bio$_piecesDocumentation$

Release 1.1.0

Tyghe Vallard, Michael Panciera

January 04, 2016

Contents

Various bioinformatics scripts

All documentation is hosted at http://bio-pieces.readthedocs.org/en/latest

CHAPTER 1

TODO

- Include existing scripts

Contents:

1.1 Installation

1. Install dependencies

```
pip install -r requirements.txt
```

   For python 2.6 you will need to also install some additional packages

```
pip install -r requirements-py26.txt
```

2. Install bio_pieces

   It is recommended to install into a virtualenv. If you know what you are doing and don't want to install into virtualenv, then you can skip right to step 3

   (a) Setup Virtualenv

   It is assumed you have virtualenv already installed. If not see https://virtualenv.pypa.io/en/latest/installation.html

```
virtualenv env
```

   (b) Activate virtualenv

```
. env/bin/activate
```

   (c) Install bio_pieces

```
python setup.py install
```

1.2 Scripts

1.2.1 rename_fasta

Many times you find you have a fasta file where the identifiers are all wrong and you want to rename them all via some mapping file.

Take the example where you have the following fasta file(example.fasta):

```
>id1
ATGC
>id2
ATGC
>id3
ATGC
```

You want to rename each identifier(id1, id2, id3) based on a mapping you have. In a file called renamelist.csv you would have the following:

```
#From,To
id1,samplename1
id2,samplename2
id3,samplename3
```

Then to rename your fasta without replacing the original file you have two options:

1. Rename without replacing original file

```
        rename_fasta renamelist.csv example.fasta > renamedfasta.fasta
```

2. Rename replacing original file's contents

```
        reanme_fasta renamelist.csv example.fasta --inplace
```

Rename Mapping File Syntax

The file you specify as the rename map file is a simple comma separated text file.

The following rules apply to the format:

- The first entry is the identifier to find in the supplied fasta file.
- The second entry is what to replace the found identifier with
- Any line beginning with a pound sign(#) will be ignored by the renamer

Missing identifiers that are in fasta but not rename file

In the case where your fasta file contains an identifier that is not in the rename map file you supply, an error will be displayed in the console telling you as such:

```
idwhatever is not in provided mapping
```

1.2.2 beast_checkpoint

beast_checkpoint is a fork of https://gist.github.com/trvrb/5277297 that has been rewritten in python and slightly improved as the ruby script seemed to have a few errors.

It accepts any previously run or terminated beast run and will generate an xml file that essentially starts from the last generated tree/log state.

Since beast is random in nature, there does not appear to be a way to restart the run exactly from the same state that it left off.

Example

We will use the benchmark2.xml file that comes with Beast 1.8 This file is located in:

```
BEASTv1.8.0/examples/Benchmarks/benchmark2.xml
```

First you need to fix the benchmark2.xml because each taxa has a trailing space and that is annoying

```
$> sed 's/ "/"/' benchmark2.xml > beast.xml
```

Now run beast for about half of the iterations and hit CTRL-C to kill it This benchmark is set to run 1,000,000 iterations so around 500,000 you can kill it. Notice we are using a predefined seed

```
$> seed=1234567890
$> mkdir run1
$> cp beast.xml run1/beast.xml
$> beast -seed $seed -beagle_SSE beast.xml
```

Now we will want to re-run beast from that last state. We can use beast_checkpoint to do so by supplying the original xml and the produced trees and log files. We will put the new xml into a new directory since the .trees and .log files would create an error or possibly be overwritten.

NOTE If your fileLog and treeFileLog do not have the same logEvery then when beast exits you may end up with more/less tree states than log states. For now you will have to manually edit the files and ensure that the last tree state matches the last log state.

Todo

Could be possible to get beast_checkpoint to check for that scenario and use the last tree state that matches the last log state

```
$> mkdir run2
$> beast_checkpoint beast.xml *.trees *.log > run2/beast.xml
```

Now you can simply just re-run beast on the new xml using the same seed

```
$> cd run2
$> beast_checkpoint -seed $seed -beagle_SSE beast.xml
```

Tracer

If you name your runs sequentially as we did in the example(aka, run1, run2,...) then you can easily load all log files into tracer via the command line as follows

```
tracer run*/*.log
```

LogCombiner

After you have run all your beast checkpointed xml files you will probably want to combine them with logcombiner which comes with beast

1.2.3 beast_wrapper

Beast wrapper is intended as a helper script to run beast. At this point it just runs beast with the same arguments you would normally give to beast from the command line and just adds a estimated time left column to the console output

Example

```
$> beast_wrapper -beagle_SSE my_beast.xml
...
state   Posterior       Prior           Likelihood      rootHeight      my_beast.ucld.mean  location
0   -86527.5880     -6850.8316      -79676.7564     57.6772         1.16103E-3      4.86012         1
20000   -29044.3753     -1123.5287      -27920.8466     288.102         3.02471E-4      0.11891
40000   -25517.9525     -979.5343       -24538.4182     211.705         1.35118E-4      0.25060
60000   -24212.1250     -1040.4103      -23171.7147     188.454         1.05572E-4      0.18908
80000   -24097.9354     -1019.8099      -23078.1256     182.242         1.53593E-4      0.12857
100000  -24121.5382     -1105.6545      -23015.8837     178.060         1.26907E-4      0.10367
120000  -23930.6897     -1105.7390      -22824.9507     187.411         1.01885E-4      0.34214
140000  -23869.4856     -1087.1915      -22782.2942     178.535         8.76375E-5      0.26128
```

1.2.4 group_references

group_references splits an alignment file by reference into seperate FASTQ files. group_references takes a SAM or BAM file as input, and can optionally be given an output directory where the FASTQ files will be saved. If not output directory name is provided, the files will be saved in the new folder group_references_out.

```
$> group_references contigs.bam
$> group_references contigs.bam --outdir split_fastqs
```

1.3 AMOS

AMOS is a file format that is similar to any assembly file format such as ACE or SAM. It contains information about each read that is used to assemble each contig.

The format is broken into different message blocks. For the Ray assembler, it produces an AMOS file that is broken into 3 types of message blocks

- RED

```
        {RED
        iid:\d+
        eid:\d+
        seq:
        [ATGC]+
        .
        qlt:
        [A-Z]+
        }
```

iid Integer identifier

eid Same as iid?

seq Sequence data

qlt Should be quality, but is only a series of D's from Ray assembler

- TLE

```
{TLE
src:\d+
off:\d+
clr:\d+,\d+
}
```

src RED iid that was used

off One would think offset, but unsure what it actually means

clr Not sure what this is either

- CTG

```
{CTG
iid:\d+
eid:\w+
com:
.*$
.
seq:
[ATGC]+
.
qlt:
[A-Z]+
.
{TLE
...
}
}
```

iid integer id of contig

eid contig name

com Communication software that generated this contig

seq Contig sequence data

qlt Supposed to be contig quality data, but for Ray it only produces D's

TLE 0 or more TLE blocks that represent RED sequences that compose the contig

1.3.1 Parsing

bio_pieces contains an interface to parse a given file handle that has been opened on an AMOS file.

To read in the AMOS file you simply do the following

```python
from bio_pieces import amos
a = None
with open('AMOS.afg') as fh:
    a = amos.AMOS(fh)
```

CTG

To get information about the contigs(CTG) you can access the `.ctgs` attribute. The contigs are indexed based on their iid so to get the sequence of contig iid 1 you would do the following:

```
ctg = a.ctgs[1]
seq = ctg.seq
```

To retrieve all the reads(RED) that belong to a specific contig:

```
reads = []
for tle in ctg.tlelist:
    reads.append(a.reds[tle.src])
```

RED

To get information about the reads(RED) you can access the `.reds` attribute. The reds are indexed based on their iid so to get the sequence of red iid 1 you would do the following:

```
red = a.reds[1]
seq = red.seq
```

If you want to convert a RED entry into anything you can use the `.format` method. The `.format` method allows you to utilize any of the properties of a RED object such as `.iid`, `.eid`, `.seq`, `.qlt`. You can see in the examples below how to do this.

1.3.2 Examples

Here is an example of how to convert all RED blocks into a single fastq file

```
from bio_pieces import amos

# Fastq format string
fastq_fmt = '@{iid}\n{seq}\n+\n{qlt}'

with open('amos.fastq','w') as fh_out:
    with open('AMOS.afg') as fh_in:
        for iid, red in amos.AMOS(fh_in).reds.items():
            fq = red.format(fastq_fmt)
            fh_out.write(fq + '\n')
```

1.4 CHANGELOG

1.4.1 Version 1.1.0

- Renamed parse_contigs to group_references to better name functionality
- group_references now supports bam files

1.4.2 Version 1.0.0

- Version bump. Starting here we will employ semantic versioning

- Added version script to get version from project

1.4.3 Version 0.1.0

- Started project over to setup for Continuous Integration testing
- Added rename_fasta that can rename fasta sequence identifiers based on a input rename file
- Added travis, coveralls, readthedocs
- Added amos file parser that is specific to Ray assembler amos format
- Added format functionality for amos classes such that it is easy to convert to different formats
- Added amos2fastq to pull sequences out of AMOS files organized by their contigs.
- Added vcfcat.py, a commandline app for filtering and comparing vcf files.
- Completed documentation for vcfcat
- Added beast_checkpoint script and documentation
- Added beast_wrapper script that prints estimated time column in beast output
- Added beast_est_time script that allows you to easily get estimated time left from already running beast run

1.5 TODO

Todo

Could be possible to get beast_checkpoint to check for that scenario and use the last tree state that matches the last log state

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/bio-bits/checkouts/v1.1.0/docs/scripts/beast_checkpoint.rst, line 52.)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search