gff3-py Documentation

Release 0.3.0

Han Lin

May 14, 2015

Contents

Contents:

CHAPTER 1

gff3-py

Manipulate genomic features and validate the syntax and reference sequence of your GFF3 files.

- Free software: BSD license
- Documentation: https://gff3-py.readthedocs.org.

1.1 Features

- **Simple data structures**: Parses a GFF3 file into a structure composed of simple python dict and list.
- **Validation**: Validates the GFF3 syntax on parse, and saves the error messages in the parsed structure.
- **Best effort parsing**: Despite any detected errors, continue to parse the whole file and make as much sense to it as possible.
- Uses the python logging library to log error messages with support for custom loggers.
- Parses embeded or external FASTA sequences to check bounds and number of N s.
- Check and correct the phase for CDS features.
- Tree traversal methods ancestors and descendants returns a simple list in Breadth-first search order.
- Transfer children and parents using the adopt and adopted methods.
- Test for overlapping features using the overlap method.
- Remove a feature and its associated features using the remove method.
- Write the modified structure to a GFF3 file using the write mthod.

1.2 Quick Start

An example that just parses a GFF3 file named annotations.gff and validates it using an external FASTA file named annotations.fa looks like:

```
# validate.py
# ============
from gff3 import Gff3

# initialize a Gff3 object
gff = Gff3()
# parse GFF3 file and do syntax checking, this populates gff.lines and gff.features
```

```
# if an embedded ##FASTA directive is found, parse the sequences into gff.fasta_embedded
gff.parse('annotations.gff')
# parse the external FASTA file into gff.fasta_external
gff.parse_fasta_external('annotations.fa')
# Check seqid, bounds and the number of Ns in each feature using one or more reference sources
gff.check_reference(allowed_num_of_n=0, feature_types=['CDS'])
# Checks whether child features are within the coordinate boundaries of parent features
gff.check_parent_boundary()
# Calculates the correct phase and checks if it matches the given phase for CDS features
gff.check_phase()
```

A more feature complete GFF3 validator with a command line interface which also generates validation report in MarkDown is available under `examples/gff_valid.py`

The following example demonstrates how to filter, tranverse, and modify the parsed gff3 `lines` list.

1. Change features with type `exon` to `pseudogenic_exon` and type `transcript` to `pseudogenic_transcript` if the feature has an ancestor of type `pseudogene`
2. If a `pseudogene` feature overlaps with a `gene` feature, move all of the children from the `pseudogene` feature to the `gene` feature, and remove the `pseudogene` feature.

```
# fix_pseudogene.py
# =================
from gff3 import Gff3
gff = Gff3('annotations.gff')
type_map = {'exon': 'pseudogenic_exon', 'transcript': 'pseudogenic_transcript'}
pseudogenes = [line for line in gff.lines if line['line_type'] == 'feature' and line['type'] == 'pseu
for pseudogene in pseudogenes:
    # convert types
    for line in gff.descendants(pseudogene):
        if line['type'] in type_map:
            line['type'] = type_map[line['type']]
    # find overlapping gene
    overlapping_genes = [line for line in gff.lines if line['line_type'] == 'feature' and line['type
    if overlapping_genes:
        # move pseudogene children to overlapping gene
        gff.adopt(pseudogene, overlapping_genes[0])
        # remove pseudogene
        gff.remove(pseudogene)
gff.write('annotations_fixed.gff')
```

CHAPTER 2

Installation

At the command line:

```
$ easy_install gff3
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv gff3
$ pip install gff3
```

CHAPTER 3

Usage

To use gff3-py in a project:

```
from gff3 import Gff3
```

class gff3.**Gff3**(*gff_file=None*, *fasta_external=None*, *logger=<logging.Logger object>*)

add_line_error(*line_data*, *error_info*, *log_level=40*)
Helper function to record and log an error message

Parameters

- **line_data** – dict
- **error_info** – dict
- **logger** –
- **log_level** – int

Returns

adopt(*old_parent*, *new_parent*)
Transfer children from old_parent to new_parent

Parameters

- **old_parent** – feature_id(str) or line_index(int) or line_data(dict) or feature
- **new_parent** – feature_id(str) or line_index(int) or line_data(dict)

Returns List of children transferred

adopted(*old_child*, *new_child*)
Transfer parents from old_child to new_child

Parameters

- **old_child** – line_data(dict) with line_data['line_index'] or line_index(int)
- **new_child** – line_data(dict) with line_data['line_index'] or line_index(int)

Returns List of parents transferred

ancestors(*line_data*)
BFS graph algorithm

Parameters **line_data** – line_data(dict) with line_data['line_index'] or line_index(int)

Returns list of line_data(dict)

check_parent_boundary()
checks whether child features are within the coordinate boundaries of parent features

Returns

check_phase()

1.get a list of CDS with the same parent

2.sort according to strand

3.calculate and validate phase

check_reference(*sequence_region=False*, *fasta_embedded=False*, *fasta_external=False*, *check_bounds=True*, *check_n=True*, *allowed_num_of_n=0*, *feature_types=('CDS',)*)
Check seqid, bounds and the number of Ns in each feature using one or more reference sources.

Seqid check: check if the seqid can be found in the reference sources.

Bounds check: check the start and end fields of each features and log error if the values aren't within the seqid sequence length, requires at least one of these sources: ##sequence-region, embedded #FASTA, or external FASTA file.

Ns check: count the number of Ns in each feature with the type specified in *line_types (default: 'CDS') and log an error if the number is greater than allowed_num_of_n (default: 0), requires at least one of these sources: embedded #FASTA, or external FASTA file.

When called with all source parameters set as False (default), check all available sources, and log debug message if unable to perform a check due to none of the reference sources being available.

If any source parameter is set to True, check only those sources marked as True, log error if those sources don't exist.

Parameters

- **sequence_region** – check bounds using the ##sequence-region directive (default: False)
- **fasta_embedded** – check bounds using the embedded fasta specified by the ##FASTA directive (default: False)
- **fasta_external** – check bounds using the external fasta given by the self.parse_fasta_external (default: False)
- **check_bounds** – If False, don't run the bounds check (default: True)
- **check_n** – If False, don't run the Ns check (default: True)
- **allowed_num_of_n** – only report features with a number of Ns greater than the specified value (default: 0)
- **feature_types** – only check features of these feature_types, multiple types may be specified, if none are specified, check only 'CDS'

Returns error_lines: a set of line_index(int) with errors detected by check_reference

descendants(*line_data*)
BFS graph algorithm :param line_data: line_data(dict) with line_data['line_index'] or line_index(int) :return: list of line_data(dict)

parse(*gff_file*, *strict=False*)
Parse the gff file into the following data structures:

•lines(list of line_data(dict))

- line_index(int): the index in lines
- line_raw(str)
- line_type(str in ['feature', 'directive', 'comment', 'blank', 'unknown'])
- line_errors(list of str): a list of error messages
- line_status(str in ['normal', 'modified', 'removed'])
- parents(list of feature(list of line_data(dict))): may have multiple parents
- children(list of line_data(dict))
- extra fields depending on line_type
- **directive**
  - directive(str in ['##gff-version', '##sequence-region', '##feature-ontology', '##attribute-ontology', '##source-ontology', '##species', '##genome-build', '###', '##FASTA'])
  - extra fields depending on directive
- **feature**
  - seqid(str): must escape any characters not in the set [a-zA-Z0-9.:^*$@!+_?-|] using RFC 3986 Percent-Encoding
  - source(str)
  - type(str in so_types)
  - start(int)
  - end(int)
  - score(float)
  - strand(str in ['+', '-', '.', '?'])
  - phase(int in [0, 1, 2])
  - **attributes(dict of tag(str) to value)**
    - ID(str)
    - Name(str)
    - Alias(list of str): multi value
    - Parent(list of str): multi value
    - **Target(dict)**

      target_id(str)

      start(int)

      end(int)

      strand(str in ['+', '-', ''])
    - Gap(str): CIGAR format
    - Derives_from(str)
    - Note(list of str): multi value
    - Dbxref(list of str): multi value

- Ontology_term(list of str): multi value
- Is_circular(str in ['true'])

– **fasta_dict(dict of id(str) to sequence_item(dict))**

- id(str)
- header(str)
- seq(str)
- line_length(int)

•features(dict of feature_id(str in line_data['attributes']['ID']) to feature(list of line_data(dict)))

A feature is a list of line_data(dict), since all lines that share an ID collectively represent a single feature.

During serialization, line_data(dict) references should be converted into line_index(int)

Parameters

- **gff_file** – a string path or file object
- **strict** – when true, throw exception on syntax and format errors. when false, use best effort to finish parsing while logging errors

remove(*line_data*, *root_type=None*)

Marks line_data and all of its associated feature's 'line_status' as 'removed', does not actually remove the line_data from the data structure. The write function checks the 'line_status' when writing the gff file. Find the root parent of line_data of type root_type, remove all of its descendants. If the root parent has a parent with no children after the remove, remove the root parent's parent recursively.

Parameters

- **line_data** –
- **root_type** –

Returns

sequence(*line_data*, *child_type=None*, *reference=None*)

Get the sequence of line_data, according to the columns 'seqid', 'start', 'end', 'strand'. Requires fasta reference. When used on 'mRNA' type line_data, child_type can be used to specify which kind of sequence to return: * child_type=None: pre-mRNA, returns the sequence of line_data from start to end, reverse complement according to strand. (default) * child_type='exon': mature mRNA, concatenates the sequences of children type 'exon'. * child_type='CDS': coding sequence, concatenates the sequences of children type 'CDS'. Use the helper

function translate(seq) on the returned value to obtain the protein sequence.

Parameters

- **line_data** – line_data(dict) with line_data['line_index'] or line_index(int)
- **child_type** – None or feature type(string)
- **reference** – If None, will use self.fasta_external or self.fasta_embedded(dict)

Returns sequence(string)

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/hotdogee/gff3-py/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

gff3 could always use more documentation, whether as part of the official gff3 docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hotdogee/gff3-py/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

CHAPTER 5

Credits

5.1 Development Lead

- Han Lin <hotdogee@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.4.0 (2015-05-05)

- Added sequence functions: complement(seq) and translate(seq)
- Added fasta write function: fasta_dict_to_file(fasta_dict, fasta_file, line_char_limit=None)
- Added Gff method to return the sequence of line_data: sequence(self, line_data, child_type=None, reference=None)
- Gff.write no longer prints redundent '###' when the whole gene is marked as removed

0.3.0 (2015-03-10)

- Fixed phase checking.

CHAPTER 9

0.2.0 (2015-01-28)

- Supports python 2.6, 2.7, 3.3, 3.4, pypy.
- Don't report empty attributes as errors.
- Improved documentation.

CHAPTER 10

0.1.0 (2014-12-11)

- First release on PyPI.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

A

C

D

G

P

R

S