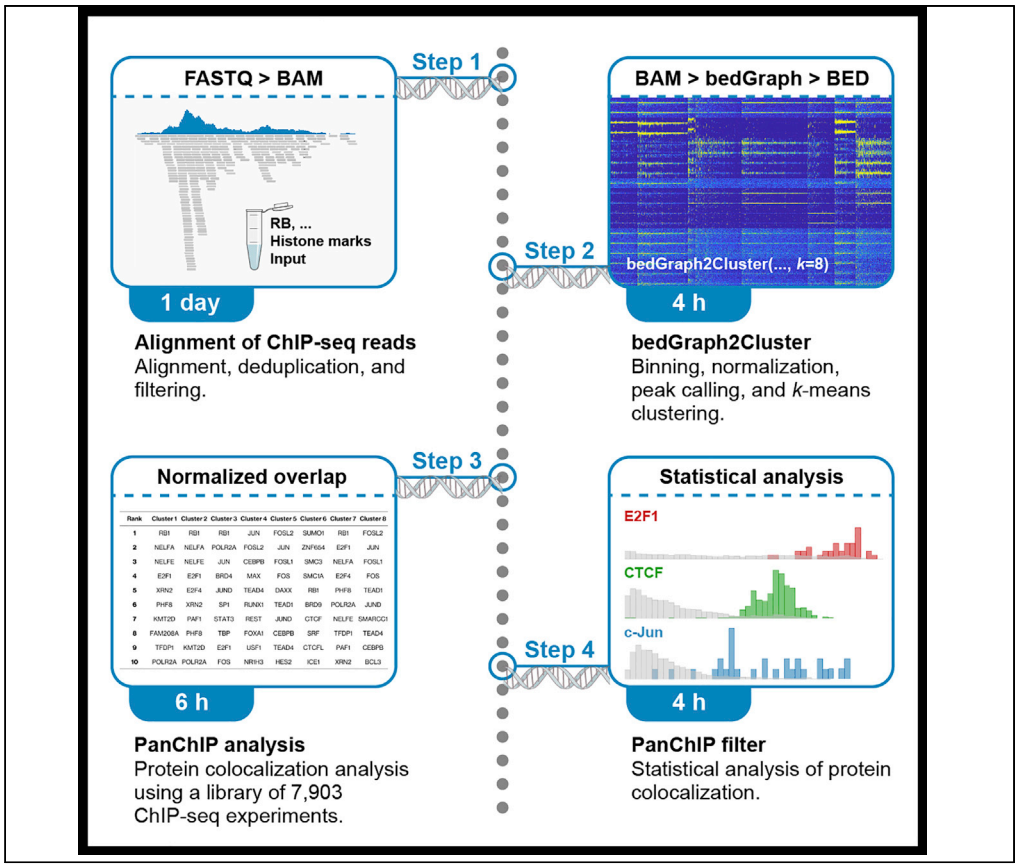
STAR Protocols

CellPress
OPEN ACCESS

Protocol

# Chromatin-bound protein colocalization analysis using bedGraph2Cluster and PanChIP




Computational pipelines for chromatin immunoprecipitation sequencing analysis can neglect colocalization events that occur in a mere subset of the genome. Here, we detail a streamlined approach for assessing colocalization of chromatin-bound proteins using the bedGraph2Cluster and PanChIP algorithms. Using histone modifications as an example, bedGraph2Cluster performs clustering analysis on chromatin binding patterns of target proteins. PanChIP then compares these clusters with a reference library of chromatin binding patterns and measures the overlap in peaks, capturing the heterogeneity in chromatin binding and colocalization patterns.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Hanjun Lee, Ioannis Sanidas, Nicholas J. Dyson, Michael S. Lawrence

hanjun@mit.edu (H.L.)
mslawrence@mgh.harvard.edu (M.S.L.)

## Highlights

Protocol identifies protein colocalization events that occur in a subset of genome

*k*-means clustering and a library of ChIP experiments enable colocalization analysis

bedGraph2Cluster implements normalization, peak calling, and *k*-means clustering

PanChIP assesses the overlap between query peaks and 7,903 ChIP-seq experiments

Lee et al., STAR Protocols 4, 101991
March 17, 2023 
https://doi.org/10.1016/j.xpro.2022.101991

STAR Protocols

CellPress
OPEN ACCESS

Protocol

# Chromatin-bound protein colocalization analysis using bedGraph2Cluster and PanChIP

Hanjun Lee,[1,2,3,5,*] Ioannis Sanidas,[1,4] Nicholas J. Dyson,[1] and Michael S. Lawrence[1,2,6,*]

[1]Massachusetts General Hospital Cancer Center, Harvard Medical School, 149 13th Street, Charlestown, MA 02129, USA

[2]Broad Institute of MIT and Harvard, 415 Main Street, Cambridge, MA 02142, USA

[3]Department of Biology, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

[4]Department of Molecular and Cellular Biology, Roswell Park Comprehensive Cancer Center, Elm & Carlton Streets, Buffalo, NY 14203, USA

[5]Technical contact

[6]Lead contact

*Correspondence: hanjun@mit.edu (H.L.), mslawrence@mgh.harvard.edu (M.S.L.)
https://doi.org/10.1016/j.xpro.2022.101991

## SUMMARY

**Computational pipelines for chromatin immunoprecipitation sequencing analysis can neglect colocalization events that occur in a mere subset of the genome. Here, we detail a streamlined approach for assessing colocalization of chromatin-bound proteins using the bedGraph2Cluster and PanChIP algorithms. Using histone modifications as an example, bedGraph2Cluster performs clustering analysis on chromatin binding patterns of target proteins. PanChIP then compares these clusters with a reference library of chromatin binding patterns and measures the overlap in peaks, capturing the heterogeneity in chromatin binding and colocalization patterns.**

**For complete details on the use and execution of this protocol, please refer to Sanidas et al. (2022).[1]**

## BEFORE YOU BEGIN

Here we describe a protocol for colocalization analysis of chromatin-bound proteins using an example set of Chromatin Immunoprecipitation Sequencing (ChIP-seq) data. The main focus of this protocol is to identify protein colocalization events that occur in subsets of the human genome and to predict other colocalization partners therein to uncover the heterogeneity of protein binding in a ChIP-seq profile. The protocol below specifically describes colocalization analysis of chromatin-bound RB, which we showed to target three fundamentally different types of genomic loci (E2F-promoters, cell-type-specific enhancers, and CTCF-insulators).[1] However, the protocol can also be applied to other chromatin-bound proteins for colocalization analyses. The protocol requires a Linux-based computational cluster and ChIP-seq datasets for both the immunoprecipitated protein and the input DNA.

### Preparation of the computational cluster

Timing: 30 min

1. Download the human reference genome FASTA from the Broad Institute's public genome references via the Google Cloud Platform:

```
{Bash}
```

```
  > mkdir -p genome && cd genome

# hg19 reference genome (this protocol)

  > wget https://storage.googleapis.com/gcp-public-data–broad-references/hg19/v0/Homo_sapiens_assembly19.fasta && cd ../

# hg38 reference genome

  > wget https://storage.googleapis.com/gcp-public-data–broad-references/hg38/v0/Homo_sapiens_assembly38.fasta && cd ../
```

*Note:* While this protocol utilizes the hg19 genome assembly to reproduce the figures and findings of Sanidas et al.,[1] we recommend the hg38 genome assembly for most use cases, as the library of ChIP-seq experiments that are utilized in the PanChIP algorithm is based on the hg38 genome assembly.

2. Install build-essentials, trim-galore, FastQC, Burrows-Wheeler Aligner (BWA), samtools, bedtools, and git:

{Bash}

```
# Debian/Ubuntu

  > sudo apt-get update && sudo apt-get install build-essential procps curl file trim-galore fastqc bwa samtools bedtools git-all

# Rocky/Fedora/RHEL

  > sudo dnf makecache --refresh && sudo dnf install gcc gcc-c++ kernel-devel make procps-ng curl file trim-galore fastqc bwa samtools bedtools git-all
```

3. Install python3-pip for facilitated installation of python packages indexed in the Python Package Index (PyPI):

{Bash}

```
# Debian/Ubuntu

  > sudo apt-get install python3-pip

# Rocky/Fedora/RHEL

  > sudo dnf install python3-pip python3-wheel
```

4. Install cutadapt and deeptools via the Python Package Index:

{Bash}

```
> pip3 install cutadapt deeptools
```

5. Install sambamba via homebrew:

{Bash}

```
> /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

> brew install brewsci/bio/sambamba
```

*Note:* If homebrew is already installed in the computational cluster, the first line in the example is unnecessary.

### Installation of run_matlab

Timing: 10 min

To enable running MATLAB-based algorithms at the Linux terminal and without having a MATLAB license, we provide a python package named run_matlab that utilizes the MATLAB Compiler Runtime (MCR) and standalone executables for simplified deployment.

6. Install run_matlab via the Python Package Index:

{Bash}

```
> pip3 install run_matlab
```

7. Create your own run_matlab directory and install the MATLAB Compiler Runtime (MATLAB version R2018b, MCR version 9.5) therein:

{Bash}

```
> run_matlab install -d run_matlab -v R2018b -r 9.5
```

a. **-d**: run_matlab directory. run_matlab will compile standalone executables using the installed data located in this directory. Set as "run_matlab" in the example.
b. **-v**: MATLAB version. Set as "R2018b" in the example.
c. **-r**: MATLAB Compiler Runtime version. Set as "9.5" in the example.

⚠ CRITICAL: MATLAB's standalone executables require MATLAB Compiler Runtime versions to be identical between the developer and the user. Standalone executables for this protocol were compiled using MATLAB version R2018b and MCR version 9.5. Hence, we guide readers to refrain from making modifications to the -v and -r flags from the example.

*Note:* run_matlab can be utilized for other Linux-compatible MATLAB standalone executables as well. For these applications, users should have the information on either the MATLAB version or the MCR version. Conversion table for MATLAB and MCR versions is available at the README markdown file in the run_matlab GitHub repository. run_matlab supports multiple simultaneous MATLAB Compiler Runtime installations. To run a Linux-compatible MATLAB standalone executable using run_matlab, run the following code in the terminal:

{Bash}

```
> run_matlab run -d run_matlab -v R2018b -r 9.5 function_dir function_name function_args
```

d. **-d**: run_matlab directory. Set as "run_matlab" in the example.
e. **-v**: MATLAB version. Set as "R2018b" in the example.
f. **-r**: MATLAB Compiler Runtime version. Set as "9.5" in the example.
g. **function_dir**: directory wherein the MATLAB standalone executable is located.
h. **function_name**: name of the MATLAB function.
i. **function_args**: space-delimited list of input arguments for the MATLAB function.

### Installation of bedGraph2Cluster

Timing: 5 min

In this protocol, we utilize bedGraph2Cluster,[1] a MATLAB-based algorithm for ChIP-seq normalization, peak calling, and clustering, for the chromatin-bound RB colocalization analysis.

8. Download its source code from the bedGraph2Cluster GitHub repository:

{Bash}

```
> git clone https://github.com/hanjunlee21/bedgraph2cluster
```

9. Install gdown via the Python Package Index:

{Bash}

```
> pip3 install gdown
```

10. bedGraph2Cluster requires a non-overlapping 200-bp-binned human reference genome file as a prerequisite. Download the premade file using gdown:

{Bash}

```
# hg19 reference genome (this protocol)

  > mkdir -p bed && gdown -O bed/hg19.200bp.bed.gz https://drive.google.com/uc?id=1chMyuUAK3rycgAbj1P-FnWhJkW7K0q0H&export=download&confirm=t

  > gunzip bed/hg19.200bp.bed.gz

# hg38 reference genome

  > mkdir -p bed && gdown -O bed/hg38.200bp.bed.gz https://drive.google.com/uc?id=1kfVOx78xTBoJHC7EL_HnkCFiWxdrNDal&export=download&confirm=t

  > gunzip bed/hg38.200bp.bed.gz
```

*Note:* While this protocol provides users with premade 200-bp-binned reference genome files for only the hg19 and hg38 reference genomes, users can also create their own custom binned reference genome file using bedtools makewindows. The size of the bin should be 200 bp, regardless of the reference genome utilized, as it is unusual for ChIP-seq experiments to exceed 200-bp resolution, and as bin sizes smaller than 200 bp will require excessive memory usage that will incur memory allocation failures in computational clusters. The 200 bp convention is adopted by many bioinformatics algorithms for similar reasons.[2]

### Installation of PanChIP

**Timing: 10 min**

For the prediction of colocalization partners in each cluster of ChIP-seq peaks, we utilize PanChIP, a python- and bash-based algorithm for pan-ChIP-seq protein colocalization analysis. PanChIP assesses the overlap between peak sets from the user-provided ChIP-seq data and a PanChIP library consisting of 7,903 ChIP-seq experiments on 915 DNA-bound proteins.[1]

11. Install PanChIP via the Python Package Index. Troubleshooting 1.

{Bash}

```
> pip3 install panchip
```

*Note:* The manual for the PanChIP software is available for download at the PanChIP GitHub Repository.

12. Download the PanChIP library using PanChIP init:

{Bash}

```
> panchip init panchip
```

⚠ CRITICAL: Job management in PanChIP is done through the jobs function of Linux. Without the function, PanChIP may exhibit excessive CPU and RAM usage. When running the pipeline with workload managers, such as SLURM, memory-optimized nodes can be utilized to avoid memory allocation failures. With an intact jobs function, the following code should return integer values as stdout in the terminal (e.g., 0):

{Bash}

```
> jobs -r | wc -l
```

## KEY RESOURCES TABLE

| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
|---|---|---|
| Deposited data | | |
| Human reference genome assembly (hg19) | Church et al.[3] | https://storage.googleapis.com/gcp-public-data–broad-references/hg19/v0/Homo_sapiens_assembly19.fasta |
| Human reference genome assembly (hg38) | Schneider et al.[4] | https://storage.googleapis.com/gcp-public-data–broad-references/hg38/v0/Homo_sapiens_assembly38.fasta |
| ChIP sequencing data | Sanidas et al.[1] | GEO accession: GSE176035 |
| Experimental models: Cell lines | | |
| hTERT-RPE1 human epithelial cells | Dr. David Pellman's laboratory, Dana Farber Cancer Institute | RRID: CVCL_4388 |
| Software and algorithms | | |
| Python (3.8.10) | N/A | https://www.python.org/ |
| MATLAB (R2018b) | MathWorks | https://www.mathworks.com/products/matlab.html |
| MATLAB Compiler Runtime (9.5) | MathWorks | https://www.mathworks.com/products/compiler/matlab-runtime.html |
| run_matlab (1.0.12) | This paper | https://doi.org/10.5281/zenodo.7017459 |
| git (2.25.1) | Linus Torvalds | https://github.com/git/git |
| cutadapt (2.8) | Martin[5] | https://github.com/marcelm/cutadapt/ |
| FastQC (0.11.9) | Simon Andrews | https://github.com/s-andrews/FastQC |
| trim-galore (0.6.4_dev) | Felix Krueger | https://github.com/FelixKrueger/TrimGalore |
| Burrows-Wheeler Aligner (0.7.17-r1188) | Li et al.[6] | http://bio-bwa.sourceforge.net/bwa.shtml |
| Python3-pip (21.3.1) | Python Package Index | https://pypi.org/ |
| gdown (4.4.0) | Kentaro Wada | https://github.com/wkentaro/gdown |
| argparse (1.1) | N/A | https://docs.python.org/3/library/argparse.html |
| pandas (1.3.0) | N/A | https://pandas.pydata.org/ |
| numpy (1.22.3) | N/A | https://numpy.org/ |
| scipy (1.7.0) | N/A | https://scipy.org/ |
| homebrew (3.5.8) | Mike McQuaid | https://github.com/Homebrew/brew |
| samtools (1.10) | Li et al.[7] | http://www.htslib.org/ |
| htslib (1.14) | Bonfield et al.[8] | http://www.htslib.org/ |
| sambamba (0.7.1) | Tarasov et al.[9] | https://github.com/biod/sambamba |
| deeptools (3.5.1) | Ramírez et al.[2] | https://github.com/deeptools/deepTools |

(*Continued on next page*)

*Continued*

| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
|---|---|---|
| bedtools (2.27.1) | Quinlan et al.[10] | https://github.com/arq5x/bedtools2 |
| bart2 (2.0) | Wang et al.[11] | https://github.com/zanglab/bart2 |
| marge (1.0) | Wang et al.[12] | http://cistrome.org/MARGE |
| liftOver (435) | Kent et al.[13]<br>Hinrichs et al.[14] | http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/liftOver |
| bedGraph2Cluster (5.2) | This paper | https://doi.org/10.5281/zenodo.7069276 |
| PanChIP (3.0.10) | Sanidas et al.[1] | https://doi.org/10.5281/zenodo.6787997 |
| OriginPro (2022b) | Origin Lab | https://www.originlab.com/ |
| ENCODE | The ENCODE Project Consortium[15] | https://www.encodeproject.org/ |
| Cistrome Data Browser | Zheng et al.[16] | http://cistrome.org/db/ |
| Other | | |
| High-performance workstation | BOXX | https://www.boxx.com/systems |

## MATERIALS AND EQUIPMENT

This protocol includes two compute-intensive tasks and two memory-intensive tasks. For each task, we recommend the following system configurations:

- Compute-intensive tasks.
  - Minimum: 1 CPU thread, 16 GB of RAM, 128 GB of storage.
  - Recommended: 24 CPU threads, 64 GB of RAM, 512 GB of storage.
  - e.g., protein colocalization analysis using PanChIP algorithm, statistical analysis of protein colocalization using PanChIP filter.
- Memory-intensive tasks.
  - Minimum: 1 CPU thread, 64 GB of RAM, 128 GB of storage.
  - Recommended: 8 CPU threads, 128 GB of RAM, 512 GB of storage.
  - e.g., alignment of ChIP-seq reads, peak calling and *k*-means clustering using bedGraph2Cluster.

The computational pipeline presented in this protocol was tested on a high-performance workstation with the following system configurations:

- CPU: Intel Xeon E5-2680 v3 @ 2.5 GHz (12 cores, 24 threads).
- RAM: Micron 18ASF1G72PZ-2G1A2 x8 (DDR4, 64 GB).
- Storage: Western Digital WDBBGB0180HBK-NESN (HDD, 18 TB).
- Operating System: Ubuntu 20.04 LTS.

## STEP-BY-STEP METHOD DETAILS

### Alignment of ChIP-seq reads

**Timing: 1 day**

Type: Memory-intensive.

In this section, we will discuss how to trim adapter sequences, perform genome indexing, align reads to the reference genome, and filter to keep only high-quality reads. From FASTQ files, we will generate BAM files that contain high-quality deduplicated reads from the ChIP-seq experiment. While this protocol mainly discusses ChIP-seq datasets generated from a single-end short-read sequencing, we also provide users with commands that can be used for paired-end short-read sequencing experiments. This section describes a protocol that contains a memory-intensive task.

1. Trim adapter sequences from FASTQ reads using trim_galore:

{Bash}

```
# single-end reads (this protocol)

  > mkdir -p trimgalore && trim_galore --illumina --fastqc -j 8 -o trimgalore/${fileID} fastq/
${fileID}.fq.gz

# paired-end reads

  > mkdir -p trimgalore && trim_galore --illumina --fastqc -j 8 --paired -o trimgalore/${fileID}
fastq/${fileID}_R1_001.fq.gz fastq/${fileID}_R2_001.fq.gz
```

   a. **--illumina**: Illumina universal adapter sequence will be trimmed.
   b. **--fastqc**: FastQC reports will be generated for each FASTQ file after trimming.
   c. **-j**: number of CPU threads. Set as "8" in the example.
   d. **-o**: output directory: Set as "trimgalore/${fileID}" in the example.

   *Note:* ${fileID} indicates the identifier for each sample in the project and acts as a placeholder in this protocol. The variable can be replaced by the file names provided by the user, which excludes the file extension. Parallelization of the protocol can be performed by a bash script using a parallelized FOR loop on the variable.

   *Note:* If the adapter sequence is unknown, the user can remove the --illumina flag to direct the trim_galore algorithm to auto-detect the adapter sequence.

2. Open trimgalore/${fileID}/${fileID}.fq.gz_trimming_report.txt and verify that the reads with adapters were successfully recognized by the algorithm by viewing its summary table presented in lines 24–32. After trimming adapter sequences, the FASTQC report file trimgalore/${fileID}/${fileID}_trimmed_fastqc.html should show no evidence of adapter content.
3. Index the human reference genome FASTA file using the Burrows-Wheeler Aligner:

{Bash}

```
> bwa index genome/Homo_sapiens_assembly19.fasta
```

   *Note:* The index building of the reference genome only needs to be performed once and can be skipped if the index files are already available.

   *Note:* While this section assumes the pipeline to be based on the hg19 human reference genome assembly, the code can be interchangeably used for other human reference genome assemblies, including hg38 and T2T-CHM13.[17]

4. Align trimmed FASTQ reads to the human reference genome using the Burrows-Wheeler Aligner and generate coordinate-sorted BAM files using samtools:

{Bash}

```
# single-end reads (this protocol)

  > mkdir -p bam

  > bwa mem -t 8 genome/Homo_sapiens_assembly19.fasta trimgalore/${fileID}/${fileID}_trim-
med.fq.gz | samtools sort -@ 8 -o bam/${fileID}.sorted.bam -

# paired-end reads

  > mkdir -p bam
```

```
> bwa mem -t 8 genome/Homo_sapiens_assembly19.fasta trimgalore/${fileID}/${fileID}_R1_001_val_1.fq.gz trimgalore/${fileID}/${fileID}_R2_001_val_2.fq.gz | samtools sort -@ 8 -o bam/${fileID}.sorted.bam -
```

a. **-t/-@**: number of CPU threads. Set as "8" in the example.
b. **-o**: output file. Set as "bam/${fileID}.sorted.bam" in the example.

5. Index coordinate-sorted BAM files using samtools:

{Bash}

```
> samtools index -@ 8 bam/${fileID}.sorted.bam
```

a. **-@**: number of CPU threads. Set as "8" in the example.

6. Generate coordinate-sorted deduplicated BAM files using sambamba:

{Bash}

```
> sambamba sort -p -t 8 -o bam/${fileID}.sambamba_sorted.bam bam/${fileID}.sorted.bam

> sambamba markdup -p -t 8 -r bam/${fileID}.sambamba_sorted.bam bam/${fileID}.deduplicated.bam
```

a. **-p**: progress bar will be shown in stderr.
b. **-t**: number of CPU threads. Set as "8" in the example.
c. **-r**: marked duplicate reads will be removed in the output file.
d. **-o**: output file. Set as "bam/${fileID}.sambamba_sorted.bam" in the example.

⚠ CRITICAL: Deduplication in ChIP-seq alignment is a critical step that removes PCR duplicates from the aligned reads. Without deduplication, peak calling algorithms often generate false-positive peaks, which may mislead users and the scientific community. Hence, we strongly recommend users include the -r flag as in the example.

*Note:* The two coordinate-sorted BAM files (i.e., bam/${fileID}.sorted.bam, bam/${fileID}.sambamba_sorted.bam) can be removed at the user's discretion.

7. Index coordinate-sorted deduplicated BAM files using sambamba:

{Bash}

```
> samtools index -@ 8 bam/${fileID}.deduplicated.bam
```

a. **-@**: number of CPU threads. Set as "8" in the example.

8. Filter to keep only high-quality reads from the coordinate-sorted deduplicated BAM files using sambamba:

{Bash}

```
> sambamba view -F "not (duplicate) and [NM] <= 2 and mapping_quality >= 30" -f bam -t 8 -o bam/${fileID}.filtered.bam bam/${fileID}.deduplicated.bam
```

a. **-F**: filtering criteria. Set as "not (duplicate) and [NM] <= 2 and mapping_quality >= 30" in the example.
   i. not (duplicate): only deduplicated reads will pass the filter.
   ii. [NM] <= 2: only reads with edit distance[18] less than or equal to 2 will pass the filter.

iii. mapping_quality >= 30: only reads with mapping quality greater than or equal to 30 will pass the filter.

b. **-f**: format of the output file. Set as "bam" in the example.
c. **-t**: number of CPU threads. Set as "8" in the example.
d. **-o**: output file. Set as "bam/${fileID}.filtered.bam" in the example.

*Note:* The deduplicated BAM file (i.e., bam/${fileID}.deduplicated.bam) can be removed at the user's discretion.

9. Convert filtered deduplicated BAM files into bigWig files of normalized ChIP-seq read counts using deeptools. The counts per million (CPM) normalization method will be utilized for the generation of bigWig files.

{Bash}

```
> mkdir -p bigwig

> bamCoverage -b bam/${fileID}.filtered.bam –normalizeUsing CPM –outFileFormat bigwig –binSize 1 -o bigwig/${fileID}.CPM.bigWig
```

a. **-b**: input BAM file. Set as "bam/${fileID}.filtered.bam" in the example.
b. **--normalizeUsing**: normalization method. Set as "CPM" in the example.
c. **--outFileFormat**: output file format for the coverage profile. Set as "bigwig" in the example.
d. **--binSize**: size of the bins for the coverage profile in bases. Set as "1" in the example.
e. **-o**: output file. Set as "bigwig/${fileID}.CPM.bigWig" in the example.

*Note:* The primary purpose of bigWig file generation in this protocol is to visualize the ChIP-seq profile. Hence, we chose the bin size of 1 bp for BAM-to-bigWig conversion. For other purposes (i.e., generation of heatmaps), users can utilize its default value, which is set as 50 bp by the software.[2]

### Peak calling and *k*-means clustering using bedGraph2Cluster

**Timing: 4 h**

Type: Memory-intensive.

In this section, we describe the protocol for the normalization, peak calling, and clustering of ChIP-seq data using bedGraph2Cluster. Using this algorithm, users can perform k-means clustering on ChIP-seq profiles to uncover the heterogeneity in chromatin binding. This section describes a protocol that contains a memory-intensive task.

10. Convert filtered deduplicated BAM files into 200-bp-binned bedGraph count files using bedtools. Troubleshooting 2.

{Bash}

```
> mkdir -p bedgraph

> sed 's/chr//g' bed/hg19.200bp.bed > bed/hg19.200bp.nochr.bed

> bedtools coverage -a bed/hg19.200bp.nochr.bed -b bam/${fileID}.filtered.bam | awk '{if($1!~/chr/) {printf "chr"; print $0} else {print $0}}' > bedgraph/${fileID}.bedgraph
```

a. **-a**: bed file for the 200-bp bins of the human genome. Set as "bed/hg19.200bp.bed" in the example.

b. **-b**: filtered deduplicated BAM file. Set as "bam/${fileID}.filtered.bam" in the example.

*Note:* We provide users with bedGraph count files that were utilized to generate figures in this protocol. Download the bedGraph count files via gdown:

*Note:* {Bash}

```
> mkdir -p bedgraph

> gdown -O bedgraph/bedgraphs.tar.gz https://drive.google.com/uc?id=129xjcSm2B6KxIrEbngZQc6Bii9sOdUha&export=download&confirm=t

> tar -xvzf bedgraph/bedgraphs.tar.gz -C bedGraph

> rm bedgraph/bedgraphs.tar.gz
```

*Note:* While this section assumes the pipeline to be based on the hg19 human reference genome assembly, the code can be interchangeably used for other human reference genome assemblies, including hg38.

11. bedGraph2Cluster is a MATLAB-based algorithm with 11 required positional arguments (i.e., (bedGraphs_Signal, bedGraphs_Control, bedGraphs_Cluster, bedGraphs_Heatmap, outdir, bed_bin, fold_change, normalization_method, k, distance_method, and clustering_method) and 1 optional positional argument (i.e., workingdir). Perform bedGraph2Cluster runs via run_matlab:

{Bash}

```
> run_matlab run -d run_matlab -v R2018b -r 9.5 bedGraph2Cluster bedGraph2Cluster bedGraphs_Signal bedGraphs_Control bedGraphs_Cluster bedGraphs_Heatmap outdir bed_bin fold_change normalization_method k distance_method clustering_method [workingdir]
```

a. **-d**: run_matlab directory. Set as "run_matlab" in the example.
b. **-v**: MATLAB version. Set as "R2018b" in the example.
c. **-r**: MATLAB Compiler Runtime version. Set as "9.5" in the example.
d. **bedGraphs_Signal**: comma-delimited list of bedGraph files that will be surveyed for signal above baseline controls during peak calling. For most applications, this would be the ChIP-seq profiles for immunoprecipitated proteins.
e. **bedGraphs_Control**: comma-delimited list of bedGraph files to be used as controls during peak calling. For most applications, this would be the ChIP-seq profiles for the input DNA. Peaks are identified as those bins whose counts in one or more of the "signal" bedGraphs exceed those in each of the "control" bedGraphs by at least the factor specified in the fold_-change parameter.
f. **bedGraphs_Cluster**: comma-delimited list of bedGraph files to be included during k-means clustering. For most applications, these would include some or all of the "signal" bedGraphs. They may also include additional bedGraphs that were not used in identifying peaks, but that are intended to help direct the assortment of peaks into clusters sharing meaningful colocalization patterns.
g. **bedGraphs_Heatmap**: comma-delimited list of bedGraph files to be included in the graphical heatmap produced by the software. For most applications, these would include some or all of the "signal" and "cluster" bedGraphs. Files listed only in this parameter do not influence the peak calling or clustering but are merely projected alongside the clusters visualized in the heatmap as additional information.
h. **outdir**: path to the output directory.
i. **bed_bin**: path to the BED file used for binned bedGraph generation.

j. **fold_change**: threshold for the fold change of signal over control during peak calling (e.g., 2 or 4).
k. **normalization_method**: normalization method to utilize. For most applications, we recommend the QNorm method, as it provides the more robust solution for clustering analysis of samples with varying signal-to-background ratios.
   i. CPM: counts-per-million.
   ii. QNorm: two-parameter normalization with linear and exponential components.
l. **k**: number of clusters to find during *k*-means clustering. For the determination of *k*, users can start from $k = 50$ to qualitatively identify the number of distinct clusters. We recommend the final value for *k* to be in between 2 and 10 to prevent potential overfitting of the data during clustering.
m. **distance_method**: distance metric for *k*-means clustering. For clustering analyses of highly analogous data sets (e.g., biological replicate experiments or ChIP-seq profiles for different phospho-isoforms of a protein), we recommend the sqeuclidean method. In contrast, for clustering analyses of non-analogous data sets (e.g., ChIP-seq profiles for histone marks), we recommend the cosine method, which is robust to differences in the magnitude of the vector.
   i. sqeuclidean: squared Euclidean distance.
   ii. cityblock: sum of absolute differences.
   iii. cosine: one minus the cosine of the included angle between points.
   iv. correlation: one minus the sample's Pearson correlation between points.
n. **clustering_method**: clustering method to utilize. For most applications, we recommend the symmetry-collapsed profile and scalar method.
   i. 1: profile.
   ii. 2: profile and scalar.
   iii. 3: symmetry-collapsed profile and scalar.
o. **workingdir**: path to the working directory.

*Note:* bedGraph2Cluster runs can be performed on the MATLAB Command Window as well in the following format:

{MATLAB}

```
> bedGraph2Cluster(bedGraphs_Signal, bedGraphs_Control, bedGraphs_Cluster, bedGraphs_Heatmap, outdir, bed_bin, fold_change, normalization_method, k, distance_method, clustering_method, [workingdir])
```

*Note:* For the peak calling, we will utilize the fold change between the ChIP-seq profiles of the immunoprecipitated protein and the input DNA as our primary metric, and neighboring bins that belong to a broader signal will be merged.

*Note:* QNorm is a two-parameter normalization method that is inspired by S3norm.[19] While both QNorm and S3norm utilize a two-parameter normalization with linear and exponential components, they use different methods of solving for the optimal parameters. QNorm shifts and stretches each sample's histogram of counts onto the same reference curve, taken to be that of the sample listed first in the bedGraphs_Signal list.

*Note:* bedGraph2Cluster uses pseudocounts[20,21] to reduce the introduction of noise during peak calling. The algorithm assumes the top 0.5% of counts per bin as signals and the bottom 99.5% of counts per bin as potential noise. In this protocol, the threshold for counts per bin thus defined was 16 and was used as the pseudocount during the peak calling.

12. Perform normalization, peak calling, and *k*-means clustering on ChIP-sequencing data on RB and histone marks using bedGraph2Cluster. Troubleshooting 3.

{Bash}

```
> run_matlab run -d run_matlab -v R2018b -r 9.5 bedGraph2Cluster bedGraph2Cluster \

bedgraph/RB.dCDK.bedgraph,bedgraph/RB.WT.bedgraph,bedgraph/RB.S230.bedgraph,bedgraph/
RB.S249.bedgraph,bedgraph/RB.T252.bedgraph,bedgraph/RB.T356.bedgraph,bedgraph/RB.T373.
bedgraph,bedgraph/RB.S608.bedgraph,bedgraph/RB.S612.bedgraph,bedgraph/RB.S780.bedgraph,
bedgraph/RB.S788.bedgraph,bedgraph/RB.S795.bedgraph,bedgraph/RB.S807.bedgraph,bedgraph/
RB.S811.bedgraph,bedgraph/RB.T821.bedgraph,bedgraph/RB.T826.bedgraph \

bedgraph/INPUT.WT.bedgraph,bedgraph/INPUT.dCDK.bedgraph,bedgraph/notag.WT.bedgraph,
bedgraph/notag.dCDK.bedgraph \

bedgraph/RB.WT.bedgraph,bedgraph/RB.dCDK.bedgraph,bedgraph/H3K4me3.WT.bedgraph,
bedgraph/H3K4me3.dCDK.bedgraph,bedgraph/H3K4me.WT.bedgraph,bedgraph/H3K4me.dCDK.
bedgraph,bedgraph/H3K27ac.WT.bedgraph,bedgraph/H3K27ac.dCDK.bedgraph \

bedgraph/INPUT.WT.bedgraph,bedgraph/INPUT.dCDK.bedgraph,bedgraph/notag.WT.bedgraph,
bedgraph/notag.dCDK.bedgraph,bedgraph/RB.WT.bedgraph,bedgraph/RB.dCDK.bedgraph,bedgraph/
RB.S230.bedgraph,bedgraph/RB.S249.bedgraph,bedgraph/RB.T252.bedgraph,bedgraph/RB.T356.
bedgraph,bedgraph/RB.T373.bedgraph,bedgraph/RB.S608.bedgraph,bedgraph/RB.S612.bedgraph,
bedgraph/RB.S780.bedgraph,bedgraph/RB.S788.bedgraph,bedgraph/RB.S795.bedgraph,bedgraph/
RB.S807.bedgraph,bedgraph/RB.S811.bedgraph,bedgraph/RB.T821.bedgraph,bedgraph/RB.T826.
bedgraph,bedgraph/E2F1.bedgraph,bedgraph/H3K4me3.WT.bedgraph,bedgraph/H3K4me3.dCDK.
bedgraph,bedgraph/H3K4me.WT.bedgraph,bedgraph/H3K4me.dCDK.bedgraph,bedgraph/H3K27ac.WT.
bedgraph,bedgraph/H3K27ac.dCDK.bedgraph \

figure_1 bed/hg19.200bp.bed 2 QNorm 3 sqeuclidean 3 $PWD
```

a. **bedGraphs_Signal**: Set as bedGraph count files for RB ChIP-seq data in the example.
b. **bedGraphs_Control**: Set as bedGraph count files for input and no-tag ChIP-seq data in the example.
c. **bedGraphs_Cluster**: Set as bedGraph count files for RB and histone marks ChIP-seq data in the example.
d. **bedGraphs_Heatmap**: Set as bedGraph count files for RB, input, no-tag, histone marks, and E2F1 ChIP-seq data in the example.
e. **outdir**: Set as "figure_1" in the example.
f. **bed_bin**: Set as "bed/hg19.200bp.bed" in the example.
g. **fold_change**: Set as "2" in the example.
h. **normalization_method**: Set as "Qnorm" in the example.
i. **k**: Set as "3" in the example.
j. **distance_method**: Set as "sqeuclidean" in the example.
k. **clustering_method**: Set as "3" in the example.
l. **workingdir**: Set as "$PWD" in the example.

13. Evaluate output files that are generated after step 12 (Figure 1A):

{Bash}

```
> ls figure_1/*
```

a. **tiles_200_data.mat**: MATLAB variable for the entire dataset.
b. **tiles_200_data_peak.mat**: MATLAB variable for only the peak set.
c. **peaks.bed**: BED file of the peak set.
d. **peaks.clust3.${1–3}.bed**: BED file of the peak set for each cluster.
e. **clustering_heatmap.pdf**: heatmap visualizing the *k*-means clustering.
   i. dpi: 300.

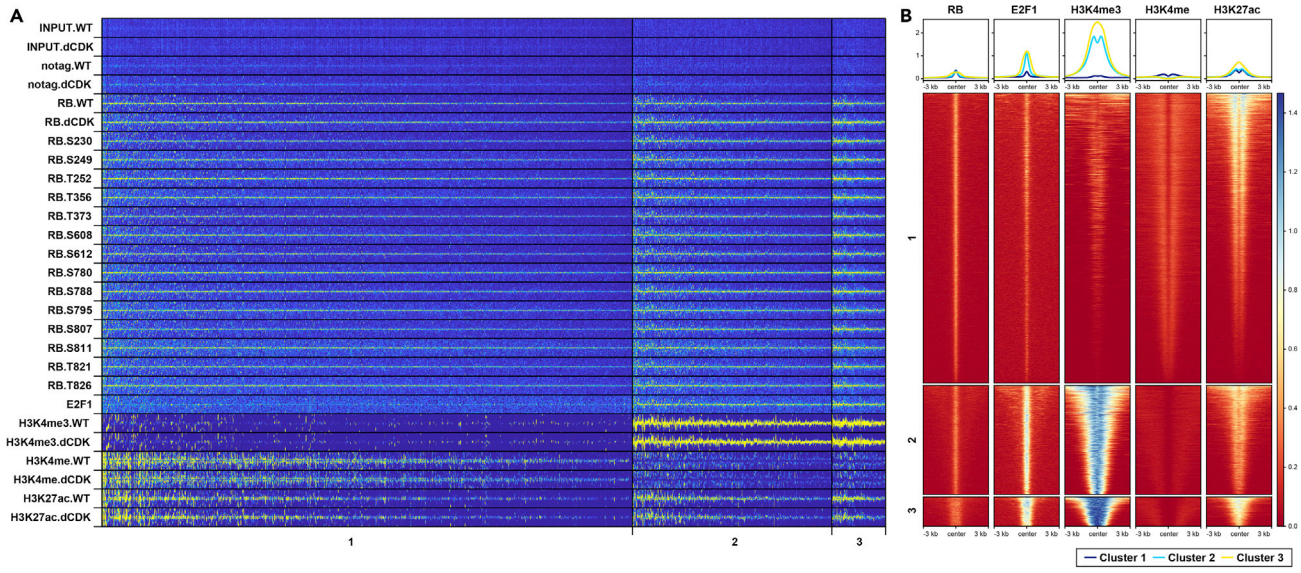



**Figure 1. Heatmap of *k*-means clustering by bedGraph2Cluster (*k* = 3)**

(A) Heatmap for RB, E2F1, histone marks, and input ChIP-seq signals for the *k*-means clusters of RB ChIP-seq peaks. All ChIP-seq profiles presented are QNorm-normalized and without pseudocounts. Cluster 1 is RB-enhancer peaks with low E2F1 signal and clusters 2–3 are RB-promoter peaks with high E2F1 signal. Rows labeled with RB followed by amino acid notations indicate monophosphorylated RB isoforms. WT, wild-type RB; dCDK, an allele of RB that is mutated in all of the CDK-consensus sites that are phosphorylated *in vivo*. Genomic regions of 10 kb centered at the peak center are presented.

(B) CPM-normalized ChIP-seq profiles for RB, E2F1, H3K4me3, H3K4me, and H3K27ac for the three *k*-means clusters of RB ChIP-seq peaks. Genomic regions of 3 kb centered at the peak center are presented. Top panel, line plots for the CPM-normalized profiles averaged over RB ChIP-seq peaks; bottom panel, heatmap for the CPM-normalized profiles. Color bar for the heatmap is presented on the right.

*Note:* Loading of tiles_200_data.mat in the MATLAB environment is a memory-intensive task. A minimum of 64 GB and a recommended 128 GB of RAM is required. For most purposes, including peak evaluations, tiles_200_data_peak.mat is a more lightweight version that would be the better fit for the task. Each file stores a container structure "X" (i.e., analogous to a class object in object-oriented programming) with the following four fields:

f. **X.samp**: contains information on the raw and normalized ChIP-seq profiles. In tiles_200_data_peak.mat, the 200-bp-binned matrices on these ChIP-seq profiles are removed to further reduce the memory consumption.
g. **X.hist**: contains information on the histogram used in Qnorm two-parameter normalization.
h. **X.peak**: contains information on the peaks and their cluster assignments.
i. **X.pixel**: contains information on the heatmap for the *k*-means clustering.

14. Compute matrix for each *k*-means cluster using deeptools:

{Bash}

```
> computeMatrix reference-point -S bigwig/RB.WT.CPM.bigWig bigwig/E2F1.CPM.bigWig bigwig/H3K4me3.WT.CPM.bigWig bigwig/H3K4me.WT.CPM.bigWig bigwig/H3K27ac.WT.CPM.bigWig -R figure_1/peaks.clust3.1.bed figure_1/peaks.clust3.2.bed figure_1/peaks.clust3.3.bed --beforeRegionStartLength 3000 --afterRegionStartLength 3000 --referencePoint center -o figure_1/heatmap.mat.gz
```

a. **-S:** bigWig files to plot. Set as bigWig files for RB, E2F1, and histone marks ChIP-seq data in the example.
b. **-R**: bed files of regions to plot. Set as bedGraph2Clutser output bed files in the example.
c. **–beforeRegionStartLength**: length of the genomic region to plot before the reference point in the unit of bp. Set as "3000" in the example.

d. **–afterRegionStartLength**: length of the genomic region to plot after the reference point in the unit of bp. Set as "3000" in the example.
e. **–referencePoint**: reference point for each bed file. Set as "center" in the example.
f. **-o**: output matrix file. Set as "figure_1/heatmap.mat.gz" in the example.

15. Visualize the computed matrix for each *k*-means cluster using deeptools (Figure 1B):

{Bash}

```
> plotHeatmap -m figure_1/heatmap.mat.gz --dpi 300 -out figure_1/heatmap.pdf
```

a. **-m:** input matrix file. Set as "figure_1/heatmap.mat.gz" in the example.
b. **–dpi**: resolution of the output heatmap file in the unit of dpi. Set as "300" in the example.
c. **-out**: output heatmap file. Set as "figure_1/heatmap.pdf" in the example.

16. Perform normalization, peak calling, and *k*-means clustering on ChIP-sequencing data on RB, E2F1, CTCF, c-Jun, and histone marks using bedGraph2Cluster. Troubleshooting 3.

{Bash}

```
> run_matlab run -d run_matlab -v R2018b -r 9.5 bedGraph2Cluster bedGraph2Cluster \

bedgraph/RB.dCDK.bedgraph,bedgraph/RB.WT.bedgraph,bedgraph/RB.S230.bedgraph,bedgraph/
RB.S249.bedgraph,bedgraph/RB.T252.bedgraph,bedgraph/RB.T356.bedgraph,bedgraph/RB.T373.
bedgraph,bedgraph/RB.S608.bedgraph,bedgraph/RB.S612.bedgraph,bedgraph/RB.S780.bedgraph,
bedgraph/RB.S788.bedgraph,bedgraph/RB.S795.bedgraph,bedgraph/RB.S807.bedgraph,bedgraph/
RB.S811.bedgraph,bedgraph/RB.T821.bedgraph,bedgraph/RB.T826.bedgraph \

bedgraph/INPUT.WT.bedgraph,bedgraph/INPUT.dCDK.bedgraph,bedgraph/notag.WT.bedgraph,
bedgraph/notag.dCDK.bedgraph \

bedgraph/RB.WT.bedgraph,bedgraph/RB.dCDK.bedgraph,bedgraph/E2F1.bedgraph,bedgraph/H3K4me3.
WT.bedgraph,bedgraph/H3K4me3.dCDK.bedgraph,bedgraph/H3K27ac.WT.bedgraph,bedgraph/H3K27ac.
dCDK.bedgraph,bedgraph/H3K4me.WT.bedgraph,bedgraph/H3K4me.dCDK.bedgraph,bedgraph/c-Jun.
shRB1.bedgraph,bedgraph/c-Jun.shSCR.bedgraph,bedgraph/CTCF.shRB1.bedgraph,bedgraph/CTCF.
shSCR.bedgraph \

bedgraph/RB.WT.bedgraph,bedgraph/RB.dCDK.bedgraph,bedgraph/H3K4me3.WT.bedgraph,bedgraph/
H3K4me3.dCDK.bedgraph,bedgraph/H3K4me.WT.bedgraph,bedgraph/H3K4me.dCDK.bedgraph,bedgraph/
H3K27ac.WT.bedgraph,bedgraph/H3K27ac.dCDK.bedgraph,bedgraph/E2F1.bedgraph,bedgraph/c-Jun.
shSCR.bedgraph,bedgraph/c-Jun.shRB1.bedgraph,bedgraph/CTCF.shSCR.bedgraph,bedgraph/CTCF.
shRB1.bedgraph,bedgraph/RB.GFP.1.bedgraph,bedgraph/RB.GFP.2.bedgraph,bedgraph/RB.A-FOS.1.
bedgraph,bedgraph/RB.A-FOS.2.bedgraph,bedgraph/RB.DNDP1.1.bedgraph,bedgraph/RB.DNDP1.2.
bedgraph \

figure_2 bed/hg19.200bp.bed 2 QNorm 8 cosine 1 $PWD
```

a. **bedGraphs_Signal**: Set as bedGraph count files for RB ChIP-seq data in the example.
b. **bedGraphs_Control**: Set as bedGraph count files for input and no-tag ChIP-seq data in the example.
c. **bedGraphs_Cluster**: Set as bedGraph count files for RB, E2F1, CTCF, c-Jun, and histone marks ChIP-seq data in the example.
d. **bedGraphs_Heatmap**: Set as bedGraph count files for RB, E2F1, CTCF, c-Jun, and histone marks ChIP-seq data in the example.
e. **outdir**: Set as "figure_2" in the example.
f. **bed_bin**: Set as "bed/hg19.200bp.bed" in the example.
g. **fold_change**: Set as "2" in the example.
h. **normalization_method**: Set as "Qnorm" in the example.
i. **k**: Set as "8" in the example.
j. **distance_method**: Set as "cosine" in the example.

k. **clustering_method**: Set as "1" in the example.
l. **workingdir**: Set as "$PWD" in the example.

17. Evaluate output files that are generated after step 16 (Figure 2A):
{Bash}

```
> ls figure_2/*
```

a. **tiles_200_data.mat**: MATLAB variable for the entire dataset.
b. **tiles_200_data_peak.mat**: MATLAB variable for only the peak set.
c. **peaks.bed**: BED file of the peak set.
d. **peaks.clust8.${1–8}.bed**: BED file of the peak set for each cluster.
e. **clustering_heatmap.pdf**: heatmap for the *k*-means clustering.
   i. dpi: 300.

18. Compute matrix for each *k*-means cluster using deeptools:
{Bash}

```
> computeMatrix reference-point -S bigwig/RB.WT.CPM.bigWig bigwig/E2F1.CPM.bigWig bigwig/CTCF.shSCR.CPM.bigWig bigwig/c-Jun.shSCR.CPM.bigWig bigwig/H3K4me3.WT.CPM.bigWig bigwig/H3K4me.WT.CPM.bigWig bigwig/H3K27ac.WT.CPM.bigWig -R figure_2/peaks.clust8.1.bed figure_2/peaks.clust8.2.bed figure_2/peaks.clust8.3.bed figure_2/peaks.clust8.4.bed figure_2/peaks.clust8.5.bed figure_2/peaks.clust8.6.bed figure_2/peaks.clust8.7.bed figure_2/peaks.clust8.8.bed --beforeRegionStartLength 3000 --afterRegionStartLength 3000 --referencePoint center -o figure_2/heatmap.mat.gz
```

a. **-S**: bigWig files to plot. Set as bigWig files for RB, E2F1, CTCF, c-Jun, and histone marks ChIP-seq data in the example.
b. **-R**: bed files of regions to plot. Set as bedGraph2Clutser output bed files in the example.

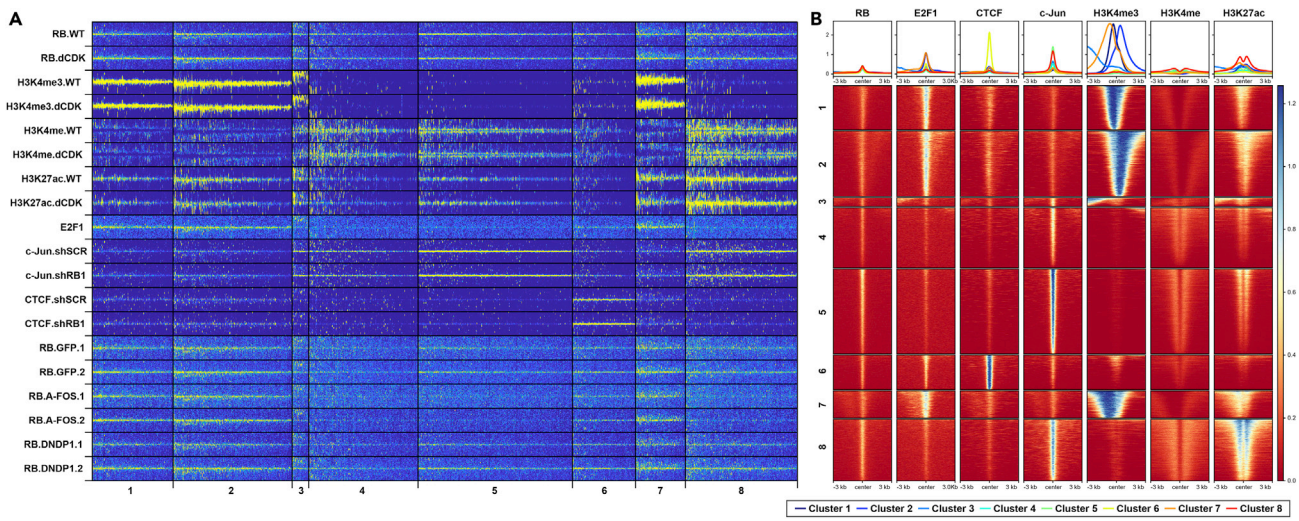



**Figure 2. Heatmap of *k*-means clustering by bedGraph2Cluster (*k* = 8)**

(A) Heatmap for RB, E2F1, CTCF, c-Jun, and histone marks ChIP-seq signals for the *k*-means clusters of RB ChIP-seq peaks. Clusters 1–3, 7 are RB-promoter peaks with high E2F1 signal, clusters 4–5, 8 are RB-enhancer peaks with high c-Jun signal, cluster 6 is RB-insulator peaks with high CTCF signal. WT, wild-type RB; dCDK, mutant allele of RB lacking CDK-phosphorylation sites. Genomic regions of 10 kb centered at the peak center are presented. (B) CPM-normalized ChIP-seq profiles for RB, E2F1, CTCF, c-Jun, H3K4me3, H3K4me1, and H3K27ac for the eight *k*-means clusters of RB ChIP-seq peaks. Genomic regions of 3 kb centered at the peak center are presented. Top panel, line plots for the CPM-normalized profiles averaged over RB ChIP-seq peaks; bottom panel, heatmap for the CPM-normalized profiles. Color bar for the heatmap is presented on the right.

c. **–beforeRegionStartLength**: length of the genomic region to plot before the reference point in the unit of bp. Set as "3000" in the example.
d. **–afterRegionStartLength**: length of the genomic region to plot after the reference point in the unit of bp. Set as "3000" in the example.
e. **–referencePoint**: reference point for each bed file. Set as "center" in the example.
f. **-o**: output matrix file. Set as "figure_2/heatmap.mat.gz" in the example.

19. Visualize the computed matrix for each *k*-means cluster using deeptools (Figure 2B):

{Bash}

```
> plotHeatmap -m figure_2/heatmap.mat.gz --dpi 300 -out figure_2/heatmap.pdf
```

a. **-m:** input matrix file. Set as "figure_2/heatmap.mat.gz" in the example.
b. **--dpi**: resolution of the output heatmap file in the unit of dpi. Set as "300" in the example.
c. **-out**: output heatmap file. Set as "figure_2/heatmap.pdf" in the example.

20. Calculate average ChIP-seq signal of RB and putative RB colocalization partners over each RB ChIP-seq peak using deeptools (Figure 3):

{Bash}

```
> mkdir -p figure_3

> multiBigwigSummary BED-file -b bigwig/RB.WT.CPM.bigWig bigwig/RB.dCDK.CPM.bigWig bigwig/
E2F1.CPM.bigWig bigwig/CTCF.shSCR.CPM.bigWig bigwig/c-Jun.shSCR.CPM.bigWig --BED figure_2/
peaks.bed -o figure_3/bigWigSummary.npz --outRawCounts figure_3/bigWigSummary.tsv
```

a. **-b:** bigWig files to calculate average ChIP-seq signals. Set as CPM-normalized bigWig files for $RB^{WT}$, $RB^{\Delta CDK}$, E2F1, CTCF, and c-Jun in the example.
b. **--BED**: bed peak file over which the ChIP-seq signals will be assessed. Set as "figure_2/peaks.bed" in the example.
c. **-o**: output compressed NPZ NumPy archive. Set as "figure_3/bigWigSummary.npz" in the example.

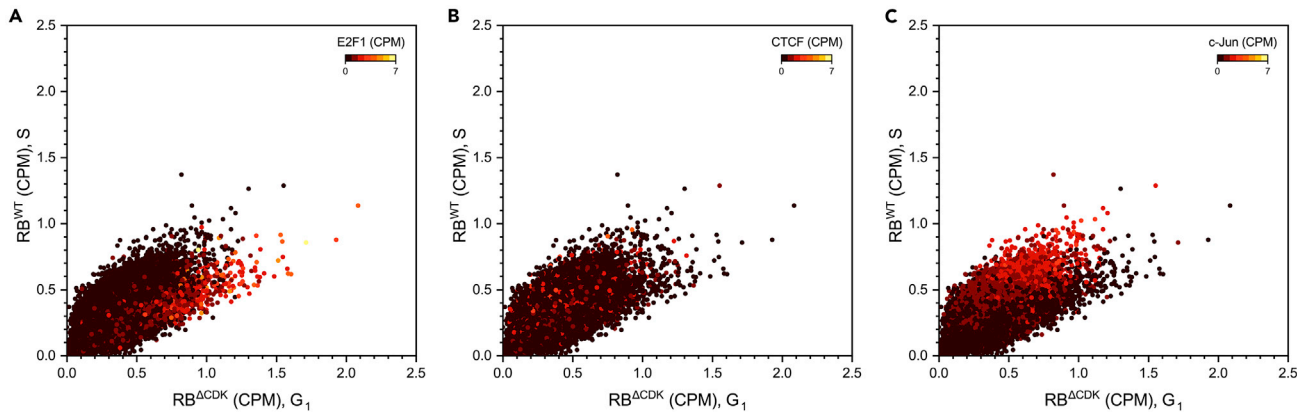



**Figure 3. Redistribution of RB upon cell-cycle-dependent activation**

(A) Scatter plot of CPM-normalized ChIP-seq signals of $RB^{\Delta CDK}$ (cells enriched in G1 phase) and wild-type RB (cells enriched in S phase) in RB ChIP-seq peaks. Color code represents CPM-normalized ChIP-seq signals for E2F1.

(B) Scatter plot of CPM-normalized ChIP-seq signals of $RB^{\Delta CDK}$ and RB wild-type in RB ChIP-seq peaks. Color code represents CPM-normalized ChIP-seq signals for CTCF.

(C) Scatter plot of CPM-normalized ChIP-seq signals of $RB^{\Delta CDK}$ and RB wild-type in RB ChIP-seq peaks. Color code represents CPM-normalized ChIP-seq signals for c-Jun. CPM, counts per million.

d. **--outRawCounts**: tab-delimited output file. Set as "figure_3/bigWigSummary.tsv" in the example.

### Protein colocalization analysis using PanChIP algorithm

Timing: 6 h

Type: Compute-intensive.

PanChIP is a pan-ChIP-seq analysis toolkit that assesses the overlap of peak sets using a library of 7,903 ChIP-seq experiments on 915 DNA-bound proteins.[1] The primary goal of a PanChIP run is to predict additional colocalization partners for a specific set of peak clusters that might otherwise be neglected in bulk analyses. The colocalization is assessed through the overlap of peak regions, with stronger weights given to peaks with greater scores. This section describes a protocol that contains a compute-intensive task.

21. PanChIP is an algorithm designed around a set of reference data based on the hg38 reference genome. To remap the peak set bed files onto the hg38 reference genome, we will utilize the UCSC liftOver software. Install UCSC liftOver via the UCSC Genome Browser utilities directory:

{Bash}

```
> mkdir -p ucsc && wget -O ucsc/liftOver http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/liftOver
```

22. Download the map.chain file required for the remapping of the data via the UCSC Genome Browser utilities directory:

{Bash}

```
> wget -O ucsc/hg19ToHg38.over.chain.gz https://hgdownload.soe.ucsc.edu/gbdb/hg19/liftOver/hg19ToHg38.over.chain.gz

> gunzip ucsc/hg19ToHg38.over.chain.gz
```

23. Remap the bed peak files generated by bedGraph2Cluster onto the hg38 reference genome using the UCSC liftOver software:

{Bash}

```
> chmod u+x ucsc/./liftOver

> mkdir -p figure_4 figure_4/input_8

> for clusterID in $(seq 1 1 8)

> do

> ucsc/./liftOver figure_2/peaks.clust8.${clusterID}.bed ucsc/hg19ToHg38.over.chain figure_4/input_8/peaks.clust8.${clusterID}.bed /dev/null

> done
```

*Note:* Steps 21–23 are unnecessary if the alignment and bedGraph2Cluster runs were performed on the hg38 genome assembly. These steps were implemented merely to reproduce the figures and findings of Sanidas et al.,[1] which utilized the hg19 genome assembly for the alignment and the *k*-means clustering.

24. Concatenate bed peak files for clusters that exhibit similar ChIP-seq profiles in the heatmap:
{Bash}

```
> mkdir -p figure_4/input_3

> cat figure_4/input_8/peaks.clust8.1.bed figure_4/input_8/peaks.clust8.2.bed figure_4/input_8/peaks.clust8.3.bed figure_4/input_8/peaks.clust8.7.bed > figure_4/input_3/RB.promoters.bed

> cat figure_4/input_8/peaks.clust8.6.bed > figure_4/input_3/RB.insulators.bed

> cat figure_4/input_8/peaks.clust8.4.bed figure_4/input_8/peaks.clust8.5.bed figure_4/input_8/peaks.clust8.8.bed > figure_4/input_3/RB.enhancers.bed
```

a. **RB.promoters.bed**: RB ChIP-seq peaks with H3K27ac, H3K4me3, and E2F1 signals.
b. **RB.insulators.bed**: RB ChIP-seq peaks with CTCF signals.
c. **RB.enhancers.bed**: RB ChIP-seq peaks with H3K27ac, H3K4me, and c-Jun signals.

25. Perform PanChIP analysis on the remapped bed peak files. Troubleshooting 4.
{Bash}

```
> panchip analysis -t 24 -r 3 panchip figure_4/input_3 figure_4/analysis_3

> panchip analysis -t 24 -r 3 panchip figure_4/input_8 figure_4/analysis_8
```

a. **-t**: number of CPU threads. Set as "24" in the example.
b. **-r**: number of repeated iterations. Set as "3" in the example.
c. **panchip**: directory wherein the PanChIP library was stored.
d. **figure_4/input_${3/8}** : input directory wherein bed peak files are located.
e. **figure_4/analysis_${3/8}**: output directory wherein output files will be stored.

⚠ CRITICAL: The input directory should only contain bed peak files. Please remove files other than the input bed peak files before running each PanChIP run.

*Note:* PanChIP utilizes the 5$^{th}$ column of each bed peak file as the weight for the calculation of the overlap. We strongly recommend that users utilize a standard BED6 format that denotes a score between 0 and 1,000 in the files' 5$^{th}$ column.[13] Non-numeric values are not accepted by the software. The bed peak files generated by bedGraph2Cluster, which this protocol uses as the input for PanChIP analysis, follow the standard BED6 convention.

*Note:* While a single iteration of PanChIP run provides users with a great resource for preliminary analysis, we recommend that users utilize at least three repeated iterations for publication purposes per the PanChIP guidelines.[1]

26. Evaluate output files that are generated after step 25:
{Bash}

```
> ls figure_4/analysis_?/*.tsv
```

a. **figure_4/analysis_${3/8}/primary.output.tsv**: tab-delimited matrix of normalized overlap between input bed peak files and peak files from the PanChIP library consisting of 7,903 ChIP-seq experiments on 915 DNA-bound proteins.

*Note:* Among *k*-means clusters with $k = 3$, cluster 1 corresponds to "non-promoters" cluster of Sanidas et al.[1] Clusters 2 and 3 correspond to "promoters-2" and "promoters-1", respectively. Among k-means clusters with $k = 8$, clusters 1 and 2 correspond to "A–B" clusters of Sanidas

et al. Clusters 3 and 7 correspond to "C–D" clusters, clusters 4, 5, and 8 to "F–H" clusters, and cluster 6 to "E" cluster of Sanidas et al. Clusters 1–3 and 7 represent RB-bound promoter peaks, clusters 4, 5, and 8 represent RB-bound enhancer peaks, and cluster 6 represents RB-bound insulator peaks.

*Note:* Normalized overlap is a metric that represents positive library-level colocalization in PanChIP. The mathematical definition of normalized overlap was previously described[1] and is also included in the PanChIP manual available for download at the PanChIP GitHub repository.

### Statistical analysis of protein colocalization using PanChIP filter

**Timing: 4 h**

Type: Compute-intensive.

While the normalized overlap calculated by the PanChIP algorithm in the previous section provides users with the primary metric that enables preliminary assessment on the colocalization, the metric alone lacks the statistical significance measures that would allow users to assess the significance of the colocalization. PanChIP filter allows users to perform statistical analysis on these colocalization events, providing users with Bonferroni-corrected p-values calculated from a two-tailed Welch's t-test and signal-to-noise ratios. The PanChIP guideline on these measures is as follows: Bonferroni-corrected p-value < 0.05 and signal-to-noise ratio > 2.[1] This section describes a protocol that contains a compute-intensive task.

*Note:* The mathematical description of the statistical analysis, including the definition on the signal-to-noise ratio and the details on the two-tailed Welch's *t*-test that was utilized in the analysis, was previously described[1] and is also included in the PanChIP manual available for download at the PanChIP GitHub repository.

27. Perform PanChIP filter on the remapped bed peak files. Troubleshooting 5.

{Bash}

```
> mkdir -p figure_4/filter_3 figure_4/filter_8
> for file in RB.promoters RB.insulators RB.enhancers
> do
> panchip filter -t 24 panchip figure_4/input_3/${file}.bed figure_4/filter_3/${file}
> done
> for clusterID in $(seq 1 1 8)
> do
> panchip filter -t 24 panchip figure_4/input_8/peaks.clust8.${clusterID}.bed figure_4/filter_8/ peaks.clust8.${clusterID}
> done
```

a. **-t**: number of CPU threads. Set as "24" in the example.
b. **panchip**: directory wherein the PanChIP library was stored.
c. **figure_4/input_${3/8}/*.bed**: input bed peak files of which statistical significance for each overlap will be measured.
d. **figure_4/filter_${3/8}/***: output directory wherein output files will be stored.

28. Evaluate output files that are generated after step 27 (Figure 4):
{Bash}

```
> ls figure_4/filter_?/*/*.tsv
```

a. **figure_4/filter_${3/8}/primary.output.tsv**: tab-delimited matrix of individual overlap between input bed peak files and peak files from the PanChIP library consisting of 7,619 ChIP-seq experiments on 639 DNA-bound proteins.
b. **figure_4/filter_${3/8}/statistics.tsv**: tab-delimited matrix of statistical measures on all 915 DNA-bound proteins in the PanChIP library.
   i. TF: gene symbol for the DNA-bound protein.
   ii. Mean: mean for the set of individual overlap values measured for each biological replicate experiment in the PanChIP library.
   iii. Standard Deviation: standard deviation for the set of individual overlap values measured for each biological replicate experiment in the PanChIP library.
   iv. Signal-to-noise ratio: mean divided by the standard deviation.
   v. Adjusted P: Bonferroni-corrected *p*-value.
   vi. Filter: whether the DNA-bound protein passes statistical significance thresholds.

*Note:* The PanChIP library for PanChIP filter is smaller compared to that used for PanChIP analysis, as it only assesses DNA-bound proteins with at least two biological replicate experiments

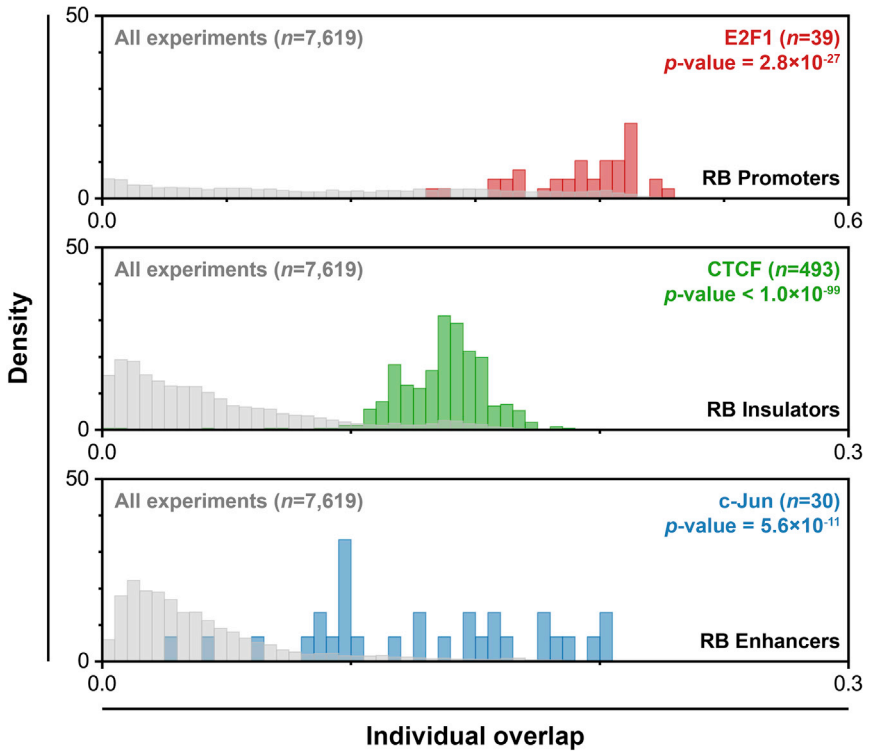



**Figure 4. Histogram of individual overlap between three groups of *k*-means clusters and E2F1, CTCF, and c-Jun**
Top panel, individual overlap between RB promoter clusters (clusters 1–3, 7) and 7,619 ChIP-seq experiments from the PanChIP library. Individual overlap is a metric that measures the overlap between the query and each ChIP-seq experiment from the PanChIP library (Sanidas et al.[1]). Center panel, individual overlap between RB insulator clusters (cluster 6) and 7,619 ChIP-seq experiments from the PanChIP library. Bottom panel, individual overlap between RB enhancer clusters (clusters 4–5, 8) and 7,619 ChIP-seq experiments from the PanChIP library. For each histogram, the distribution of individual overlaps for each RB colocalization partners is also presented with *p*-values from the two-sample two-tailed Welch's *t*-test.

available in the library for statistical analyses. figure_4/filter_${3/8}/statistics.tsv shows NA (i.e., not applicable) in rows that represent DNA-bound proteins that were filtered out when running PanChIP filter due to the lack of biological replicate experiments. figure_4/analysis_${3/8}/primary.output.tsv and figure_4/filter_${3/8}/statistics.tsv have the identical row order.

*Note:* Individual overlap is a metric that represents positive sample-level colocalization in PanChIP. The mathematical definition of individual overlap was previously described[1] and is also included in the PanChIP manual available for download at the PanChIP GitHub repository.

29. Paste the statistics.tsv file from step 27 to primary.output.tsv from step 25:

{Bash}

```
> paste figure_4/analysis_3/primary.output.tsv figure_4/filter_3/RB.promoters/statistics.tsv figure_4/filter_3/RB.insulators/statistics.tsv figure_4/filter_3/RB.enhancers/statistics.tsv > figure_4/panchip_3.tsv

> paste figure_4/analysis_8/primary.output.tsv figure_4/filter_8/ peaks.clust8.1/statistics.tsv figure_4/filter_8/ peaks.clust8.2/statistics.tsv figure_4/filter_8/ peaks.clust8.3/statistics.tsv figure_4/filter_8/ peaks.clust8.4/statistics.tsv figure_4/filter_8/ peaks.clust8.5/statistics.tsv figure_4/filter_8/ peaks.clust8.6/statistics.tsv figure_4/filter_8/ peaks.clust8.7/statistics.tsv figure_4/filter_8/ peaks.clust8.8/statistics.tsv > figure_4/panchip_8.tsv
```

a. **figure_4/panchip_?.tsv**: tab-delimited file with normalized overlap values and statistical significance measures.

⚠ CRITICAL: Interpretation of PanChIP results is done through their normalized overlap values. Sort panchip_?.tsv files on the basis of the normalized overlap column for the sample and filter to keep only rows for DNA-bound proteins that satisfy the thresholds (i.e., signal-to-noise ratio > 2 and Bonferroni-corrected *p*-value < 0.05).

30. For ENCODE-listed ChIP-seq experiments in the library, cell line information is available at the PanChIP GitHub repository and is automatically downloaded during the initialization step of PanChIP. For cell-type-specific interpretation of PanChIP results, run the following command:

{Bash}

```
> for file in RB.promoters RB.insulators RB.enhancers

> do

> paste panchip/PanChIP-v.*.*.*/lib/cellLine.tsv figure_4/filter_3/${file}/statistics.tsv > figure_4/panchip_cellLine_3_${file}.tsv

> done

> for clusterID in $(seq 1 1 8)

> do

> paste panchip/PanChIP-v.*.*.*/lib/cellLine.tsv figure_4/filter_8/peaks.clust8.${clusterID}/primary.output.tsv > figure_4/panchip_cellLine_8_peaks.clust8.${clusterID}.tsv

> done
```

⚠ CRITICAL: Interpretation of cell-type-specific PanChIP results is done through their individual overlap values. Filter rows that represent ChIP-seq experiments from the cell

type of interest and sort panchip_cellLine_*.tsv files on the basis of the individual overlap column for the sample.

*Note:* The order of the cellLine.tsv file is identical to that of the primary.output.tsv generated by PanChIP filter.

## EXPECTED OUTCOMES

This protocol utilizes a ChIP-seq dataset of RB, histone marks, and several of RB's putative colocalization partners (i.e., E2F1, CTCF, and c-Jun) and performs alignment, normalization, peak calling, *k*-means clustering, colocalization analysis, and statistical analysis using bedGraph2Cluster and PanChIP. In this section, we will describe an example application of this protocol that led to the discovery of RB's cell-cycle-dependent redistribution and colocalization with insulators and cell-type-specific enhancers.[1]

### Chromatin-bound protein colocalization analysis

In the protein colocalization analysis using PanChIP algorithm and statistical analysis of protein co-localization using PanChIP filter sections, we calculated the normalized overlap between ChIP-seq profiles of our experimental data and the PanChIP library consisting of 7,903 ChIP-seq experiments on 915 DNA-bound proteins. We performed PanChIP runs on both the raw *k*-means clusters generated by bedGraph2Cluster ($k = 8$) and the three groups of clusters that exhibited qualitatively similar ChIP-seq profiles (i.e., RB.promoters, RB.insulators, and RB.enhancers). We sorted normalized overlap in decreasing order and filtered out all rows for DNA-bound proteins that failed to satisfy the PanChIP thresholds (signal-to-noise ratio > 2 and Bonferroni-corrected $p$-value < 0.05). We confirmed our previous observation that RB binds to three fundamentally different types of loci (E2F1 promoters, CTCF insulators, and cell-type-specific enhancers) and further identified a plethora of DNA-bound proteins that colocalize with RB in each cluster of RB ChIP-seq regions (Table 1). These included proteins associated with RNA polymerase II-dependent transcription for clusters 1–3, 7 (i.e., POLR2A, NELFA, NELFE, and XRN2), with YAP/TAZ-TEAD pathway for clusters 4, 5, 8 (i.e., TEAD1 and TEAD4), and with cohesin components for cluster 6 (i.e., SMC1A and SMC3).

For a detailed view of the overlap between RB and its putative colocalization partners, we plotted the distribution of individual overlap between the three group of clusters (i.e., RB.promoters, RB.insulators, and RB.enhancers) and the three putative colocalization partners (i.e., E2F1, CTCF, and c-Jun), respectively (Figure 4). The null distribution for each statistical analysis is the distribution of individual overlap between each group of clusters and the 7,619 ChIP-seq experiments. Two-sample two-tailed Welch's *t*-test was utilized to assess the statistical significance of each overlap.

### Capturing redistribution in ChIP-seq datasets

In the peak calling and *k*-means clustering using bedGraph2Cluster section, we calculated the average ChIP-seq signal over each of the 28,115 RB ChIP-seq peaks for a few isoforms of RB

**Table 1. Top ten predicted colocalization partners of RB in each *k*-means cluster ($k = 8$)**

| Rank | Cluster 1 | Cluster 2 | Cluster 3 | Cluster 4 | Cluster 5 | Cluster 6 | Cluster 7 | Cluster 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | RB1 | RB1 | RB1 | JUN | FOSL2 | SUMO1 | RB1 | FOSL2 |
| 2 | NELFA | NELFA | POLR2A | FOSL2 | JUN | ZNF654 | E2F1 | JUN |
| 3 | NELFE | NELFE | JUN | CEBPB | FOSL1 | SMC3 | NELFA | FOSL1 |
| 4 | E2F1 | E2F1 | BRD4 | MAX | FOS | SMC1A | E2F4 | FOS |
| 5 | XRN2 | E2F4 | JUND | TEAD4 | DAXX | RB1 | PHF8 | TEAD1 |
| 6 | PHF8 | XRN2 | SP1 | RUNX1 | TEAD1 | BRD9 | POLR2A | JUND |
| 7 | KMT2D | PAF1 | STAT3 | REST | JUND | CTCF | NELFE | SMARCC1 |
| 8 | FAM208A | PHF8 | TBP | FOXA1 | CEBPB | SRF | TFDP1 | TEAD4 |
| 9 | TFDP1 | KMT2D | E2F1 | USF1 | TEAD4 | CTCFL | PAF1 | CEBPB |
| 10 | POLR2A | POLR2A | FOS | NR1H3 | HES2 | ICE1 | XRN2 | BCL3 |

(i.e., $RB^{WT}$ and $RB^{\Delta CDK}$) and their putative colocalization partners (i.e., E2F1, CTCF, and c-Jun) that we previously identified.[1] $RB^{\Delta CDK}$ is a mutant form of RB in which all known *in vivo* CDK phosphorylation sites are substituted with alanine to prevent phosphorylation by CDKs.[22–24] Hence, $RB^{\Delta CDK}$ is constitutively active in repressing transcription of genes targeted by E2F1/DP heterodimers[25] and thereby maintaining cells in G1-phase.[22,24]

To capture redistribution events that are dependent on the cell-cycle-dependent activation of RB, we compared ChIP-seq signals of $RB^{WT}$ (cells enriched in S-phase) to those of $RB^{\Delta CDK}$ (cells enriched in G1-phase) over RB ChIP-seq peaks in the form of a scatter plot (Figures 3A–3C). We color-coded each dot to indicate ChIP-seq signals of putative RB colocalization partners in each peak. We discovered that E2F1 signal in RB ChIP-seq peaks is skewed toward $RB^{\Delta CDK}$ and that c-Jun signal in RB ChIP-seq peaks is skewed toward $RB^{WT}$. CTCF signal in RB ChIP-seq peaks was largely invariant between the two isoforms. Combined with the results from additional validation experiments, this led to the discovery that RB enriches at E2F1 promoter sites in G1-phase and at cell-type-specific enhancer sites in S-phase, where it regulates MAPK signaling.[1]

## LIMITATIONS

This protocol utilizes multiple different software suites for the task of assessing colocalization of chromatin-bound proteins using ChIP-seq data. Hence, each section of this protocol is not without limitations, and these limitations mainly arise from the innate characteristics of the software.

The peak calling and *k*-means clustering using bedGraph2Cluster section requires ChIP-seq data to be aligned to the human reference genome and only allows for analysis in the non-mitochondrial chromosomal DNA. bedGraph2Cluster does not automatically filter out blacklisted regions,[26] and it is recommended to filter out any blacklisted regions from the peak set after the analysis.

The protein colocalization analysis using PanChIP algorithm and statistical analysis of protein colocalization using PanChIP filter sections utilize PanChIP, a pan-ChIP-seq analysis software for protein colocalization in human ChIP-seq profiles.[1] Model organisms other than humans are currently not supported by the pipeline. Furthermore, since statistical measures adopted by PanChIP assess how the overlap is consistently (i.e., throughout all cell types) greater compared to the null distribution, PanChIP is more effective in predicting constitutive colocalization events compared to cell-type-specific colocalization events. While we provided cell-type information to enable users to predict cell-type-specific colocalization events with the original introduction of PanChIP,[1] only ENCODE-listed ChIP-seq experiments have a well-annotated list of cell types, which makes the cell-type-specific analysis of less-studied DNA-bound proteins difficult. Finally, the job management of PanChIP primarily relies on the jobs function of Linux, of which access might be restricted for users on computational clusters with custom job management configurations. PanChIP may exhibit excessive CPU and RAM usage without an intact jobs function.

## TROUBLESHOOTING

### Problem 1

The installation of PanChIP fails due to the misrecognition of pyparsing during its gdown installation attempt (before you begin, step 11).

### Potential solution

Install pyparsing using pip3 with sudo credentials:

```
{Bash}
```

```
> sudo pip3 install pyparsing

> pip3 install panchip
```

*Note:* Discussions on this issue is also available at the PanChIP GitHub Issues page.

### Problem 2

Conversion of BAM files to bedGraph count files exceeds system's memory limit (step 10).

### Potential solution

Conversion of BAM files to bedGraph count files is a particularly memory-intensive task. For single-end short-read sequencing, 64 GB of memory would suffice. However, for paired-end short-read sequencing, system configuration with at least 128 GB of memory is recommended. A bypass for this high usage of memory is splitting BAM and 200-bp bin bed files chromosome-by-chromosome and concatenating the generated bedGraph count files in the following order: chr1, chr2, chr3, chr4, chr5, chr6, chr7, chr8, chr9, chr10, chr11, chr12, chr13, chr14, chr15, chr16, chr17, chr18, chr19, chr20, chr21, chr22, chrX, and chrY.

- Split filtered deduplicated BAM and 200-bp bin bed files chromosome-by-chromosome:

{Bash}

```
> mkdir -p bed/hg19

> bedgraphs=""

> for chr in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y

> do

> awk '{if($1=="${chr}") {print}}' bed/hg19.200bp.nochr.bed > bed/hg19/${chr}.bed

> samtools view -b bam/${fileID}.filtered.bam chr${chr} > bam/${fileID}.filtered.${chr}.bam

> samtools index bam/${fileID}.filtered.${chr}.bam

> bedtools coverage -a bed/hg19/${chr}.bed -b bam/${fileID}.filtered.${chr}.bam | awk '{if($1!~/chr/) {printf "chr"; print $0} else {print $0}}' > bedgraph/${fileID}.chr${chr}.bedgraph

> rm bam/${fileID}.filtered.${chr}.bam bam/${fileID}.filtered.${chr}.bam.bai

> bedgraphs="${bedgraphs} bedgraph/${fileID}.chr${chr}.bedgraph"

> done
```

- Concatenate the generated bedGraph count files:

{Bash}

```
> cat ${bedgraphs} > bedgraph/${fileID}.bedgraph

> rm ${bedgraphs}
```

### Problem 3

MATLAB Compiler Runtime outputs a difff failure message after running bedGraph2Cluster (steps 12 or 16).

### Potential solution

- difff failure may occur when one of the bedGraph files contains only zero values in its fourth column.
- Evaluate bedGraph count files using awk:

{Bash}

```
> awk '{sum+=$4;} END{print sum}' bedgraph/${fileID}.bedgraph
```

- This command sums all counts in a bedGraph file and outputs the total counts. If it outputs "0" for a file, that means the file contains only zero values, indicating that there is a problem.
- The generation of bedGraph files that contain only zero values may occur due to the different nomenclature for chromosomes between the UCSC and the NCBI. In this protocol, we utilized the hg19 human genome reference assembly with the UCSC chromosomal nomenclature (i.e., 1 instead of chr1).
- For genomes that follow the NCBI chromosomal nomenclature, use hg19.200bp.bed instead of hg19.200bp.nochr.bed in step 10:

{Bash}

```
> bedtools coverage -a bed/hg19.200bp.bed -b bam/${fileID}.filtered.bam | awk '{if($1!~/chr/)
{printf "chr"; print $0} else {print $0}}' > bedgraph/${fileID}.bedgraph
```

### Problem 4

PanChIP analysis generates the following error message: "cat: … no such file or directory." (step 25).

### Potential solution

This error occurs because the input directory contains files other than the bed peak files. Remove all files other than bed peaks files in the directory using the following command (any other files will be deleted, so be sure to first copy them elsewhere if they are needed):

{Bash}

```
> shopt -s extglob

> cd figure_4/input_3 && rm *.!(bed) && cd ../../

> cd figure_4/input_8 && rm *.!(bed) && cd ../../
```

### Problem 5

PanChIP filter generates the following error message: "sort: cannot read: … no such file or directory." (step 27).

### Potential solution

This error occurs because the user provided a directory or an invalid path to the bed peak file as the input for PanChIP filter. Verify that the file to which the input path directs is a bed peak file.

## RESOURCE AVAILABILITY

### Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Michael S. Lawrence (mslawrence@mgh.harvard.edu).

### Materials availability

This study did not generate new unique reagents.

## Data and code availability

Source code for run_matlab is available for download at the run_matlab GitHub repository and is archived at Zenodo [https://doi.org/10.5281/zenodo.7017459]. Source code for bedGraph2Cluster is available for download at the bedGraph2Cluster GitHub repository and is archived at Zenodo [https://doi.org/10.5281/zenodo.7030285]. The code for PanChIP was described previously[1] and is available for download at the PanChIP GitHub repository. Source code for PanChIP is also archived at Zenodo [https://doi.org/10.5281/zenodo.6787997]. ChIP sequencing data have been deposited at GEO under the accession code GSE176035. To further ensure the reproducibility of our results, we also made the processed bedGraph count files available for download at the PanChIP cloud storage. Small exemplary gzipped FASTQ file for the alignment step is also available for download at the PanChIP cloud storage.

## ACKNOWLEDGMENTS

This work was supported by NIH grants CA236538 and GM117413 to N.J.D. and by the Rullo Family Innovation Award to M.S.L.

## AUTHOR CONTRIBUTIONS

H.L., I.S., N.J.D., and M.S.L. designed the study. H.L. wrote the manuscript. H.L., I.S., N.J.D., and M.S.L. edited the manuscript. H.L. and M.S.L. designed the software and conducted bioinformatics analyses.

## DECLARATION OF INTERESTS

The authors declare no competing interests.

## REFERENCES

1. Sanidas, I., Lee, H., Rumde, P.H., Boulay, G., Morris, R., Golczer, G., Stanzione, M., Hajizadeh, S., Zhong, J., Ryan, M.B., et al. (2022). Chromatin-bound RB targets promoters, enhancers, and CTCF-bound loci and is redistributed by cell-cycle progression. Mol. Cell *82*, 3333–3349.e9. https://doi.org/10.1016/j.molcel.2022.07.014.

2. Ramírez, F., Dündar, F., Diehl, S., Grüning, B.A., and Manke, T. (2014). deepTools: a flexible platform for exploring deep-sequencing data. Nucleic Acids Res. *42*, W187–W191. https://doi.org/10.1093/nar/gku365.

3. Church, D.M., Schneider, V.A., Graves, T., Auger, K., Cunningham, F., Bouk, N., Chen, H.-C., Agarwala, R., Mclaren, W.M., Ritchie, G.R.S., et al. (2011). Modernizing reference genome assemblies. PLoS Biol. *9*, e1001091. https://doi.org/10.1371/journal.pbio.1001091.

4. Schneider, V.A., Graves-Lindsay, T., Howe, K., Bouk, N., Chen, H.-C., Kitts, P.A., Murphy, T.D., Pruitt, K.D., Thibaud-Nissen, F., Albracht, D., et al. (2017). Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. Genome Res. *27*, 849–864. https://doi.org/10.1101/gr.213611.116.

5. Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet. J. *17*, 10. https://doi.org/10.14806/ej.17.1.200.

6. Li, H., and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics *25*, 1754–1760. https://doi.org/10.1093/bioinformatics/btp324.

7. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R.; 1000 Genome Project Data Processing Subgroup (2009). The sequence alignment/map format and SAMtools. Bioinformatics *25*, 2078–2079. https://doi.org/10.1093/bioinformatics/btp352.

8. Bonfield, J.K., Marshall, J., Danecek, P., Li, H., Ohan, V., Whitwham, A., Keane, T., and Davies, R.M. (2021). HTSlib: C library for reading/writing high-throughput sequencing data. GigaScience *10*. https://doi.org/10.1093/gigascience/giab007.

9. Tarasov, A., Vilella, A.J., Cuppen, E., Nijman, I.J., and Prins, P. (2015). Sambamba: fast processing of NGS alignment formats. Bioinformatics *31*, 2032–2034. https://doi.org/10.1093/bioinformatics/btv098.

10. Quinlan, A.R., and Hall, I.M. (2010). BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics *26*, 841–842. https://doi.org/10.1093/bioinformatics/btq033.

11. Wang, Z., Civelek, M., Miller, C.L., Sheffield, N.C., Guertin, M.J., and Zang, C. (2018). BART: a transcription factor prediction tool with query gene sets or epigenomic profiles. Bioinformatics *34*, 2867–2869. https://doi.org/10.1093/bioinformatics/bty194.

12. Wang, S., Zang, C., Xiao, T., Fan, J., Mei, S., Qin, Q., Wu, Q., Li, X., Xu, K., He, H.H., et al. (2016). Modeling cis-regulation with a compendium of genome-wide histone H3K27ac profiles. Genome Res. *26*, 1417–1429. https://doi.org/10.1101/gr.201574.115.

13. Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., Zahler, A.M., and Haussler, D. (2002). The human genome browser at UCSC. Genome Res. *12*, 996–1006. https://doi.org/10.1101/gr.229102.

14. Hinrichs, A.S., Karolchik, D., Baertsch, R., Barber, G.P., Bejerano, G., Clawson, H., Diekhans, M., Furey, T.S., Harte, R.A., Hsu, F., et al. (2006). The UCSC genome browser database: update 2006. Nucleic Acids Res. *34*, D590–D598. https://doi.org/10.1093/nar/gkj144.

15. ENCODE Project Consortium (2012). An integrated encyclopedia of DNA elements in the human genome. Nature *489*, 57–74. https://doi.org/10.1038/nature11247.

16. Zheng, R., Wan, C., Mei, S., Qin, Q., Wu, Q., Sun, H., Chen, C.-H., Brown, M., Zhang, X., Meyer, C.A., et al. (2019). Cistrome data browser: expanded datasets and new tools for gene regulatory analysis. Nucleic Acids Res. *47*, D729–D735. https://doi.org/10.1093/nar/gky1094.

17. Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A.V., Mikheenko, A., Vollger, M.R., Altemose, N., Uralsky, L., Gershman, A., et al. (2022). The complete sequence of a human genome. Science *376*, 44–53. https://doi.org/10.1126/science.abj6987.

18. Levenshtein, V.I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk *163*, 845–848.

19. Xiang, G., Keller, C.A., Giardine, B., An, L., Li, Q., Zhang, Y., and Hardison, R.C.

(2020). S3norm: simultaneous normalization of sequencing depth and signal-to-noise ratio in epigenomic data. Nucleic Acids Res. *48*, e43. https://doi.org/10.1093/nar/gkaa105.

20. Zhang, Y., Liu, T., Meyer, C.A., Eeckhoute, J., Johnson, D.S., Bernstein, B.E., Nusbaum, C., Myers, R.M., Brown, M., Li, W., and Liu, X.S. (2008). Model-based analysis of ChIP-seq (MACS). Genome Biol. *9*, R137. https://doi.org/10.1186/gb-2008-9-9-r137.

21. Liu, T. (2014). Use model-based analysis of ChIP-seq (MACS) to analyze short reads generated by sequencing protein–DNA interactions in embryonic stem cells. In Methods in Molecular Biology (Springer), pp. 81–95. https://doi.org/10.1007/978-1-4939-0512-6_4.

22. Sanidas, I., Morris, R., Fella, K.A., Rumde, P.H., Boukhali, M., Tai, E.C., Ting, D.T., Lawrence, M.S., Haas, W., and Dyson, N.J. (2019). A code of mono-phosphorylation modulates the function of RB. Mol. Cell *73*, 985–1000.e6. https://doi.org/10.1016/j.molcel.2019.01.004.

23. Narasimha, A.M., Kaulich, M., Shapiro, G.S., Choi, Y.J., Sicinski, P., and Dowdy, S.F. (2014). Cyclin D activates the Rb tumor suppressor by mono-phosphorylation. Elife *3*, e02872. https://doi.org/10.7554/elife.02872.

24. Krishnan, B., Yasuhara, T., Rumde, P., Stanzione, M., Lu, C., Lee, H., Lawrence, M.S., Zou, L., Nieman, L.T., Sanidas, I., and Dyson, N.J. (2022). Active RB causes visible changes in nuclear organization. J. Cell Biol. *221*, e202102144. https://doi.org/10.1083/jcb.202102144.

25. Dyson, N. (1998). The regulation of E2F by pRB-family proteins. Genes Dev. *12*, 2245–2262. https://doi.org/10.1101/gad.12.15.2245.

26. Amemiya, H.M., Kundaje, A., and Boyle, A.P. (2019). The ENCODE blacklist: identification of problematic regions of the genome. Sci. Rep. *9*, 9354. https://doi.org/10.1038/s41598-019-45839-z.