circRNA detection from RNA-seq reads

This repository holds python code to detect head-to-tail spliced (back-spliced) sequencing reads, indicative of circular RNA (circRNA) in RNA-seq data. It is also used extensively by circbase.

Author notes and preface

The scripts in this package were designed by Nikolaus Rajewsky, Antigoni Elefsinioti and Marvin Jens. The current implementation was written by Marvin Jens.

This software is provided to you "as is", without any warranty. We used this code (v1) to produce the results presented in

```
Nature. 2013 Mar 21;495(7441):333-8.
doi: 10.1038/nature11928. Epub 2013 Feb 27.

Circular RNAs are a large class of animal RNAs with regulatory potency.

Memczak S, Jens M, Elefsinioti A, Torti F, Krueger J, Rybak A, Maier L,
Mackowiak SD, Gregersen LH, Munschauer M, Loewer A, Ziebold U,
Landthaler M, Kocks C, le Noble F, Rajewsky N.
```

Have a look at Memczak *et al.* 2013 and the supplementary material for additional information. Please don't contact the authors, who are very busy, regarding this software unless you have found a bug, wish to provide improvements to the code, propose a collaboration, or use our code for commercial purposes. Thank you very much for your understanding!

Brief version history

The current code (v1.2) has some small bug fixes and improvements. Each version deemed stable is tagged and a brief summary of the improvements is given here:

- **v1** : as used in Memczak et al. 2013
- **v1.2** :
  - by default no longer report junctions with only one uniquely aligned anchor. Original behaviour can be restored using the `--halfuniq` switch.
  - fix the uniqueness handling. Occasionally reads would have either anchor align uniquely, but never both. These rare cases now get flagged as "NO_UNIQ_BRIDGES".

  - support all chromosomes in one fasta file
  - store the size of each chromosome upon first access, pad violations of chromosome bounds by padding with 'N'
  - category labels have been extended for clarity ("UNAMBIGUOUS_BP" instead of "UNAMBIGUOUS", etc.), in a manner which preserves functionality of `grep` commands.

- **v2** : (not released yet): produce multiple anchor lengths to potentially yield more junctions with unique anchor mappings.

License

For non-commercial purposes the code is released under the General Public License (version 3). See the file LICENSE for more detail. If you are interested in commercial use of this software contact the authors.

Prerequisites

The scripts run on Ubuntu 12.04.2 on a 64Bit machine with python 2.7. We do not know if it runs in different environments but other 64Bit unix versions should run if you can get the required 3rd party software installed.

You need to install the python packages numpy and pysam. If there are no packages in your linux distro's repositories, try the very useful python installer (building numpy requires many dependencies, so obtaining pre- compiled packages from a repository is a better option).

```
pip install --user pysam
```

Next you need the short read mapper bowtie2 and samtools up and running. samtools now is in the repositories of many distros, but here you can get the most fresh versions:

```
http://samtools.sourceforge.net/
http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
```

At this point you should have everything to run a built-in test data set

```
cd test_data
./run_test.sh
```

If you get error messages here, first make sure the dependencies are really installed correctly and run on their own. The authors of this code can not give support bowtie2, samtools, python, or any other third-party packages! Sorry,

but not enough time for this. If you are sure that the problem is with our code, just zip the test_data folder and e-mail it to us. MAYBE, we can help.

If everything goes well you can get started with your real data! :)
You need to have the reference genome and a bowtie2 index for it. As an example, let's assume you use C.elegans genome ce6 (WS190):

```
wget -c http://hgdownload.cse.ucsc.edu/goldenPath/ce6/bigZips/chromFa.tar.gz -P tmp
mkdir genome
cd genome; tar -xzf ../tmp/chromFa.tar.gz; cd ..
cat genome/chr*.fa > genome/ce6.fa
```

This will retrieve the genome from the UCSC website, unpack it into a folder 'genome' and create one big fasta file with all chromosomes to build the index:

```
bowtie2-build genome/ce6.fa bt2_ce6
```

How to use the unmapped2anchors.py script

It is recommended to map your RNA-seq reads against the genome first and keep the part that can not be mapped contiguously to look for splice- junctions afterwards. The genome alignments can be used for gene expression analysis and the unmapped reads will represent a fraction of the input, thus downstream analysis will be faster. (Note that –phred64 is only for Illumina quality scores. Sanger scores are –phred33, but that's the default anyway, so it could be omitted)

```
bowtie2 -p16 --very-sensitive --phred64 --mm -M20 --score-min=C,-15,0 \
-x bt2_ce6 -q -U <your_reads.qfa.gz> 2> bowtie2.log  \
| samtools view -hbuS - | samtools sort - test_vs_ce6
```

put the unaligned reads aside and split good quality reads into anchors for independent mapping (used to identify splice junctions)

```
samtools view -hf 4 test_vs_ce6.bam | samtools view -Sb - > unmapped_ce6.bam
./unmapped2anchors.py unmapped_ce6.bam | gzip > ce6_anchors.qfa.gz
```

How to use find_circ.py

Now we have everything to screen for spliced reads, from either linear or head-to-tail (circular) splicing:

```
mkdir <run_folder>
bowtie2 -p 16 --reorder --mm -M20 --score-min=C,-15,0 -q -x bt2_ce6 \
-U ce6_anchors.qfa.gz | ./find_circ.py -G genome -p ce6_test_
-s <run_folder>/sites.log > <run_folder>/sites.bed 2> <run_folder>/sites.reads
```

The important ingredients here are a folder in which each chromosome can be found as a separate fasta file (genome/chrI.fa ....), and a prefix for naming the splice junctions ("ce6test"). The rest is about output -s ce6_test/sites.log will receive statistics of the run, stdout reports the encountered junctions, and stderr the reads supporting each junction. Note: the read sequence can also be reported as the reverse complement of the raw reads sequence.

If you analyse reads from multiple sources (tissues, samples) in one run, you can ask find_circ.py to divvy up the contributions from each sample. For this, provide a two-column tab-separated flat file with a read-name prefix to match for and an associated tissue. Note that the matching is performed from top to bottom. so you want to match for "read_from_exp_1" BEFORE "read_from". See samples_example.txt to get an idea. The correspon- ding switch is find_circ.py -r samples_examples.txt

How to filter the output

The first 6 columns of 'sites.bed' are standard BED. The rest hold various quality metrics about the junction. (the 'score' field holds the number of distinct reads supporting the junction). Check the header line for a quick overview. It is usually a good idea to demand at least 2 reads supporting the junction, unambiguous breakpoint detection and some sane mapping quality:

To get a reasonable set of circRNA candidates try:

```
grep circ <run_folder>/sites.bed | grep -v chrM | ./sum.py -2,3 | \
    ./scorethresh.py -16 1 | ./scorethresh.py -15 2 | \
    ./scorethresh.py -14 2 | ./scorethresh.py 7 2 | \
    ./scorethresh.py 8,9 35 | ./scorethresh.py -17 100000 \
    > <run_folder>/circ_candidates.bed
```