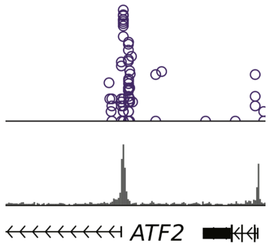
Oct 25, 2019

Processing Bulk Calling Card Sequencing Data



DOI
dx.doi.org/10.17504/protocols.io.xwjfpcn

Arnav Moudgil[1], Rob Mitra[1]
[1]Washington University, Saint Louis

Transposon Calling Cards

Arnav Moudgil
Washington University, Saint Louis

OPEN ACCESS

DOI: dx.doi.org/10.17504/protocols.io.xwjfpcn

Protocol Citation: Arnav Moudgil, Rob Mitra 2019. Processing Bulk Calling Card Sequencing Data. **protocols.io** **https://dx.doi.org/10.17504/protocols.io.xwjfpcn**

License: This is an open access protocol distributed under the terms of the **Creative Commons Attribution License**, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited

Protocol status: Working
We use this protocol and it's working

Created: February 08, 2019

Last Modified: October 25, 2019

Protocol Integer ID: 20139

Abstract

Here we present a computational pipeline for processing bulk RNA calling card data. These data will have been generated from transfection/-duction of either undirected *piggyBac* transposase or your favorite transcription factor (YFTF) fused to *piggyBac*. Multiple biological replicates should have been generated, each with a unique combination of primer barcode and index sequences. This workflow demonstrates how to analyze a single replicate; the workflow can be parallelized on distributed computing architectures (e.g. slurm).

Materials

The following external programs are required:

- **cutadapt** (≥ 1.16)
- **samtools** (≥ 1.9)

You will need a genomic aligner. Here we will use novoalign, but in theory any aligner should be sufficient (e.g. bowtie2, GATK, STAR, etc.). Also, you will need a .2bit version of the genome sequence you are aligning to; these are readily available from the **UCSC Genome Browser**. (They can also be generated from a FASTA file using the **faToTwoBit** utility)

The following programs are optional, but highly recommended:

- **bedtools** (≥ 2.27)
- **bedops** (≥ 2.4)

In addition, this workflow calls some calling card-specific scripts, which use Python 3. It is recommended that your Python installation be relatively up-to-date (i.e. ≥ 3.4). To check your python version, type

```
python -V
```

You will need to install the following Python modules:

- **numpy**
- **pandas**
- **pysam**
- **twobitreader**

All of these packages are available on PyPI and can be installed via pip:

```
pip install numpy pandas pysam twobitreader
```

(If Python3 is not the default on your system, replace pip with pip3)

Finally, these are the calling card-specific scripts you will need, all of which are available on **GitHub**:

- TagBam.py
- AnnotateInsertionSites.py
- BamToCallingCard.py

Before start

Please make sure you have installed the required software and packages (see Materials section).

This protocol describes how to analyze a **SINGLE** biological replicate from a bulk RNA calling cards* experiment. Multiple replicates (e.g. 10-12) should be analyzed in each experiment to distinguish independent transposition events into the same insertion site. This is essential for adequate statistical power to detect transcription factor binding sites. Each replicate can be processed following this protocol, making appropriate changes to the primer barcode sequence and/or the index sequence(s). Data from multiple calling card replicates can be pooled at the end into a single file.

∗If you are unfamiliar with calling card libraries, we recommend reading our **quick start guide** and our **library preparation protocol**.

Preamble

1 The objective of this protocol is to take sequencing reads from a calling cards library and process them into a CCF (calling card format; .ccf) file. A CCF file is a modified BED file (BED3+3) that concisely enumerates every transposition event in the sequenced library.

CCF files typically have six columns:

1. chrom: chromosome
2. start: beginning coordinate of the insertion site
3. end: ending coordinate of the insertion site; since *piggyBac* inserts into TTAA's, this typically spans the motif itself.
4. count: the number of reads supporting this insertion
5. strand: + or -, indicating which strand was targetted (optional but highly recommended)
6. barcode: a string identifying the library from which this insertion originated (optional but highly recommended)

This workflow will walk through how to perform quality control, alignment, filtering, and processing of calling card sequencing libraries to generate a CCF file. This file can then be used in downstream applications, such as visualization on the (legacy) **WashU Epigenome Browser** (instructions **here**), and as input for peak calling.

2 To illustrate the workflow, let's say that we have performed bulk RNA calling cards on our favorite transcription factor (YFTF) in a human cell line. We have prepared libraries from 10 biological replicates of cells transfected with wild-type *piggyBac* transposase, and 10 replicates of cells with YFTF-*piggyBac*. We have sequenced these libraries and now need to map these insertions across the genome.

We will consider a single replicate; the workflow can then be repeated for all remaining replicates. At the end we can combine the data from the 10 piggyBac replicates, and the 10 YFTF-piggyBac replicates, respectively, into a single CCF file each.

3 In this example, we will be analyzing a single replicate from the wild-type *piggyBac* libraries: PBase_rep1. The read 1 sequencing file is PBase_rep1_L001_R1_001.fastq.gz

Note

For bulk RNA calling card libraries, only read 1 is analyzed, as it contains the junction between the transposon and genome.

This biological replicate had GAT as its primer barcode and CTCACGGTGA as its index sequence. It was prepared by PCR ligation with the following primers:

>OM-PB-GAT (barcode in bolded)
AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT**GAT**
TTTACGCAGACTATCTTTCTAG
>Nextera_N7_CTCACGGTGA (index in bold; note the reverse complement orientation)
CAAGCAGAAGACGGCATACGAGAT**TCACCGTGAG**GTCTCGTGGGCTCGG

Thus, each read 1 *should* begin with GATTTTACGCAGACTATCTTTCTAG.

Adapter Trimming

4 The purpose of this step is to check that reads have (1) the appropriate primer barcode sequence and (2) the transposon sequence is correct and ends in TTAA, *piggyBac*'s insertion motif. If these conditions are true, those bases are trimmed (hard clipped), to facilitate genomic alignment. Only reads with perfect matches to the barcode and transposon sequence are carried forward.

```
cutadapt \
    -g ^GATTTTACGCAGACTATCTTTCTAGGGTTAA \
    --minimum-length 1 \
    --discard-untrimmed \
    -e 0 \
    --no-indels \
    -o PBase_rep1_trimBC.fastq.gz \
    PBase_rep1_L001_R1_001.fastq.gz
```

Typically 70-90% of reads will pass this filter, although there may be sample-dependent variation.

5 Next, we re-examine the passing reads and trim any reads that end in the Nextera adapter that was added during tagmentation. This step reduces the amount of non-genomic bases, which should accelerate alignment. Only a small fraction (5-10%) typically have any adapter sequence at all, so virtually every read will pass this filter.

```
cutadapt \
    -a
CTGTCTCTTATACACATCTCCGAGCCCACGAGACTCTCACGGTGATCTCGTATGCCGTCTTCTGCT
TG \
    --minimum-length 1 \
    -o PBase_rep1_trimmed.fastq.gz \
    Base_rep1_trimBC.fastq.gz
```

The index sequence has been emphasized in bold, but if you are processing libraries with many index different indexes, you can replace the bolded sequence with N's (keeping the length same). cutadapt can handle degenerate bases in adapters.

Alignment

6 Now that our reads are trimmed, we are ready to align them to the genome. This step can be done with any aligner; we typically use novoalign, so that is what we will demonstrate here.

```
novoalign \
    -d hg38.nvx \
    -f PBase_rep1_trimmed.fastq.gz \
    -n 40 \
    -o SAM \
    -o SoftClip > PBase_rep1_trimmed.sam
```

The "-n 40" flag tells novoalign to align only the first 40 bases of the read. We have found that this reduction can increase the speed of alignment with minimal impact on total number of insertions recovered. Faster aligners (e.g. bowtie2, GATK, STAR) may not need this setting.

7 After alignment, we filter out reads that mapped to multiple locations in the genome (e.g. in a repetitive element) and convert to the more space-efficient BAM format.

```
samtools view \
    -bS -h -F 260 \
    PBase_rep1_trimmed.sam | \
    samtools sort - -o PBase_rep1_mapped.bam
```

Annotation

8 The BAM format provides a flexible way to annotate reads through the use of short tags. These tags remain with the reads in the BAM file, which makes for a simple and portable archive of a calling card experiment. We use the following custom tags:

- XP: primer barcode
- XJ: index 1 sequence
- XK: index 2 sequence (optional; reserved for future use)
- XI: insertion site annotation
- XZ: adjacent sequence (to verify transposase motif)

9 First, we will annotate reads with the XP tag for the primer barcode GAT.

```
python TagBam.py \
    --tag XP:Z:GAT \
    PBase_rep1_mapped.bam \
    PBase_rep1_tagged.bam
```

10 Next, we will annotate reads with the XJ tag for the index sequence CTCACGGTGA.

```
python TagBam.py \
    --tag XJ:Z:CTCACGGTGA \
    PBase_rep1_tagged.bam \
    PBase_rep1_tagged2.bam
```

11 Lastly, we will annotate reads with respect to the insertion site. This script checks each read to make sure that it maps next to the *piggyBac* insertion site motif TTAA. Remember, this part of read 1 was trimmed in step 4. By double checking that the read maps next to a genomic TTAA, we add an extra layer of specificity to the alignment. The sequence of the adjacent bases will also be annotated with the XZ tag. Reads that pass will be annotated with the insertion site coordinates in the XI tag and written to the output file.

```
python AnnotateInsertionSites.py \
    --transposase PB \
    -f \
    PBase_rep1_tagged2.bam \
    hg38.2bit \
    PBase_rep1_final.bam
```

You can provide a path to the .2bit file if your genome references are in another directory.

Finishing Up

12 To finish, we first index the BAM file.

```
samtools index PBase_rep1_final.bam
```

13 Next, clean up intermediate files.

```
rm PBase_rep1_trimBC.fastq.gz
rm PBase_rep1_trimmed.fastq.gz
rm PBase_rep1_trimmed.sam
rm PBase_rep1_mapped.bam
rm PBase_rep1_tagged.bam
rm PBase_rep1_tagged2.bam
```

14 Lastly, convert the BAM file to a CCF file.

```
python BamToCallingCard.py \
    -b XP XJ \
    -i PBase_rep1_final.bam \
    -o PBase_rep1_final.ccf
```

This will use the combination of primer barcode and index sequence (XP and XJ, respectively) to identify insertions derived from different biological replicates.

Here is an example of a CCF file:

```
chr1    28575   28579   2       +       GCA/TCGCCACCC
chr1    28575   28579   10      +       TAG/GAGGTACAG
chr1    28575   28579   1       +       GAT/GAGGTACAG
chr1    31191   31195   1       +       GCA/TCGCCACCC
chr1    31191   31195   49      +       TAG/TCGCCACCC
chr1    46620   46624   5       +       CTA/GAGGTACAG
chr1    54136   54140   42      -       GCA/TCGCCACCC
chr1    54818   54822   16      -       CTA/TCGCCACCC
chr1    57829   57833   6       -       CGT/GAGGTACAG
chr1    58414   58418   40      +       CTA/TCGCCACCC
```

Notes

15 This workflow described how to process a **SINGLE** biological replicate. After each replicate has been processed, CCF files can be combined to consolidate all insertions from a given experiment. For example, to combine data from all replicates from our wild-type *piggyBac* libraries, we can use cat and bedops (preferred):

```
cat \
PBase_rep1_final.ccf \
PBase_rep2_final.ccf \
PBase_rep3_final.ccf \
PBase_rep4_final.ccf \
PBase_rep5_final.ccf \
PBase_rep6_final.ccf \
PBase_rep7_final.ccf \
PBase_rep8_final.ccf \
PBase_rep9_final.ccf \
PBase_rep10_final.ccf | sort-bed - > PBase.ccf
```

Similarly, CCFs from the YFTF replicates can be combined:

```
cat YFTF-PBase_rep*_final.ccf | sort-bed - > YFTF-PBase.ccf
```

The concatenated CCF files can also be sorted using bedtools:

```
cat PBase_rep*_final.ccf | bedtools sort -i - > PBase.ccf
```

Or, using the standard shell sort command:

```
cat PBase_rep*_final.ccf | sort -k1V -k2n -k3n > PBase.ccf
```

16 Analogously, we can combine BAM files from biological replicates into a single archival-quality BAM file for an entire experiment:

```
samtools merge PBase.bam PBase_rep*_final.bam
```

17 Ideally, each biological replicate will have a unique primer barcode AND unique index sequence. However, sometimes this is not possible. If so, each replicate should be identifiable from a unique combination of primer barcode and index sequence. If multiple replicates share an index, their reads will be found in the same FASTQ file. This is okay as step 3 can separate each replicate based on an exact match to the primer barcode sequence. In that case, you will have to provide the same input file to step 4 multiple times, each with a different primer barcode at the start of the adapter.