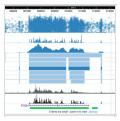


Mar 10, 2020

Calling Peaks on piggyBac Calling Card Data

Arnav Moudgil[1], Rob Mitra[1]

[1]Washington University, Saint Louis

1 Works for me dx.doi.org/10.17504/protocols.io.bb9xir7n

Transposon Calling Cards

Arnav Moudgil
Washington University, Saint Louis

ABSTRACT

This protocol describes how to all peaks on mammalian calling card data using either undirected, or transcription factor fusions, to the *piggyBac* transposase. It is applicable for both bulk as well as single cell calling card data.

THIS PROTOCOL ACCOMPANIES THE FOLLOWING PUBLICATION

https://doi.org/10.1101/538553

MATERIALS TEXT

This protocol requires a FASTA file for your genome of interest (e.g. hg38.fa) and the following script:

- kmer.cc

In addition, this workflow relies on 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 the following Python modules:

- numpy
- pandas
- scipy
- statsmodels
- pybedtools
- astropy

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

```
pip install numpy pandas scipy statsmodels pybedtools astropy
```

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

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

- SegmentCCF.py
- CCFIdeogram.py
- BBPeakCaller_TF.py
- BBPeakCaller_BRD4.py

The following programs are optional, but highly recommended:

- bedtools (≥ 2.27)
- bedops (≥ 2.4)

BEFORE STARTING

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

This protocol describes how to go from a CCF file, the processed output of a calling cards experiment, to a set of peaks enriched for

transcription factor (TF) binding. If you are unfamiliar CCF files, we recommend reading our bulk or single cell calling card data processing protocols first. We assume that you have performed either bulk calling cards (with sufficient replicates) or single cell calling cards with undirected *piggyBac* (to map BRD4 binding) and, optionally, with a TF-*piggyBac* fusion for your favorite TF (YFTF). To identify peaks in BRD4 binding, you should prepare a single CCF containing insertions from all replicates of undirected *piggyBac* calling cards. To call peaks on YFTF peaks, you will need two CCF files: one from all replicates of undirected *piggyBac* experiments, and one from all replicates of YFTF-*piggyBac* experiments.

Preprocessing

1 Before calling peaks on calling card data, it is useful to create a file listing the location of every TTAA tetramer in the genome. The *piggyBac* transposase inserts almost exclusively into this motif. Morever, we use the presence of a TTAA adjacent to a mapped read as an internal quality check when creating CCF files. This section will walk you through how to quickly find all TTAA's in a genome.

2 Compile the kmer.cc program as follows:

```
g++ kmer.cc -o kmer
```

The result should be a C++ executable in your directory called **kmer**.

3 This program takes as input a FASTA file and k-mer and outputs a BED file of all exact matches to that k-mer. Here we use it to find all exact matches to the 4-mer TTAA.

Download or copy to your working directory a FASTA file of your genome of interest. Using the latest human genome build as an example:

```
./kmer hg38.fa TTAA > hg38_TTAA.bed
```

The file hg38_TTAA.bed now lists all TTAA's in hg38.fa.

> It is important that the FASTA file that you use in this step is the same FASTA sequence used for aligning calling card reads. For example, if you mapped to a repeat-masked genome earlier, you should supply a repeat-masked FASTA file here.

4 (Optional) More recent builds of the human and mouse genomes contain unplaced contigs and alternate haplotypes. You maybe interested in restricting your analysis to the "canonical" chromosomes (e.g. 1-22, X & Y for humans). Here is a simple way to filter only "canonical" TTAAs:

```
grep -v '_' hg38_TTAA.bed > hg38_TTAA_canon.bed
```

(If you are being nitpicky, this file will include TTAAs on the mitochondrial chromosome, but we have not found this to be a problem for peak calling).

> If you filter only "canonical" TTAAs, it is important that you also filter your CCF file so it contains only insertions mapping to "canonical" chromosomes. The above command can be used to do so. If you do not do this, you may get "divide by zero" errors in subsequent steps.

5 The TTAA file only needs to be generated once per genome. Afterwards, all experiments using the same reference genome can use the same TTAA file.

Bayesian Blocks

6 The core of calling peaks in calling card data is Bayesian blocks. This algorithm, originally developed in astrophysics, segments one-dimensional datasets into regions of piecewise-constant density. We use it to initially partition the genome into intervals, where each interval contains a constant rate of *piggyBac* insertions. These intervals are referred to as **blocks**; two adjacent blocks are characterized by different insertion rates and, accordingly, different insertion densities. One attractive reason for using Bayesian blocks is that it can find a mathematically optimal partition of the data into blocks. Peak calling then proceeds by testing each block to see if it contains more insertions than expected by some background model.

This much overview of Bayesian blocks is sufficient to understand peak calling. For more details, we recommend reading the original paper (Scargle et al. 2013) or this blog post by Jake VanderPlas.

7 We generate a list of blocks from CCF files, but these files must first be sorted. Here are three ways to sort CCF files, in order of preference:

Using bedops:

```
sort-bed sample.ccf > sample_sorted.ccf
```

Using bedtools:

```
bedtools sort -i sample.ccf > sample_sorted.ccf
```

Using the standard shell sort command:

```
sort -k1V -k2n -k3n sample.ccf > sample_sorted.ccf
```

For the remainder of this protocol, we assume your CCF files are already sorted.

Calling BRD4 Peaks

8 Here we will describe how to call BRD4 peaks from undirected *piggyBac* insertions. Our sample file is HCT-116_PBase.ccf, which contains insertions from 10 replicates of bulk RNA calling cards in the HCT-116 cell line. This file contains 1.5 million insertions:

```
❯ wc -l HCT-116_PBase.ccf
1521048 HCT-116_PBase.ccf
```

9 We start by creating creating blocks from the CCF file. To do this, we use the SegmentCCF.py script:

```
python SegmentCCF.py HCT-116_PBase.ccf | sed -e '/^\s*$/d' > HCT-116_PBase.blocks
```

The output file is a BED-formatted list of blocks inferred by Bayesian blocks. The piped sed command simply removes blank lines.

You may see a warning about false positive rates for event data, as well as possibly a dividing by zero warning. These are automatically generated by astropy, the library which contains the Bayesian blocks algorithm and can be safely ignored. We have successfully called peaks with the blocks generated despite these warnings.

Segmenting the CCF file is often the most time-consuming step. The Bayesian blocks algorithm has quadratic runtime complexity. If one CCF file has twice as many insertions as another, the former is expected to take roughly four times longer to segment as the latter.

10 We then provide the CCF and blocks files, as well as the TTAA file, to BBPeakCaller_BRD4.py, which tests each block for statistical significance. This script performs the following steps:

1. Divide the number of insertions by the number of TTAAs in the TTAA file. This defines a global rate parameter (r) under a null model assuming a uniform distribution of insertions.
2. For each block b, count the number of TTAAs in b and multiply it by r. This value specifies the expected number of insertions in b if insertions were uniformly distribution (denoted λ_b).
3. For each block b, let x_b be the number of observed insertions in the block. The script then performs a one-tailed Poisson significance test on the block. This is calculated as the probability of observing x_b insertions or more in the block parametrized by a Poisson distribution with expected value λ_b.
4. Multiple hypothesis correction is performed (based on user preferences).
5. Finally, blocks that pass multiple hypothesis correction are polished and written to file. The output file is in BED format.

11 BBPeakCaller_BRD4.py takes four required positional arguments: CCF file, blocks file, TTAA file, and output filename. An example command would look like this:

```
python BBPeakCaller_BRD4.py HCT-116_PBase.ccf HCT-116_PBase.blocks hg38_TTAA_canon.bed HCT-116_PBase_peaks.bed
```

This command **WILL NOT RUN** as written because it does not specify how peaks should be thresholded. See Step 12 for details.

12 **Required flag:**

BBPeakCaller_BRD4.py has one required flag which specifies the statistical threshold for filtering blocks. There are two options for this: a straight p-value cutoff or with a multiple hypothesis correction method. The former is specified by the –p flag; the value supplied to it must be the –log10 transformation of the desired p-value.

Example: select only those blocks with $p < 10^{-9}$

```
-p 9
```

Alternatively, you can control for multiple hypotheses at a desired alpha level. This is specified by the –a flag and the value supplied is **not** transformed. If this option is used, you must supply a method (–m) of multiple hypothesis correction. Valid methods are listed here.

Example: Bonferroni correction at an alpha of 0.05

```
-a 0.05 -m bonferroni
```

Example: Benjamini-Hochberg correction at false discovery rate of 10%

```
-a 0.1 -m fdr_bh
```

Remember: You must use **EITHER** –p **OR** –a –m for the program to run.

BRD4 peaks may require choosing a p-value cutoff that is more stringent than, for example, Bonferroni correction. This appears to scale with size of the dataset: with more insertions, a more stringent cutoff is needed. To guide settling on an optimal p-value, we recommend calling peaks at a variety of cutoffs, visualizing CCF data and peak files (eg. on the WashU Epigenome Browser), and choosing a value whose peak boundaries reasonably accord with insertion densities.

Optional flags:

Mutliple significant blocks may occur in close proximity to one another. If you want to merge these into larger peaks, you can specify a **distance (–d)**. Significant blocks within this distance will be merged together.

Example: merge blocks within 12.5 kb

```
-d 12500
```

Finally, while the primary output of BBPeakCaller_BRD4.py is a list of peaks in BED format, an **intermediate filename (-i)** can be supplied to write information about each block and it's p-value. This file will be written in CSV format.

Example: write an intermediate file for the HCT-116 PBase dataset

```
-i HCT-116_PBase_intermediate.csv
```

13 The blocks file, in addition to being used to call peaks, can also be used to calculate normalized insertion densities across the genome. This is done using the CCFIdeogram.py script, which takes as input a CCF file and a blocks file and outputs a bedgraph file. Each entry in the bedgraph file is a block and the numerical value for each block is the number of insertions in that block per million mapped insertions per kilobase (IPKM).

```
python CCFIdeogram.py HCT-116_PBase.ccf HCT-116_PBase.blocks HCT-116_PBase.bedgraph
```

This script is named after ideograms because the resulting bedgraph files, when visualized as densities, create banding patterns that resemble karyotyped chromosomes.

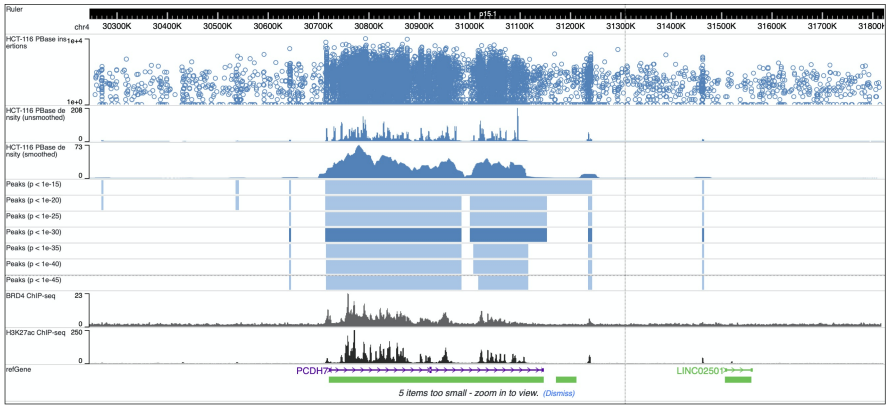
14 Let's put this all together. Here is the command to recreate our analysis from our recent preprint:

```
python BBPeakCaller_BRD4.py -p 30 -d 12500 HCT-116_PBase.ccf HCT-116_PBase.blocks
hg38_TTAA_canon.bed HCT-116_PBase_peaks.bed
```

This generates a BED file containing nearly 2000 peaks.

```
❯ wc -l HCT-116_PBase_peaks.bed
1939 HCT-116_PBase_peaks.bed
```

15 Here is the output of our sample analysis as visualized on the WashU Epigenome Browser. The top track is the raw CCF data. The next track is the per-block insertion densities as calculated by CCFIdeogram.py. The third track is the same as the second but with in-browser smoothing (15 px). We then show peak boundaries at a variety of p-value thresholds, in order of increasing stringency. Notice how peaks grow, merge, shrink, and vanish at different cutoffs. The dark blue peaks track corresponds to the threshold used in the previous step. BRD4 and H3K27ac data, marks of enhancers and super-enhancers, are shown for reference.



Calling TF Peaks

16 Peak calling on TF-directed *piggyBac* insertions is similar to calling BRD4 peaks. The major distinction is in the choice of background model. With undirected *piggyBac*, the background is modeled as a uniform distribution of the observed number of insertions. For TF-directed *piggyBac*, however, the background is the undirected *piggyBac* dataset from the same cell line or system.

For this example, we will use HCT-116_SP1-PBase.ccf. This file was generated from 10 replicates of bulk RNA calling cards with SP1-*piggyBac* in HCT-116 cells. The control file is the undirected *piggyBac* data from the same system, i.e. HCT-116_PBase.ccf.

```
❯ wc -l HCT-116_SP1-PBase.ccf
410588 HCT-116_SP1-PBase.ccf
```

17 As before, we start by creating creating blocks from the CCF file:

```
python SegmentCCF.py HCT-116_SP1-PBase.ccf | sed -e '/^\s*$/d' > HCT-116_SP1-PBase.blocks
```

The same notes from Step 9 apply here as well.

18 We then provide the CCF and blocks files, as well as the background CCF file, to BBPeakCaller_TF.py, which tests each block for statistical significance. This script performs the following steps:

1. Divide the number of insertions in the TF CCF file by the number of insertions in the background CCF file. This defines a global scaling parameter (s). This enables us to account for library size differences between the TF-directed and undirected control libraries.
2. For each block b, count the number of insertions from the background CCF file in b and multiply it by s, then add a pseudocount c. This value specifies the normalized expected number of insertions in b from the undirected control experiment (denoted λ_b).
3. For each block b, let x_b be the number of insertions from the TF-directed CCF file. The script then performs a one-tailed Poisson significance test on the block. This is calculated as the probability of observing x_b insertions or more in the block parametrized by a Poisson distribution with expected value λ_b.
4. Multiple hypothesis correction is performed (based on user preferences).
5. Finally, blocks that pass multiple hypothesis correction are polished and written to file. The output file is in BED format.

19 BBPeakCaller_TF.py takes four required positional arguments: TF-directed CCF file, TF-directed blocks file, undirected CCF file, and output filename. An example command would look like this:

```
python BBPeakCaller_TF.py HCT-116_SP1-PBase.ccf HCT-116_SP1-PBase.blocks HCT-116_PBase.ccf HCT-116_SP1-PBase_peaks.bed
```

This command **WILL NOT RUN** as written because it does not specify how peaks should be thresholded. See Step 20 for details.

20 **Required flag:**

BBPeakCaller_TF.py has one required flag which specifies the statistical threshold for filtering blocks. There are two options for this: a straight p-value cutoff or with a multiple hypothesis correction method. The former is specified by the –p flag; the value supplied to it must be the –log10 transformation of the desired p-value.

Example: select only those blocks with $p < 10^{-9}$

```
-p 9
```

Alternatively, you can control for multiple hypotheses at a desired alpha level. This is specified by the –a flag and the value supplied is **not** transformed. If this option is used, you must supply a method (–m) of multiple hypothesis correction. Valid methods are listed here.

Example: Bonferroni correction at an alpha of 0.05

```
-a 0.05 -m bonferroni
```

Example: Benjamini-Hochberg correction at false discovery rate of 10%

```
-a 0.1 -m fdr_bh
```

Remember: You must use **EITHER** –p **OR** –a –m for the program to run.

Optional flags:

Mutliple significant blocks may occur in close proximity to one another. If you want to merge these into larger peaks, you can specify a **distance (–d)**. Significant blocks within this distance will be merged together.

Example: merge blocks within 250 bp

```
-d 250
```

Peaks are composed of one or more blocks. Bayesian blocks draws block boundaries halfway between adjacent insertions. This can, in some cases, lead to unnecessarily wide peaks. The **refine (–r)** flag constrains the block edges so that they start and end at insertions. This, in turn, helps increase the resolution of peak calls.

Peaks can be further filtered based on a size threshold. You can specify a **minimum (–n)** and **maximum (–x)** size bound on reported peaks.

Example: report all peaks less than 5 kb in length

```
-x 5000
```

Example: report only peaks between 100 and 500 bp in length

```
-n 100 -x 500
```

TF-*piggyBac* fusions redirect, but do not abolish, *piggyBac*'s natural affinity for BRD4. This is why TF-directed experiments must use an undirected calling card experiment as a control. This can also pose a challenge for peak calling: whereas most TF's have narrow, sharp peaks, BRD4 can bind much broader stretches of the genome. Peak calling may not completely eliminate this signal, which is typically reflected in large, but statisically significant, peaks. Broad peaks can also occur in the shoulder regions flanking a TF binding site, likely from "spillover" of insertions by the increased local concentration of transposase.

A simple way to increase peak specificity is to threshold on peak size. In our experience, in a number of cell lines with a variety of TFs, 5 kb is a reasonable upper bound for filtering peaks. This threshold is greater than the median peak sizes we have observed, which lets us preserve the majority of called peaks.

By default, the pseudocount added to all peaks is 1. This **value (–c)** can be changed if desired.

Example: use a pseudocount of 0.1

```
-c 0.1
```

Finally, while the primary output of BBPeakCaller_TF.py is a list of peaks in BED format, an **intermediate filename (-i)** can be supplied to write information about each block and it's p-value. This file will be written in CSV format.

Example: write an intermediate file for the HCT-116 SP1-PBase dataset

```
-i HCT-116_SP1-PBase_intermediate.csv
```

21 As before, TF-directed CCF files can also be used to create insertion density tracks, following the instructions in Step 13.

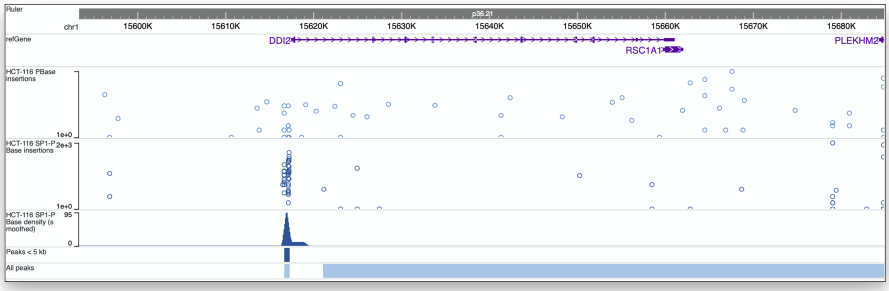
22 Let's put this all together. This command calls peaks from SP1-PBase at a false discovery rate of 5%, merging significant blocks within 250 bp, refining block edges, and outputing all peaks less than 5 kb in length:

```
python BBPeakCaller_TF.py -a 0.05 -m fdr_bh -d 250 -r -x 5000 HCT-116_SP1-PBase.ccf HCT-116_SP1-PBase.blocks HCT-116_PBase.ccf HCT-116_SP1-PBase_peaks.bed
```

This generates a BED file containing around 5600 peaks.

```
❯ wc -l HCT-116_SP1-PBase_peaks.bed
5615 HCT-116_SP1-PBase_peaks.bed
```

23 Here is the output of our sample analysis as visualized on the WashU Epigenome Browser. The top track are the undirected insertions, followed by the SP1-directed insertions. The next track is the per-block insertion densities for the SP1-PBase data with in-browser smoothing (3 px). Finally, we plot peak boundaries at a variety of p-value thresholds, in order of decreasing stringency. Notice how peaks grow, merge, shrink, and vanish at different cutoffs. The dark blue peaks track corresponds to the threshold used in the previous step. The dark blue track shows all significant peaks at 5% FDR less then 5 kb in length. The light blue track shows all peaks without size restriction. Notice how imposing a maximum peak size filters out potentially artifactual peaks.



Final Thoughts

24 BBPeakCaller_TF.py can also be used to call differential peaks between two different undirected or TF-directed calling card datasets, such as between cell types or treatments. In this use case, one sample serves as the "background" for the other. For example, let's imagine that we have done one undirected (a.k.a. BRD4) calling cards experiment on cells treated with DMSO and another treated with dexamethasone. To identify peaks that are enriched in the dexamethasone condition, we could call:

```
python BBPeakCaller_TF.py -p 9 -d 12500 DEX.ccf DEX.blocks DMSO.ccf DEX_peaks.bed
```

Since these tests are one-tailed, to find peaks that are enriched in the other direction, i.e. in the DMSO condition, we simply swap the datasets:

```
python BBPeakCaller_TF.py -p 9 -d 12500 DMSO.ccf DMSO.blocks DEX.ccf DMSO_peaks.bed
```

25 The output of these peak calling scripts are BED files listing peak coordinates only. They do not annotate the peaks themselves. There are a number of programs for secondary analyses:

- Peaks can be annotated with overlapping and nearest genes using HOMER.
- Peaks can be connected to putative genetic targets using GREAT.
- *De novo* motif analysis on peaks can be done with either HOMER or MEME.

26 For guidance on how to visualize calling card data, see our companion protocol. Documentation is also available from the WashU Epigenome Browser.

This is an open access protocol distributed under the terms of the **Creative Commons Attribution License** (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited