Contents

circRNA detection from RNA-seq reads 1
Author notes and preface 1
License 2
Brief version history 2
Prerequisites 2
How to use the unmapped2anchors.py script 4
How to use find_circ.py 4
Output format 5
How to filter the output 6
Analyzing multiple samples 7
Command line reference 7

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!

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.

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** :
  - 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.
  - by default no longer report junctions with only one uniquely aligned anchor. Original behaviour can be restored using the `--halfuniq` switch.
  - by default no longer report junctions that lack a read where both anchors align uniquely (NO_UNIQ_BRIDGES keyword). Original behaviour can be restored using the `--report_nobridge` switch
- **v2** : (*under development*): produce multiple anchor lengths to potentially yield more junctions with unique anchor mappings.

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
make
```

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.

In case you are working with human data and have the hg19 genome and a bowtie2 index around, there is an additional test/sanity-check you can run:

```
cd test_data
make hek_test2 \
    GENOME_HG19=/path_to/hg19.fa \
    INDEX_HG19=/path_to/bowtie2_hg19_index
```

(obviously, the paths to genome and index will have to be changed for this to work) This will push known spliced reads, belonging to previously identified junctions, through `find_circ.py`, then take the found spliced reads and run them through `find_circ.py` a second time. Ultimately, it compares the detected splice sites and ensures the two sets are identical.

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 \
    -O - | gzip -dc | tar -xO > ce6.fa
```

This will retrieve the genome from the UCSC website, unpack it into a single fasta file with all chromosomes to build the index:

```
bowtie2-build 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.

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

single out the unaligned reads and split those with good quality into anchors for independent mapping (used to identify splice junctions)

```
# get the unmapped and pipe through unmapped2anchors.py
samtools view -hf 4 test_vs_ce6.bam | samtools view -Sb - | \
    ./unmapped2anchors.py unmapped_ce6.bam | gzip \
        > ce6_anchors.fastq.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 -p <run_folder>
bowtie2 -p 16 --score-min=C,-15,0 --reorder --mm \
    -q -U ce6_anchors.fastq.gz -x bt2_ce6 |\
        ./find_circ.py \
            --genome=ce6.fa \
            --prefix=ce6_test_ \
            --name=my_test_sample \
            --stats=<run_folder>/stats.txt \
            --reads=<run_folder>/spliced_reads.fa \
                > <run_folder>/splice_sites.bed
```

The prefix `ce6_test` is arbitrary, and pre-pended to every identified splice junction. You may consider setting it to `tmp` or similar for single samples out of a

larger set. Note that `find_circ.py` outputs both, circRNA splice junctions (containing the keyword `CIRCULAR`) linear splice junctions (containing the keyword `LINEAR`). You may want to `grep CIRCULAR <run_folder>/splice_sites.bed > circs_sample1.bed` or similar, to sort out the circRNAs.

Output format

The detected linear and circular candidate splice sites are printed to stdout. The first 6 columns are standard BED. The rest hold various quality metrics about the junction. Here is an overview:

column	name	description
1	chrom	chromosome/contig name
2	start	left splice site (zero-based)
3	end	right splice site (zero-based).
		(Always: end > start. 5' 3' depends on strand)
4	name	(provisional) running number/name assigned to junction
5	n_reads	number of reads supporting the junction (BED 'score')
6	strand	genomic strand (+ or -)
7	n_uniq	number of *distinct* read sequences supporting the junction
8	uniq_bridges	number of reads with both anchors aligning uniquely
9	best_qual_left	alignment score margin of the best anchor alignment
		supporting the left splice junction (`max=2 * anchor_length`)
10	best_qual_right	same for the right splice site
11	tissues	comma-separated, alphabetically sorted list of
		tissues/samples with this junction
12	tiss_counts	comma-separated list of corresponding read-counts
13	edits	number of mismatches in the anchor extension process
14	anchor_overlap	number of nucleotides the breakpoint resides within one anchor
15	breakpoints	number of alternative ways to break the read with flanking GT/AG
16	signal	flanking dinucleotide splice signal (normally GT/AG)
17	strandmatch	'MATCH', 'MISMATCH' or 'NA' for non-stranded analysis
18	category	list of keywords describing the junction. Useful for quick `grep` filtering

The following list of keywords is assigned to splice sites by `find_circ.py` for

easy filtering:

keyword	description
LINEAR	linear (mRNA) splice site, joining consecutive exons
CIRCULAR	potential circRNA splice site. Exons are joint in reverse order.
UNAMBIGUOUS_BP	demanding flanking GT/GA, only one way of splitting the spliced read was found (only one possible breakpoint within the read)
PERFECT_EXT	The read sequence between the anchors aligned perfectly during the extension process.
GOOD_EXT	The extension (see above) required not more than one mismatch or one nucleotide overlap with an anchor
OK_EXT	The extension (see above) required not more than two mismatches or two nucleotides overlap with an anchor
ANCHOR_UNIQUE	Unique anchor alignments have been found, supporting both sides of the junction. Unless `--halfunique` is used, this should be true for all reported results.
CANONICAL	splice sites are flanked by GT/AG. Unless `--noncanonical` is used, this should be true for all reported results.
NO_UNIQ_BRIDGES	While both sides of the junction are individually supported by unique anchor alignments, there is not a single read, where both anchors align uniquely at the same time (bridging the junction). Unless `--report_nobridge` is used, this should never appear.
STRANDMATCH	Only appears when `--stranded` is used and GT/AG were found in the correct orientation

How to filter the output

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 CIRCULAR <run_folder>/splice_sites.bed | \
    grep -v chrM | \
    grep UNAMBIGUOUS_BP | grep ANCHOR_UNIQUE | \
```

```
./maxlength.py 100000 \
> <run_folder>/circ_candidates.bed
```

This selects the circular splice sites with unambiguous detection of the breakpoint (*i.e.* the exact nucleotides at which splicing occurred), and unique anchor alignments on both sides of the junction. The last part subtracts start from end coordinates to compute the genomic length, and removes splice sites that are more than 100 kb apart. These are perhaps trans-splicing events, but for sure they are so huge they can seriously slow down any downstream scripts you may want to run on this output.

Analyzing multiple samples

If you intend to analyze multiple samples, it is now strongly advised to run them individually through `find_circ.py`, and merge the separate outputs later! Use the `find_circ.py --name <sample_name>` flag to assign sample IDs, tissue names, *etc.* to each sample.

Merging should then be done with `merge_bed.py`:

```
./merge_bed.py sample1.bed sample2.bed [...] > combined.bed
```

This will deal properly with the various columns: quality scores will be assigned the maximum value of all samples, total read counts will be summed up, `tissue` column will contain a comma-separated list, *etc.*.

Command line reference

```
Usage:

bowtie2 [mapping options] anchors.fastq.gz | find_circ.py [options] > candidates.bed


Options:
-h, --help            show this help message and exit
-v, --version         get version information
-S SYSTEM, --system=SYSTEM
                        model system database (optional! Requires byo
                        library.)
-G GENOME, --genome=GENOME
                        path to genome (either a folder with chr*.fa or one
                        multichromosome FASTA file)
-n NAME, --name=NAME  tissue/sample name to use (default='unknown')
-p PREFIX, --prefix=PREFIX
```

```
                        prefix to prepend to each junction name (default='')
-q MIN_UNIQ_QUAL, --min_uniq_qual=MIN_UNIQ_QUAL
                        minimal uniqness for anchor alignments (default=2)
-a ASIZE, --anchor=ASIZE
                        anchor size (default=20)
-m MARGIN, --margin=MARGIN
                        maximum nts the BP is allowed to reside inside an
                        anchor (default=2)
-d MAXDIST, --maxdist=MAXDIST
                        maximum mismatches (no indels) allowed in anchor
                        extensions (default=2)
--noncanonical          relax the GU/AG constraint (will produce many more
                        ambiguous counts)
--randomize             select randomly from tied, best, ambiguous hits
--allhits               in case of ambiguities, report each hit
--stranded              use if the reads are stranded. By default it will be
                        used as control only, use with --strandpref for
                        breakpoint disambiguation.
--strandpref            prefer splice sites that match annotated direction of
                        transcription
--halfunique            also report junctions where only one anchor aligns
                        uniquely (less likely to be true)
--report_nobridges      also report junctions lacking at least a single read
                        where both anchors, jointly align uniquely (not
                        recommended. Much less likely to be true.)
-R READS, --reads=READS
                        write spliced reads to this file instead of stderr
                        (RECOMMENDED!)
-B BAM, --bam=BAM       filename to store anchor alignments that were recorded
                        as linear or circular junction candidates
-r READS2SAMPLES, --reads2samples=READS2SAMPLES
                        path to tab-separated two-column file with read-name
                        prefix -> sample ID mapping
-s STATS, --stats=STATS
                        write numeric statistics on the run to this file
```