# atactk Documentation

***Release 0.1.5***

**The Parker Lab**

January 11, 2016

# Contents

CHAPTER 1

# Introduction

## 1.1 What's in the box?

### 1.1.1 Programs we've found useful in ATAC-seq pipelines

- `trim_adapters`: based on Jason Buenrostro's utility for trimming Illumina adapters by aligning paired reads to each other.
- `make_cut_matrix`: useful in conjunction with CENTIPEDE, and in generating plots of transcription factor binding sites.
- `plot_aggregate_matrix.R`: generates plots for motifs given the aggregate output of *make_cut_matrix*

### 1.1.2 A Python library you can use in your own tools for processing ATAC-seq data

The code underpinning our command-line tools has allowed us to make our pipelines shorter and faster. Our ATAC-seq scoring functions work directly with a BAM file and run in parallel, without the overhead of invoking external applications. Particularly if you're trying to produce quantitative metrics from your data, starting with your BAM files, converting them to BED and bigWig so you can run bigWigSummary, you might find your pipeline can be simplified too.

## 1.2 License

GPLv3 or any later version.

CHAPTER 2

# Installation

At the command line:

```
git clone https://github.com/ParkerLab/atactk
pip install ./atactk
```

The dependencies should be installed automatically.

## 2.1 Requirements

- Python. We've run it successfully under versions 2.7.10 and 3.4.3.
- pysam
- python-levenshtein
- sexpdata

CHAPTER 3

# Usage

## 3.1 Command-line applications

### 3.1.1 Adapter trimming

The trim_adapters utility is based on a script by Jason Buenrostro (see the original ATAC-seq paper: Buenrostro, Jason D; Giresi, Paul G; Zaba, Lisa C; Chang, Howard Y; Greenleaf, William J. 2013. *Transposition of native chromatin for fast and sensitive epigenomic profiling of open chromatin, DNA-binding proteins and nucleosome position*. Nat Meth 10, 1213–1218.)

Instead of looking for known adapter sequence, it aligns paired reads to each other and trims off sequence outside the alignment. More precisely, it searches the forward read for the reverse complement of a specified number of bases (20 by default) at the beginning of the reverse read, then falls back to finding the best alignment of the two reads, using the minimum Levenshtein distance between them.

Changes from the original script:

- The `--max-edit-distance` option to specify the maximum edit distance when aligning the reads.
- The `--fudge` option to not trim a base from the result to satisfy bowtie – other aligners like bwa don't have the apparent problem with exactly overlapping reads. The default is 1 for compatibility with the original script.
- The `--rc-length` option to specify the amount of the reverse read to reverse complement and search for in the forward read.
- The `--trim-from-start` option to remove extra bases from the beginning of each read. We found this necessary with OH-seq.
- Output is gzipped.

### 3.1.2 Cut matrix generation

The `make_cut_matrix` script can be used to generate two types of matrices by counting the ATAC-seq cut points (transposition sites) around known motifs, given a BAM file of aligned reads from an ATAC-seq experiment and a BED file of motifs.

#### Discrete matrices

The first type of output, which we call *discrete*, is intended to produce input for CENTIPEDE. The output matrix contains a row for each motif, representing the cut point counts at positions within and around the motif for each fragment size bin. For each fragment size bin and resolution you specify, the cut points at each position in the motif and an extended region you specify are counted, and every *resolution* positions in the extended area on either side of the motif are summed. So each row will contain for each fragment size bin a sequence of (possibly aggregated) scores in the region upstream of the motif, a sequence of scores for each position in the motif, and a sequence of (possibly aggregated) scores in the region downstream of the motif.

An example invocation:

```
make_cut_matrix -d -b '(36-149 150-324 325-400 1)' -p 8 \
  input.bam \
  JASPAR2014_CTCF.bed.gz | \
  gzip -c > CTCF.discrete.matrix.gz
```

That would count any reads with lengths between 36-149, 150-324, and 325-400 from `input.bam` whose cut points fell in the specified region around motifs from `JASPAR2014_CTCF.bed.gz`. The cut point counts would be recorded for each fragment size bin, at nucleotide resolution (with no score aggregation in the extended region around each motif). Since the length of the extended region was not specified, it would use the default. The program would use eight concurrent scoring processes, and the output would wind up in CTCF.discrete.matrix.gz.

## Aggregate matrices

After you've run CENTIPEDE with the resulting *discrete* matrix, and identified bound and unbound motifs (perhaps using posterior probabilities of at least 0.95 or at most 0.5, respectively), you can move on to generating what we call *aggregate* matrices. These are designed for creating a plot of the ATAC-seq signal around a single motif.

An *aggregate* matrix contains a row for each combination of position, fragment size bin, and strand within and around the motif, with columns representing the absolute and mean cut point counts at each position.

An example invocation, using a BED file of bound motifs:

```
make_cut_matrix -a -b '(36-149 150-324 325-400 1)' -p 8 \
  input.bam \
  CTCF_bound.bed.gz | \
  gzip -c > CTCF_bound.aggregate.matrix.gz
```

Do the same for your unbound motifs, and you're ready to plot.

## Binning

For either matrix, we count cut points in groups of bins according to the length of the reads' fragments, with optional reduction of scores in regions around motifs to a resolution you specify for each group. This is regrettably complex to explain, so I will resort to crude pictures.

Assume you want to count cut points from fragments with lengths in the following ranges at different resolutions:

| Fragment bin group | Resolution |
|---|---|
| 36-149 | 1 |
| 150-224 and 225-324 | 2 |
| 325-400 | 5 |

The command line specification for this scenario would look like this:

```
make_cut_matrix -a -b '(36-149 1) (150-224 225-324 2) (325-400 5)' ...
```

Pretend you're scoring a motif 5 bases long, with a 10-base extended region on either side, and for simplicity, pretend that each template length bin had the same counts of cut points around the motif, shown here:

```
extended region     motif     extended region
------------------- --------- -------------------
0 1 2 3 3 4 4 4 4 5 9 2 0 2 7 5 4 4 4 4 3 3 2 1 0
```

The matrix would contain scores for each position in the first bin group, `(36-149 1)`:

```
extended region     motif     extended region
------------------- --------- -------------------
0 1 2 3 3 4 4 4 4 5 9 2 0 2 7 5 4 4 4 4 3 3 2 1 0
```

The second bin group, `(150-224 225-324 2)` would contain sums of every two scores in the extended region, plus every score in the motif itself:

```
extended region                 motif     extended region
------------------------------- --------- ------------------------------
(0+1) (2+3) (3+4) (4+4) (4 + 5) 9 2 0 2 7 (5+4) (4+4) (4+3) (3+2) (1+ 0)
```

resulting in:

```
e.r.      motif     e.r.
--------- --------- ---------
1 5 7 8 9 9 2 0 2 7 9 8 7 5 1
```

Furthermore, since this group contains two bins, what ultimately goes into the output matrix would be the entrywise sum of each bin's scores.

The scores for the last bin group, `(325-400 5)`, after adding every five scores in the extended region:

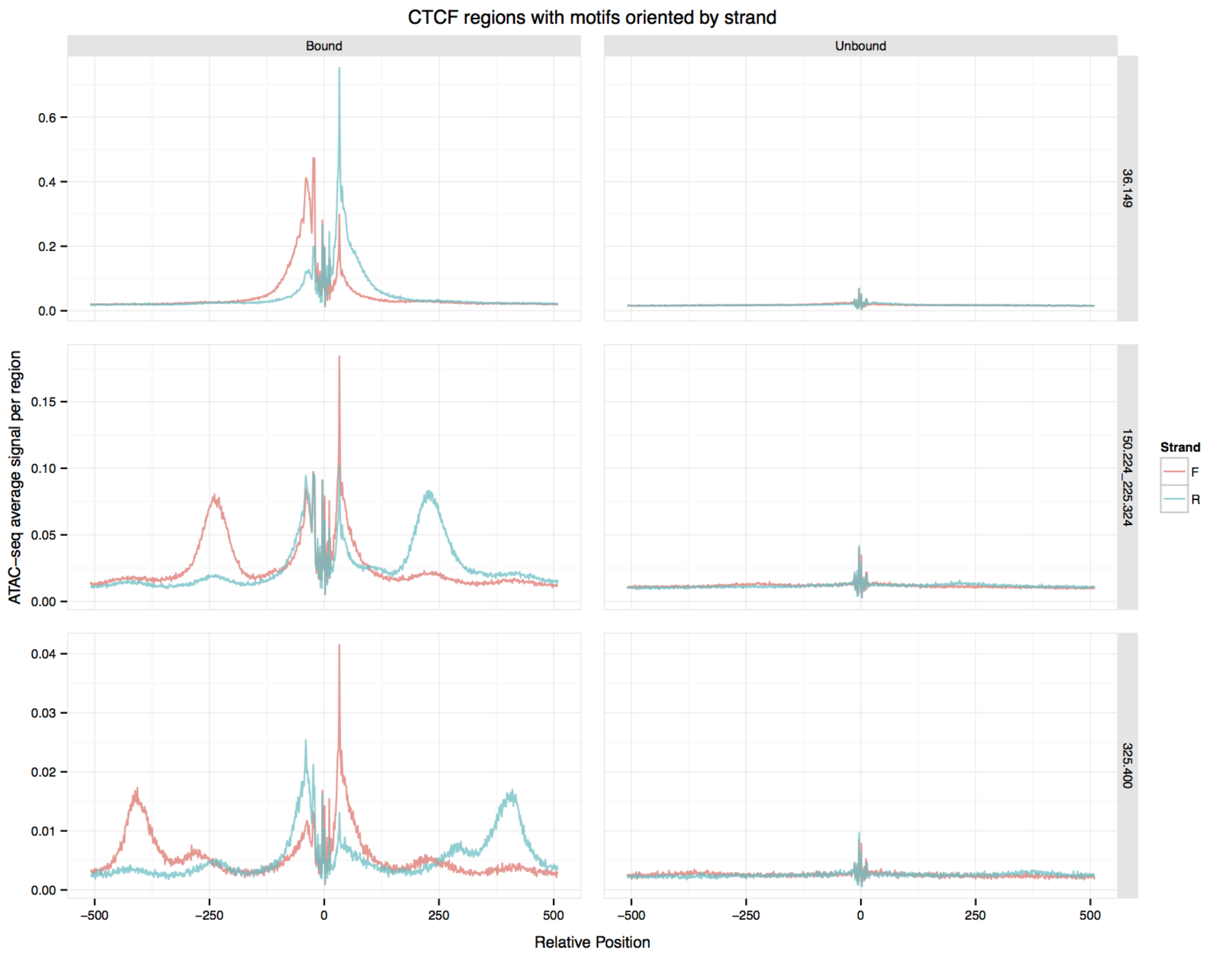
```
e.r. motif     e.r.
---- --------- ----
9 21 9 2 0 2 7 21 9
```

### 3.1.3 Creating ATAC-seq signal plots for motifs

The output of `make_cut_matrix --aggregate-output` can be plotted with `plot_aggregate_matrix.R`. Pass it the aggregate output for a bound motif, the aggregate output for an unbound motif, a title for the plot and the name of the PDF file in which to save the plot.

An example of the output produced by:

```
plot_aggregate_matrix.R CTCF_bound.aggregate.matrix CTCF_unbound.aggregate.matrix "CTCF regions with
```



## 3.2 Using the `atactk` library

There are several modules in the `atactk` package that you might find useful in processing ATAC-seq data, particularly `atactk.data` and `atactk.metrics`.

The `atactk.data` module simplifies reading and manipulating features from BED files. It handles gzipped or uncompressed files automatically, and makes it simple to filter aligned segments from a BAM file using standard SAM flags. It also makes it easy to read two FASTQ files simultaneously, producing a sequence of paired reads.

The `atactk.metrics` module makes it easy to measure ATAC-seq cut points around a feature.

There are also the `atactk.command` and `atactk.util` modules, which support parsing of our interval specifications and provide some generic functional tools used in the other modules.

We've made an effort to ensure the library is completely documented (see the modindex). If you find the documentation incomplete or unclear, please file a bug report.

CHAPTER 4

# Getting help

If you have questions about installing or using atactk, please visit the project's Google Group at:

https://groups.google.com/forum/#!forum/atactk/

If you've found a bug, please file a report at GitHub:

https://github.com/ParkerLab/atactk/issues/

If you'd prefer to contact us privately, send mail to:

parkerlab-software@umich.edu

CHAPTER 5

# Contributing

## 5.1 Contributing knowledge

If you’ve found a bug, or have a suggestion for improving the toolkit, please let us know by creating a GitHub issue at:

https://github.com/ParkerLab/atactk/issues

## 5.2 Contributing code

We welcome contributions of code, too. Here’s how to get started.

1. Fork the repo on GitHub: https://github.com/ParkerLab/atactk/
2. Set up a Python virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv atactk
$ git clone git@github.com:your_name_here/atactk.git
$ cd atactk/
$ python setup.py develop
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. *(Optional, but much appreciated.)* When you’re done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 atactk tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just *pip install* them into your virtualenv.

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, please update the documentation, especially docstrings and script help text.
2. If you have time to write tests too, great, but we understand you're volunteering your time to help our project, and we will take care of making sure changes are tested.

CHAPTER 6

# Credits

The atactk package was built by the Parker Lab at the University of Michigan, with contributions from:

- John Hensley
- Ricardo D'Oliveira Albanus
- Stephen Parker

The trim_adapters utility is based on a script by Jason Buenrostro (see the original ATAC-seq paper: Buenrostro, Jason D; Giresi, Paul G; Zaba, Lisa C; Chang, Howard Y; Greenleaf, William J. 2013. *Transposition of native chromatin for fast and sensitive epigenomic profiling of open chromatin, DNA-binding proteins and nucleosome position*. Nat Meth 10, 1213–1218.)

CHAPTER 7

# History

## 7.1 0.1.5 (2016-01-08)

Fix bug in score_feature introduced in 0.1.2: careless use of a generator expression caused aligned_segments_in_bin to be consumed when gathering forward_aligned_segments, so reverse_aligned_segments was always empty, resulting in low cut point counts.

Change bin key construction to ensure the R script treated bins as characters.

Stop insisting motif input files match an official BED format; as long as the first six fields work, ignore the rest and get on with it.

## 7.2 0.1.4 (2015-12-17)

When generating an aggregate matrix, ensure that there is always a line for each position, fragment size bin, and strand, even if there is no signal there.

Support reading motifs from standard input, which required removing time estimates, and returning the motif from atactk.metrics.score_feature.

Remove option for reverse feature shift.

## 7.3 0.1.3 (2015-12-10)

Speed up scoring.

Open the alignment file once in each worker process, instead of in each call to score_feature. There's a surprising amount of overhead in pysam's opening of BAM files. The actual fetch calls are really quick.

The AlignmentFile instances are not supposed to be safe to share between processes, so each worker still has to have its own, but it's a big reduction in overhead; things are roughly twice as fast now.

Also refactor make_cut_matrix, mainly to enable profiling.

## 7.4 0.1.2 (2015-12-06)

Fix an overlooked realization of the list of motifs, which would be copied into all processes. Use a generator expression for the results when using only one process.

Convert a couple of other list comprehensions to generator expressions.

Change –parallel default to 1.

Improve logging.

## 7.5 0.1.1 (2015-12-01)

Arbitrary cut point location.

Now you can specify where (or whether) the cut point happens relative to the ends of reads. We're using this to compare our footprinting with DNase-seq data.

## 7.6 0.1.0 (2015-10-31)

- First release.

CHAPTER 8

# Indices and tables

- genindex
- modindex