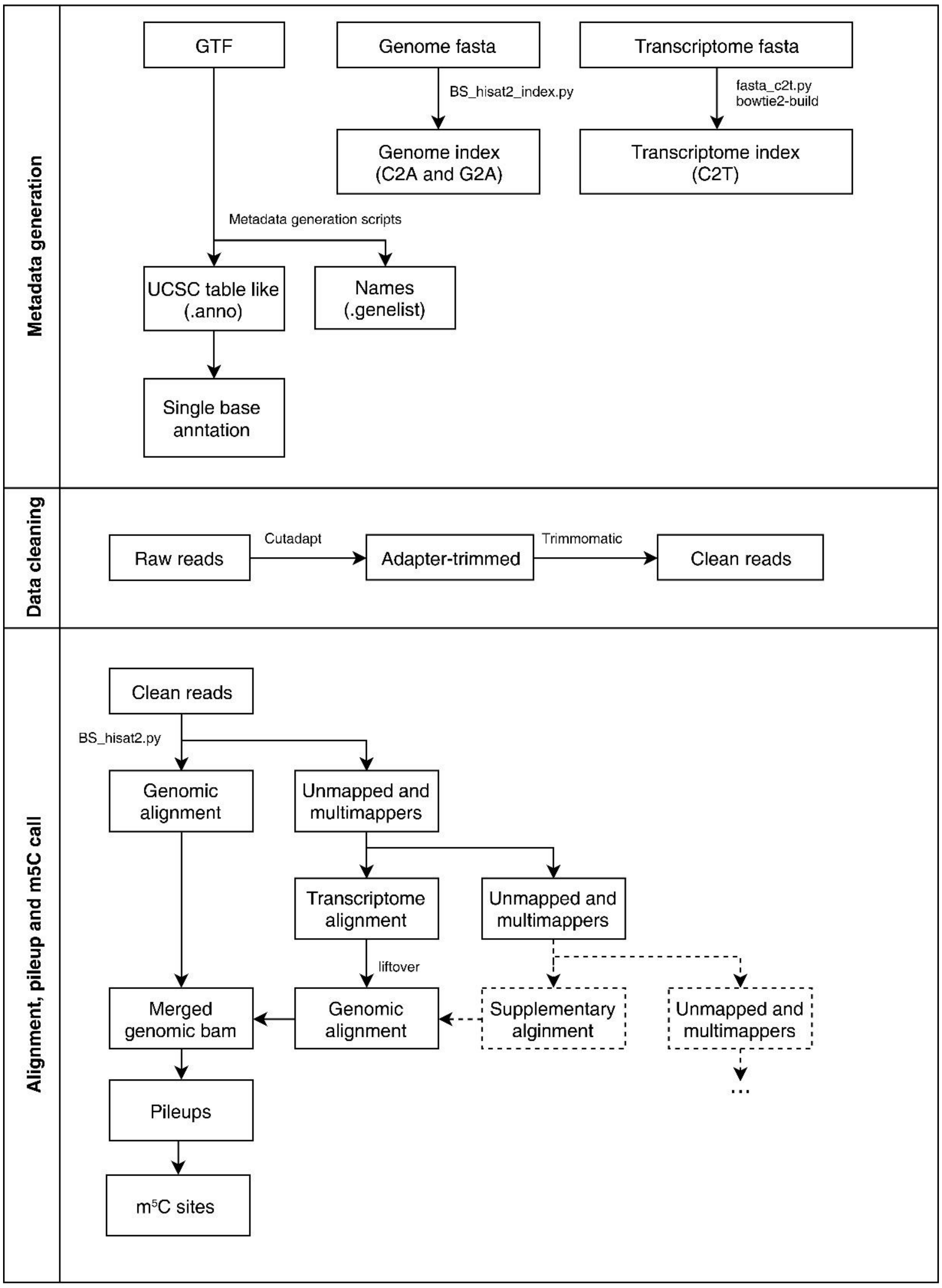
Step-by-step m^{5}C BS-seq pipeline (v1.0)

Release date: Jan 2019, Jianheng Liu

1. Overview



2. Environment

2.1 Operation system and hardware

System (development)	CentOS Linux release 7.4.1708 (Core)
CPU	8 cores or more
RAM	70 GB or more
Disk	● 50 GB for human metadata ● 200 GB for a 30 million PE150 data ● 500 GB for a 200 million PE125 data

* Windows will never be compatible since some modules cannot be installed.

2.2 Python version and modules

Python 2.7.14 with numpy (1.13.3), scipy (0.19.1), pysam (0.12.0.1), Biopython (1.70)

2.3 Software and open-source scripts

JAVA	JRE 1.8.0_131
Quality control and formatting	Cutadapt 1.14 Trimmomatic 0.36
Mapping	HISAT2 2.1.0 (the same as index build) Bowtie 2.2.9 (the same as index build)
Bam processing	Samtools 1.6

3. Customized scripts

Name	Usage
Metadata generation	
gtf2anno.py	Transfer GTF to UCSC annotation table (Ensembl format only)
gtf2genelist.py	Extract gene/isoform information from GTF (Ensembl format only)
anno_to_base.py	Annotate each base in GTF
anno_to_base_remove_redundance.py	Remove redundancies by determining which gene a single base belongs to
fasta_c2t.py	Convert Cs in fasta to Ts
BS_hisat2_index.py	Build HISAT2 indexes
ref_sizes.py	Extract the lengths of references

Alignment and pileup	
BS_hisat2.py	Map reads to genome
BS_bowtie2.py	Map reads to transcriptome
concat_bam.py	Merge BAM files
pileup_genome_multiprocessing_v1.4.py	Split reference and pileup in multiprocessing manner
m5C_pileup_formatter.py	Format pileups
Call sites	
m5C_caller.py	Call m5C sites from formatted pileups
m5C_caller_multiple.py	Call m5C sites from multiple samples
m5C_intersection_ single_r1.py	Identify m5C sites in each sample
m5C_intersection_ multi_r1.py	Identify m5C sites among replicates

4. Metadata

Using human meta data as example:

Name	Source	Suffix	Comment
Homo_sapiens.GRCh37.75.all.RNA.format.fa	Merge fasta from Ensembl: Homo_sapiens.GRCh37.75.ncrna.fa Homo_sapiens.GRCh37.75.cdna.all.fa	.fa	
Homo_sapiens.GRCh37.75.all.RNA.format.c2t.fa	fasta_c2t.py	.fa	*In silico* converted transcriptome
Homo_sapiens.GRCh37.75.dna_sm.primary_assembly.fa	Ensembl	.fa	
Homo_sapiens.GRCh37.75.gtf	Ensembl	.gtf	GTF annotation, header removed
Homo_sapiens.GRCh37.75.anno	gtf2anno.py	.anno	Exons positions of gene
Homo_sapiens.GRCh37.75. genelist	gtf2genelist.py	.genelist	Isoform annotations like gene id, gene name, biotype and length
Homo_sapiens.GRCh37.75.noredundance.base.sorted.base	anno_to_bed_single.py anno_to_bed_single_remove_reduandance.py	.base	Genomic bases annotated by gene and transcript, without any redundance

Human (GRCh37.75) soft-masked genome sequence, transcriptome sequence (cDNA and ncRNA) and GTF annotation were downloaded from Ensembl (https://asia.ensembl.org/index.html).

GTF header should be removed to guarantee successful running of scripts:

5. Pipeline

5.0 Metadata preparation

This step generates reusable metadata from GTF and index for m^{5}C mapping, pileup and calling. For human genome and transcriptome, the metadata might occupy ~47 GB disk (800 MB for GTF file, 450 MB for bowtie2 indexes, 17 GB for HISAT2 indexes and 14 GB for single base annotation database).

The sequences and GTF file are highly recommended linked, i.e. Ensembl fasta with Ensembl GTF, UCSC fasta with UCSC GTF. If those files come from difference source, please check whether they are really matched, they are a lot of traps: **1 vs chr1, MT vs chrM, HSCHRUN_RANDOM_CTG1 vs GL000211.1**, etc. Also, you should check if the transcript IDs were identical between GTF and fasta, for example, GTF from Gencode have a transcript version id, but transcripts in Ensembl GTF only have a stable ID. Finally, you should also check if the biotypes in your GTF is the same as that in Ensembl, or you should provide a score list for the biotypes present.

When everything is OK, begin metadata generation:

It's better to build the genomic index first, since it will take a long time, just run it in background, make sure you have enough memory. See script document if you want to run index build in parallel or if you want to include spike-ins/strand specific sequences in genomic alignment:

```
#Make hisat2 index
#This will take up to 250 GB RAM with GTF, and ~30 GB RAM for genome only. Manually set hisat2-build if memory is not enough.
python BS_hisat2_index.py \
-i Homo_sapiens.GRCh37.75.dna_sm.primary_assembly.fa \
--gtf Homo_sapiens.GRCh37.75.gtf \
--hisat2-path /PATH/TO/HISAT2/
-o HISAT2_INDEX
```

Then generate the transcriptome index. Run bowtie2-build in background since it will take a bit long:

```
#Merge all transcriptome fasta, add spike-in to this file if needed
cat Homo_sapiens.GRCh37.75.ncrna.fa Homo_sapiens.GRCh37.75.cdna.all.fa > Homo_sapiens.GRCh37.75.all.RNA.fa

#In silico convert reference
python fasta_c2t.py \
-i Homo_sapiens.GRCh37.75.all.RNA.format.fa \
> Homo_sapiens.GRCh37.75.all.RNA.format.c2t.fa

#Make bowtie2 index
bowtie2-build \
Homo_sapiens.GRCh37.75.all.RNA.format.c2t.fa \
Homo_sapiens.GRCh37.75.all.RNA.c2t
```

Then generate other metadata:

```
#Get a UCSC table like file
python gtf2anno.py \
-i Homo_sapiens.GRCh37.75.gtf \
> Homo_sapiens.GRCh37.75.anno

#Get a list of transcript information
python gtf2genelist.py \
-i Homo_sapiens.GRCh37.75.noheader.gtf \
-f Homo_sapiens.GRCh37.75.all.RNA.format.c2t.fa \
> Homo_sapiens.GRCh37.75.genelist

#Get reference lengths
python ref_sizes.py \
-i Homo_sapiens.GRCh37.75.dna_sm.primary_assembly.fa,control.fa \
-o Homo_sapiens.GRCh37.size

#Run the following two steps in background
#This step will use 10 GB RAM to sort the file by default
python anno_to_base.py \
-i Homo_sapiens.GRCh37.75.anno \
-o Homo_sapiens.GRCh37.75.base

#A sorted file *must* be used
python anno_to_base_remove_redundance_v1.0.py \
-i Homo_sapiens.GRCh37.75.base.sorted \
-o Homo_sapiens.GRCh37.75.noredundance.base \
-g Homo_sapiens.GRCh37.75.genelist
```

5.1 Data cleaning

We use a two-step data cleaning strategy: adapters were firstly removed by Cutadapt, then adapter-free reads were quality trimmed with Trimmomatic to get high quality alignment input. Note that the parameters we used here will generate a great number of devotailing reads — you can omit the HEADCROP:10 in Trimmomatic and just ignore the first 10 bases in pileup.

```
#Suppose using gunzip files as input, using dUTP stranded library (read1-ATC; read2-ATG) and illumina standard adapters
#You can add NNNNNN at the beginning of adapters to fully remove any hexamer
cutadapt \
-a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC \
-A AGATCGGAAGAGCGTCGTGTAGGGAAAGAG \
-e 0.25 \
-q 25 \
--trim-n \
-o $read1.cutadapt.fastq \
-p $read2.cutadapt.fastq \
$read1.fastq.gz \
$read2.fastq.gz

#Remove hexamer and low quality bases
java -jar trimmomatic-0.36.jar $read1.cutadapt.fastq $read2.cutadapt.fastq rev.fastq rev.UP.fastq fwd.fastq fwd.UP.fastq HEADCROP:10 SLIDINGWINDOW:4:22 AVGQUAL:25 MINLEN:40
```

5.2 Mapping to genome

Two HISAT2 programs will be run simultaneously, and overall take ~10 GB RAM. Since the script will load C-containing reads to memory, this script is not suitable for a huge file of lowly converted sample. When handling this case, split your file, align them and concatenate them after align ment. Unmapped reads and multimappers will be stored in fastq for further alignment.

```
python BS_hisat2.py \
-F fwd.fastq \
-R rev.fastq \
-o hisat2_genome \
-I /PATH/TO/HISAT2_INDEX/ \
--index-prefix HISAT2 \
--hisat2-path /PATH/TO/HISAT2/ \
--del-convert \
--del-sam

#output: hisat2_genome.bam, hisat2_genome.multimapping.bam, fwd.unmmaped.fastq, rev.unmmaped.fastq
```

If you want to change the parameters in alignment, just provide a tabular file using "--hisat2-param" option.

```
python BS_hisat2.py \
-F fwd.fastq \
-R rev.fastq \
-o hisat2_genome \
-I /PATH/TO/HISAT2_INDEX/ \
--index-prefix HISAT2 \
--hisat2-path /PATH/TO/HISAT2/ \
--del-convert \
--del-sam \
--hisat2-param <parameters>

#Parameter file example:
#-p[\t]12
```

5.3 Mapping to transcriptome

For other reference, Bowtie2 rather than HISAT2 is recommend, since Bowtie2 somehow overcomes HSIAT2 in handling alignments that map to junction-free reference. You can map the reads to multiple reference subsequently and finally merge them together.

A script liftover transcriptomic coordinates to genomic coordinates is used here to make pileup easy. You can use “--bowtie2-param” to change the parameters you use.

```
#Map to transcriptome
python BS_bowtie2.py \
-F fwd.unmapped.fastq \
-R rev.unmapped.fastq \
-o bowtie2_trans \
--bowtie2-path /PATH/TO/BOWTIE2/ \
-I /PATH/TO/Homo_sapiens.GRCh37.75.all.RNA.c2t \
-g /PATH/TO/Homo_sapiens.GRCh37.75.genelist \
--del-convert \
--del-sam
#output: bowtie2_trans.bam, bowtie2_trans.multimapping.bam, fwd.unmapped.unmapped.fastq, rev.unmapped.unmapped.fastq

#Lift-over to genome
python Bam_transcriptome_to_genome_v1.0.py \
-i bowtie2_trans.bam \
-o bowtie2_genome.bam \
-a /PATH/TO/Homo_sapiens.GRCh37.75.anno \
--dict /PATH/TO/Homo_sapiens.GRCh37.75.size
#output: bowtie2_genome.bam, bowtie2_trans.unlift.bam
```

5.4 Merge BAM files (optional)

If you have multiple BAM files in same coordinates, you can merge them before pileup and save time.

```
#Merge bam, use 4 threads (-t 4) while sorting (--sort) bam files, generate index for bam (--index)
python concat_bam.py \
-t 4 \
-i hisat2_genome.bam bowtie2_genome.bam […] \
-o genome_merged.bam \
--sort \
--index
#output: genome_merged.sorted.bam genome_merged.sorted.bam.bai
```

5.5 Pileup

This is the last step before m^{5}C call. BAM file will be pileup by pysam first, then the pileup result will be format by certain C-cutoff filters. It’s time-consuming running a pileup, especially handling BAM input with over 100 million records. The script here splits the genomic coordinates to accelerate this process. However, this turn the program from CPU bound to CPU bound and I/O bound, and the optimized parallel number should around 8. You can also reduce the max depth in pileup to ~1000 to make it quicker.

If you only have several regions interested in, you might need to modify the code to scan for specific regions. If you want to trim some bases, you can use --trim-tail or --trim-head here.

Note that the C-cutoff used in downstream analysis is determined in the pileup formation step. Modify the script if you want to use other C-cutoff settings (default is 1-10, 15, 20, None).

```
#Pileup genome, each process will use 2-3 Gb RAM
python pileup_genome_multiprocessing_v1.4.py \
-P 6 \
-f /PATH/TO/Homo_sapiens.GRCh37.75.dna_sm.primary_assembly.fa control.fa\
-i genome_merge.sorted.bam \
-o m5C.pileups.tmp
#output: genome_merged.pileup

#Sort merged pileups, if you have seperately run multiple pileups
sort -k 3,3 -k 1,2 -S 5G --parallel 10 -
T ./ pileup.merged > pileup.merged.sorted

#Convert temp file to formatted pileup
python m5C_pileup_formatter.py \
--db /PATH/TO/Homo_sapiens.GRCh37.75.noredundance.base \
-i m5C.pileups.tmp \
-o m5C.pileups.formatted.txt \
--CR CR.txt
```

5.6 Call m^{5}C sites from a sample

Now you can call m^{5}C sites from the pileups, with different C-cutoffs (default in [1-10, 15, 20, None]) or different statistic methods (binomial as default, alternatives are Poisson and Fisher exact. test, one-tail P-value for binomial and Fisher exact. test)

```
#See what options can be used
python m5C_caller -h

#Call sites with gene specific conversion rates, default setting calls C-cutoff in [3, None]
python m5C_caller.py \
-i m5C.pileups.txt \
-o m5C_sites_gene \
-c 10 \
-C 3 \
-r 0.1 \
-p 0.05 \
-s 0.9 \
-R 0.8 \
-cutoff 1,3,None
--CR gene \
--method binomial

#output: m5C_sites_gene.1.txt m5C_sites_gene.3.txt m5C_sites_gene.None.txt

#Call sites with overall conversion rate
python m5C_caller.py \
-i m5C.pileups.txt \
-o m5C_sites_overall \
--CR overall \
--method binomial

#output: m5C_sites_overall.3.txt m5C_sites_overall.None.txt

#Call sites with conversion rates of controls
#Control file look likes:
cat ERCC.txt
ERCC-00002
ERCC-00003
ERCC-00004
ERCC-00009
ERCC-00014
```

```
[...]

python m5C_caller.py \
-i m5C.pileups.txt \
-o m5C_sites_control \
--CR control \
--control ERCC.txt \
--method binomial

#output: m5C_sites_control.3.txt m5C_sites_control.None.txt
```

5.7 Call m^{5}C sites from multiple samples

You might have a set of samples, and you want to call all of the sites, determine which sites is methylated in which sample and compare the m5C level between samples. This step fits what you want.

Firstly, you get the list of information for each sample and the sites detected in at least one sample.

```
#Suppose you want to call sites from two samples: 293T and 293T.siNSUNs
#Now you have the pileups of them: 293T.pileups.txt 293T.siNSUNs.pileups.txt

#Suppose you want to use a gene specific CR cutoff for 293T, and a overall C
R cutoff for 293T.siNSUNs. And you want to apply a C-
cutoff=3 on 293T, while 5 for 293T.siNSUNs
#Draw a table like this:
cat samples.txt
293T    293T.pileups.txt    gene    3
293T.siNSUNs    293T.siNSUNs.pileups.txt    overall 5

#Then run m5C_caller_multiple.py:
python m5C_caller_multiple.py \
-i samples.txt \
-o output.csv \
-P 2 \
-c 20 \
-C 3 \
-r 0.1 \
-p 0.05 \
--method binomial

#Then it will run in 2 processes
```

```

#Else if you have a list to search for, like this:
cat m5C.list.txt
chr1    1000    +
chrX    5000    -

#You can run as:
python m5C_caller_multiple.py \
-i samples.txt \
-o output.csv \
-l m5C.list.txt \
-P 2 \
-c 20 \
-C 3 \
-r 0.1 \
-p 0.05 \
--method binomial

#The script will skip the de novo searching step
```

Then you can determine which sites passed filter:

```
python m5C_intersection_single_r1.py \
-i output.csv \
-o output_single.csv \
--P-method stouffer
```

And also, merge replicates

```
python m5C_intersection_multi_r1.py \
-i output_single.csv \
-l groups.txt\
-o output_multi.csv \
--P-method stouffer
```