7887345) feature.relative_to_parent_pos(863)	✗ Bedtools ✗ BEDOPS ✗ Pybedtools ✗ PyRanges GenomicFeatures ✗ Chr to transcript ✓ Transcript to chr
Get location of one feature relative to another feature		feature1.location_relative_to(feature2)	✗ Bedtools ✗ BEDOPS ✗ Pybedtools ✗ PyRanges ✗ GenomicFeatures
Define multiple nested coordinate systems and convert location through multiple layers		Single statements to: • Define a sequence that is a chunk of a chromosome, e.g. from a database • Define a feature on the sequence • Get an exon of the feature • Get exon coordinates relative to sequence • Get exon coordinates relative to chromosome • Get exon coordinates relative to transcript	✗ Bedtools ✗ BEDOPS ✗ Pybedtools ✗ PyRanges ✗ GenomicFeatures
Take union of two features		feature1.union(feature2)	✓ Bedtools ✓ BEDOPS ✓ Pybedtools ✓ PyRanges ✗ GenomicFeatures
Extract spliced sequence of transcript		feature.extract_sequence()	✓ Bedtools ✗ BEDOPS ✓ Pybedtools ✓ PyRanges ✓ GenomicFeatures

Table 1. Example BioCantor operations and availability in existing tools.

Results

BioCantor paradigm

The basis of the BioCantor paradigm is that objects are linked by parent/child relationships. Once a parent/child hierarchy is established, coordinate operations can move around the hierarchy with this detail abstracted to the user; for example, converting a feature annotation from one reference sequence to another. In most cases, parent/child relationships are used to establish the parent as the frame of reference for the location of the child, though these relationships may exist without defining a coordinate system.

Three main object types populate this paradigm. `Location` objects represent blocked and stranded features. `Sequence` objects hold sequence data. `Parent` objects define parent/child relationships. `Sequence` and `Parent` are concrete classes. `Location` is an abstract class with three implementations: the singleton `EmptyLocation`,

`SingleInterval` (a contiguous interval with start and end coordinates), and `CompoundInterval` (a multi-block feature).

All objects hold pointers to their own optional parent. `Parent` objects do not hold pointers to children and can be reused for multiple children. `Location` objects, `Sequence` objects, and `Parent` objects can all have parents. Multi-level hierarchies are established when `Parent` objects have their own parents.

Example: instantiating a `Location` object that refers to a parent sequence

```
sequence = Sequence('AAACCCAAAAAAAAAAAAAA', Alphabet.NT_STRICT)
location = SingleInterval(5, 8, Strand.PLUS, parent=sequence)
```

The `Parent` class is very flexible in order to accommodate different types of relationships. For example, a `Parent` object can optionally hold a pointer to a `Sequence`, meaning that sequence is the frame of reference for an object with that `Parent`. A `Parent` object can optionally hold a `Location`, meaning that is the location of the child relative to that parent. `Parent` has several optional parameters which enable different types of relationships and operations.

Example: a `Location` points to a slice of a chromosome as its parent. The chromosome slice holds sequence data. Additionally, the chromosome slice has its own `Parent` representing the location of the slice relative to a chromosome.

```
chr_slice = Sequence('TTTTTTTTTT', Alphabet.NT_STRICT,
                     parent=Parent(
                         id="chr1",
                         location=SingleInterval(1000, 1010, Strand.PLUS),
                         sequence_type="chromosome"))

location = SingleInterval(5, 8, Strand.PLUS, parent=chr_slice)
```

Coordinates and coordinate conversion

`Location` classes represent blocked, stranded features with block coordinates represented by zero-based, end exclusive coordinates. These classes provide a variety of conversion methods: coordinates and features can be converted between any coordinate systems in the hierarchy with a single statement. In particular, seamless conversion between genomic and transcript-relative coordinates enables expressive statements operating in transcript space.

Example: converting between transcript-relative and chromosome-relative coordinates

```
# Define a feature; parent can be omitted; ambient coordinate system is implied
feature = SingleInterval(100, 200, Strand.MINUS)

# Convert a feature-relative interval to parent (e.g implied chromosome) relative
chr_relative = feature.relative_interval_to_parent_location(7, 9, Strand.PLUS)

# Convert a chromosome-relative coordinate to feature-relative
feat_relative_coord = feature.parent_to_relative_pos(130)
```

Feature arithmetic

Alongside the support for coordinate systems, feature arithmetic functionality includes standard set theoretic operations (intersection, union, contains, overlaps, etc.) and other useful location operations. Splicing and strand are handled seamlessly. Moreover, the library includes special support for transcripts, coding sequences, codons, and translation, allowing users to quickly navigate among these features and retrieve their sequences.

Example: feature arithmetic operations

```
# Overlap
SingleInterval(5, 10, Strand.PLUS).has_overlap(SingleInterval(9, 20, Strand.PLUS))

# Intersection
CompoundInterval([2, 8], [5, 13], Strand.PLUS).intersection(SingleInterval(4, 10,
Strand.PLUS))

# Minus
SingleInterval(10, 20, Strand.PLUS).minus(SingleInterval(13, 15, Strand.PLUS))

# Reverse strand
SingleInterval(5, 10, Strand.PLUS).reverse_strand()
```

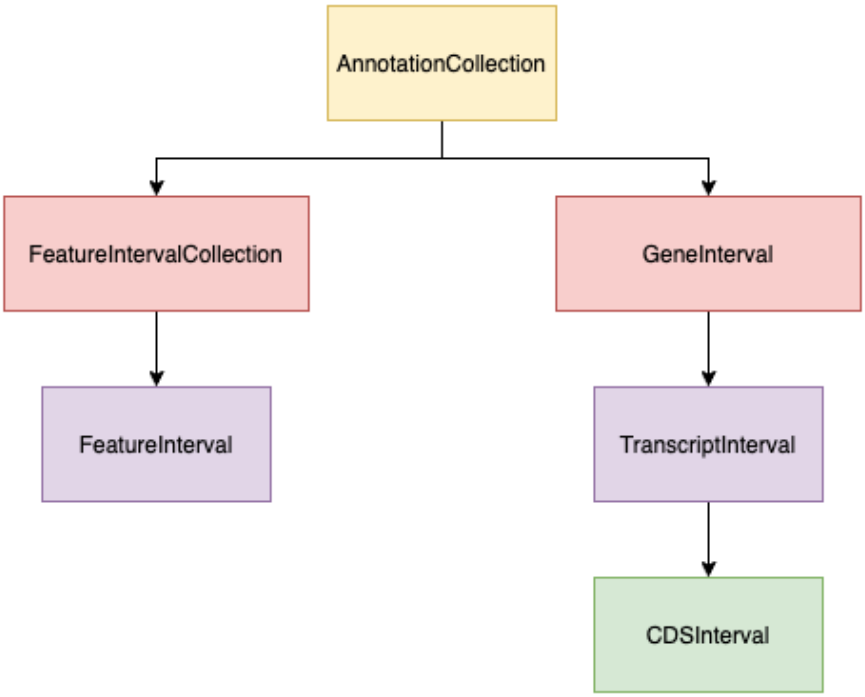


Figure 1. Annotation data structure. `AnnotationCollection` objects hold any number of arbitrary intervals in a contiguous genomic region on a single sequence. `AnnotationCollection` objects contain one or more `FeatureIntervalCollection` and `GeneInterval` children. `FeatureIntervalCollection` are thought of as generic regions of the genome, such as promoters or transcription factor binding sites. Both `GeneInterval` and `FeatureIntervalCollection` have one or more `TranscriptInterval` or `FeatureInterval` children respectively. `TranscriptInterval` objects have an optional child `CDSInterval` object that model their coding potential.

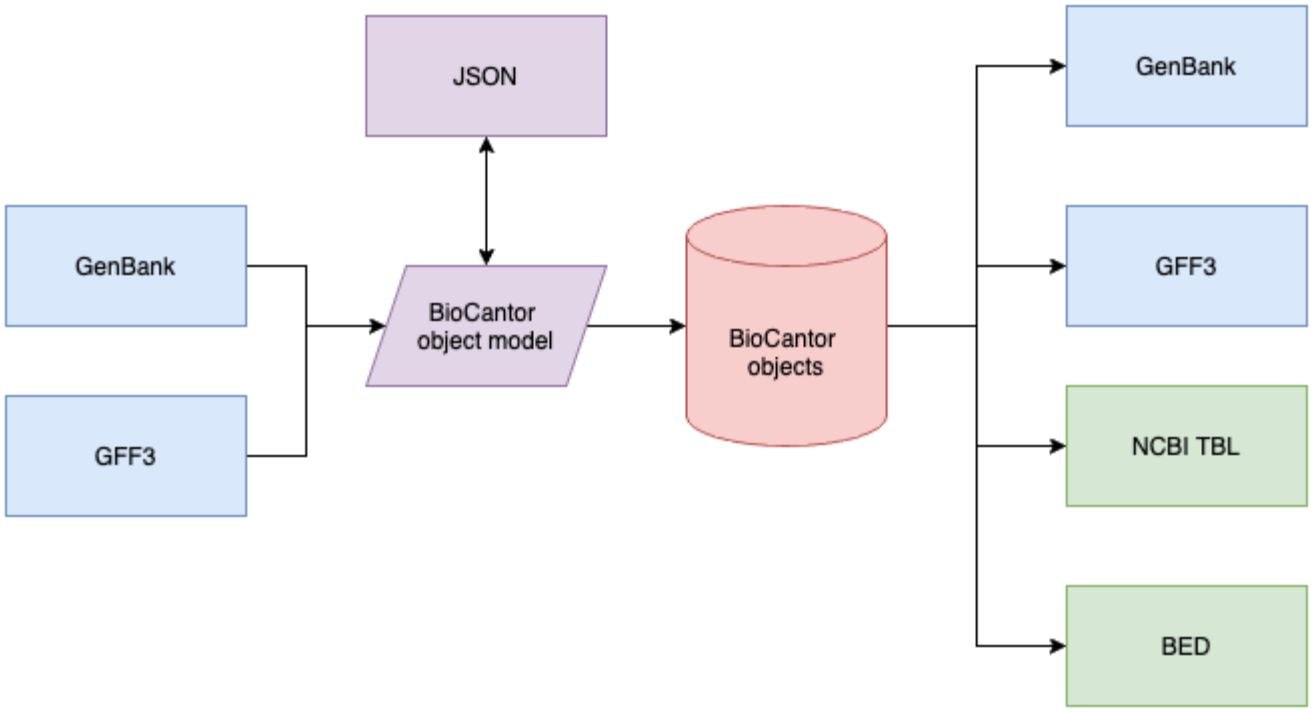


Figure 2. BioCantor file parsing. Parsers for GenBank and GFF3 files produce a JSON-serializable object model representation. The object model can be converted to BioCantor interval objects, with optional sequence information. These interval objects can be exported to GenBank, GFF3, NCBI Feature table (.tbl), and BED format.

Format	Underlying Parser	Notes
GenBank	BioPython[6]	Automatically associates sequence information.
GFF3	gffutils[7]	
GFF3 + separate FASTA	gffutils + BioPython	Automatically associates sequence information.
GFF3 + embedded FASTA	gffutils + BioPython	Automatically associates sequence information.

Table 2. Annotation file parser support. BioCantor supports parsing GenBank files as well as GFF3 files with or without FASTA files. Parsing GFF3 without FASTA will produce data structures that can perform coordinate arithmetic and be exported to other file formats but lack sequence information.

Feature collections

BioCantor provides container classes to combine sets of transcripts into a gene (`GeneInterval`), sets of arbitrary features into a collection, (`FeatureIntervalCollection`), and sets of genes and/or features into an arbitrary collection (`AnnotationCollection`) **(Figure 1)**. For example, an `AnnotationCollection` object could represent the full annotation of a chromosome loaded in from a GenBank or GFF3 file. `AnnotationCollection` objects can be queried for subsets of features overlapping specific coordinates.

Example: Constructing and using AnnotationCollection objects

```
# load data from a GFF3 with embedded FASTA (this file is a BioCantor test file)

from inscripta.biocantor.io.gff3.parser import parse_gff3_embedded_fasta,
ParsedAnnotationRecord
rec = list(
    ParsedAnnotationRecord.parsed_annotation_records_to_model(
        parse_gff3_embedded_fasta("tests/data/INSC1006_chrI.gff3")
    )
)[0]

# take a look at a (non-coding) transcript

rec.genes[0].transcripts[0]

>>> <TranscriptInterval((16174-18079:-), cds=[None], symbol=GI526_G0000001)>

# convert transcription oriented coordinate to chromosome

rec.genes[0].transcripts[0].transcript_pos_to_sequence(10)

>>> 18068

# get translation of a coding transcript

str(rec.genes[1].transcripts[0].get_protein_sequence())[:10]

>>> 'MTSEPEFQQA'
```

Data models and file formats

All of the BioCantor data structures are representable in JSON format, allowing them to be serialized to disk **(Figure 2)**. In order to facilitate building these representations, BioCantor includes parsers for GenBank and GFF3(+FASTA) **(Table 2)** format annotation files. In order to provide interoperability with common bioinformatics workflows, BioCantor data models can also be exported to GFF3, GenBank, BED, and NCBI TBL format.

Example: JSON representation of BioCantor gene model

```
{
    "transcripts": [
        {
            "exon_starts": [
                37461
            ],
            "exon_ends": [
                39103
            ],
            "strand": "PLUS",
            "cds_starts": [
                37637
            ],
            "cds_ends": [
                39011
            ],
            "cds_frames": [
                "ZERO"
            ],
```

```
            "qualifiers": {
                "gene": [
                    "GDH3"
                ]
            },
            "is_primary_tx": false,
            "transcript_id": "GI526_G0000002",
            "transcript_symbol": "GDH3",
            "transcript_type": "protein_coding",
            "sequence_name": "CM021111.1",
            "sequence_guid": null,
            "protein_id": "KAF1903245.1",
            "product": "GDH3 isoform 1",
            "transcript_guid": null,
            "transcript_interval_guid": "b78ce7f0-e8c4-0004-c92a-7e03ea6020f9"
        }
    ],
    "gene_id": "bc2688c9-db1f-42eb-b69c-353548e54174",
    "gene_symbol": "GDH3",
    "gene_type": "protein_coding",
    "locus_tag": "GI526_G0000002",
    "qualifiers": {
        "gene": [
            "GDH3"
        ]
    },
    "sequence_name": "CM021111.1",
    "sequence_guid": null,
    "gene_guid": "068b5fd8-0235-c816-45de-c0e0414bd67f"
}
```


Conclusions

BioCantor enables elegant genomic workflows to be expressed in custom Python code through full end-to-end support of rich feature operations. Furthermore, the abstraction of coordinate operations lets programmers operate in multiple simultaneous coordinate systems without the need to keep track of coordinate arithmetic. The flexible paradigm can be deployed to address any use case requiring genomic feature arithmetic.

References

1. Quinlan AR and Hall IM. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics. doi:10.1093/bioinformatics/btq033.
2. Neph, S. et al. (2012) BEDOPS: high-performance genomic feature operations. Bioinformatics. doi:10.1093/bioinformatics/bts277
3. Dale RK, Pedersen BS, and Quinlan AR. (2011) Pybedtools: a flexible Python library for manipulating genomic datasets and annotations. Bioinformatics. doi:10.1093/bioinformatics/btr539
4. Stovner EB and Saetrom P. (2019) PyRanges: efficient comparison of genomic intervals in Python. Bioinformatics. doi:10.1093/bioinformatics/btz615
5. Lawrence M et al. (2013) Software for Computing and Annotating Genomic Ranges. PLoS Computational Biology. doi:10.1371/journal.pcbi.1003118
6. Cock, P. et al. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. doi:10.1093/bioinformatics/btp163
7. Dale R. gffutils [Internet]. Github; Available: https://github.com/daler/gffutils