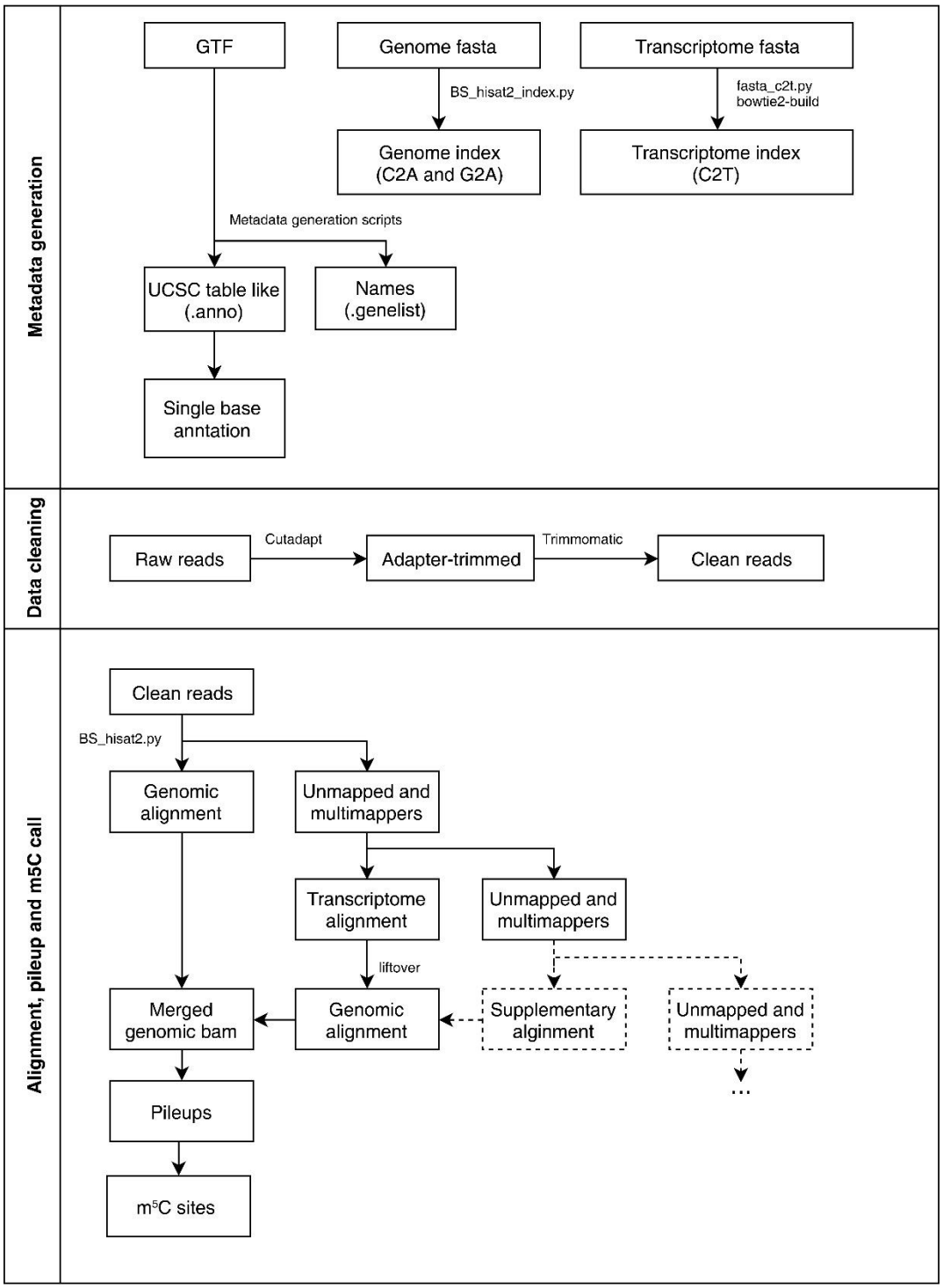
Step-by-step m^{5}C site calling pipeline (v1.1)

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 is not 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. Custom 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 the redundance
fasta_c2t.py	Convert Cs in fasta to Ts
BS_hisat2_index.py	Build HISAT2 indexes
ref_sizes.py	Extract the lengths of the references
Alignment and pileup	

BS_hisat2.py	Map reads to the genome
BS_bowtie2.py	Map reads to the transcriptome
Bam_transcriptome_to_genome_v1.0.py	Convert transcriptome alignment to genome alignment (BAM to BAM)
concat_bam.py	Merge BAM files
pileup_genome_multiprocessing_v1.4.py	Split the reference and pileup the file in a multiprocessing manner
m5C_pileup_formatter.py	Format pileups
Call sites	
m5C_caller.py	Call m5C sites from the formatted pileup files
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 in replicates

4. Metadata

Using the human metadata as an example:

Name	Source	Suffix	Comment
Homo_sapiens.GRCh37.75.all.RNA.format.fa	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	Exon positions of the genes
Homo_sapiens.GRCh37.75. genelist	gtf2genelist.py	.genelist	Isoform annotations, such as gene id, gene name, biotype and gene 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 the successful running of scripts.

5. Pipeline

5.0 Metadata preparation

This step generates reusable metadata index for RNA BS-seq mapping, pileup and site calling. For the human genome and transcriptome, the metadata occupies ~47 GB disk (800 MB for GTF file, 450 MB for bowtie2 indexes, 17 GB for HISAT2 indexes and 14 GB for annotation database).

Matters needing attention:

1) The sequences and GTF file are highly recommended to be downloaded from the same source, i.e. Ensembl fasta with Ensembl GTF, UCSC fasta with UCSC GTF. If these files come from difference sources, please check whether they are fully matched (e.g. **1 vs chr1, MT vs chrM, HSCHRUN_RANDOM_CTG1 vs GL000211.1**).

2) Check if the transcript IDs were identical between GTF and fasta. For example, GTF from Gencode has a transcript version id, but transcripts in Ensembl GTF only have a stable ID.

3) Check if the biotypes in your GTF is the same as that in Ensembl.

When everything is OK, begin to generate the genome index.

```
#Build hisat2 index
#This will take up to 250 GB RAM with GTF, and ~30 GB RAM for genome only. Manually set hisat2-build if the 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
```

Generate the transcriptome index. See the script document if you want to include spike-ins in this step.

```
#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 > Hom
o_sapiens.GRCh37.75.all.RNA.fa

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

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

Generate other metadata:

```
#Get a UCSC table format 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 the 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 the 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 first removed by Cutadapt, then adapter-free reads were quality trimmed with Trimmomatic.

```
#Suppose using gunzip files as the input, dUTP stranded library (read1-ATC; read2-ATG) and standard illumina 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 the 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 Map to the genome

Two HISAT2 programs are run simultaneously, and take ~10 GB RAM. Since the script loads C-containing reads to the memory, it is not suitable for a large fastq file of low conversion rate. When handling this kind of data, you can split your file, align them separately, and concatenate them after the alignment. 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
```

```
10.
11. #output: hisat2_genome.bam, hisat2_genome.multimapping.bam, fwd.unmmaped.fas
    tq, rev.unmmaped.fastq
```

Use "--hisat2-param" option and provide a tabular file to change the parameters in the alignment step, if needed. Your parameter file should look like (the same in transcriptome mapping):

```
1. cat parameter_file
2.
3. #[parameter name][space][value]
4. -p 10
```

```
1. python BS_hisat2.py \
2. -F fwd.fastq \
3. -R rev.fastq \
4. -o hisat2_genome \
5. -I /PATH/TO/HISAT2_INDEX/ \
6. --index-prefix HISAT2 \
7. --hisat2-path /PATH/TO/HISAT2/ \
8. --del-convert \
9. --del-sam \
10. --hisat2-param parameter_file
11.
```

5.3 Map to transcriptome

Use “--bowtie2-param” to change the parameters if needed. A script that liftover the transcriptomic coordinates to the genomic coordinates is provided.

```
1. #Map to the transcriptome
2. python BS_bowtie2.py \
3. -F fwd.unmapped.fastq \
4. -R rev.unmapped.fastq \
5. -o bowtie2_trans \
6. --bowtie2-path /PATH/TO/BOWTIE2/ \
7. -I /PATH/TO/Homo_sapiens.GRCh37.75.all.RNA.c2t \
8. -g /PATH/TO/Homo_sapiens.GRCh37.75.genelist \
9. --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, you can merge them before the pileup step.

```
#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

BAM file is piled up by pysam, then the pileup result is formatted using a given C-cutoff filter. It's time-consuming to run a pileup, especially when handling a BAM input with >100 million records. The script below can split the genomic coordinates to accelerate the process. You can also reduce the max depth in pileup to ~1000 to accelerate the process.

If you need to trim some bases (e.g. hexamer), you can use --trim-tail or --trim-head option.

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

```
#Pileup the 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: m5C.pileups.tmp

#Sort merged pileups, if you have run multiple pileups (not suggested, merge BAM files instead)
sort -k 3,3 -k 1,2 -S 5G --parallel 10 -T ./ m5C.pileups.tmp > 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

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

This script is used for testing whether the sample is OK. I highly recommend you use the "multiple" pipeline below.

```
#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 the 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 the conversion rates of spike-ins
#Example of spike-in file:
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 and analyze m^{5}C sites from multiple samples

Call sites from multiple samples (sites that are detected in at least one sample are listed).

```
#Suppose you want to call sites from two samples: 293T and 293T.siNSUNs
#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 an overall CR cutoff for 293T.siNSUNs. And you want to apply a C-cutoff=3 on 293T, and C-cutoff=5 for 293T.siNSUNs
#Draw a sample list table:
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


#If you have a list to search for, e.g.:
cat m5C.list.txt
chr1    1000    +
chrX    5000    -

#You can run the script 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
```

Now you have the CSV file (output.csv) containing the m^{5}C candidates without annotating which sites pass the filter. You can use the following script to mark those sites passed filter (remember using the same filter parameters as above). Those sites didn't pass the signal noise filter will be masked in this step.

```
python m5C_intersection_single_r1.py \
```

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

Until now you have the list of m^{5}C sites from each sample, but sites were not merged together among replicates. To do this, please provide a list (groups.txt) simply modified from samples.txt:

```
#Suppose you have some replicates of 293T sample.
#Note that only the first two columns are necessary:
#the first column is the name of groups, #the second column is the name of sample that the same as what in output.csv.
#You can just add the group column to samples.txt
cat groups.txt
293T    293T_rep1   293T_1.pileups.txt  gene    3
293T    293T_rep2   293T_2.pileups.txt  gene    3
293T_siNSUN2    293T_siNSUN2_rep1   293T_siNSUN2_1.pileups.txt  gene    3
293T_siNSUN2    293T_siNSUN2_rep2   293T_siNSUN2_2.pileups.txt  gene    3
```

Then run the following script to get high-confident sites called from replicates:

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