HIPPIE User Manual

(v0.1c, 2014/9/29, Yih-Chii Hwang, yihhwang [at] mail.med.upenn.edu)

OVERVIEW OF HIPPIE [↑top]

HIPPIE (High-throughput Identification Pipeline for Promoter Interacting Enhancer elements) is a software package that takes Hi-C raw reads as input and ultimately identifies **enhancer–target target gene relationships** by mapping the reads to reference genome, calling peak fragments, detecting DNA–DNA interactions with quality controls, and integrating functional epigenomics knowledge. It is designed to be executing on oracle grid engine system with memory and error control, as well as prerequisite control.

The entire script can be downloaded here.

Flowchart of HIPPIE [↑top]

A complete HIPPIE workflow run consists of four phases as outlined in the following flowchart.

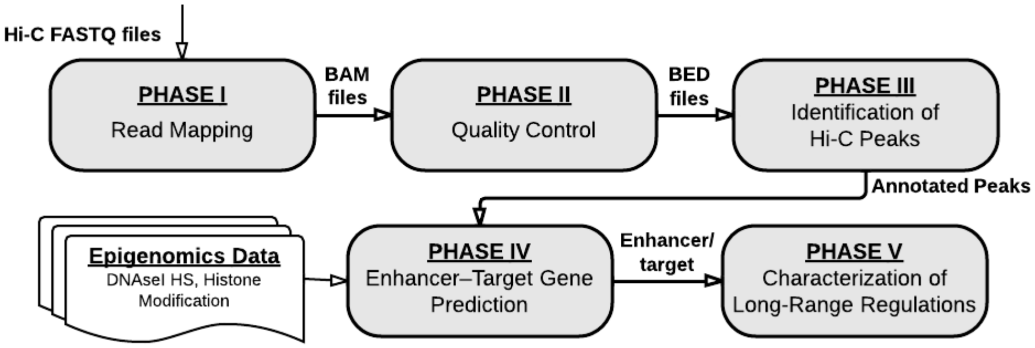

Figure 1. HIPPIE flowchart.

Requirements [↑top]

- bwa http://sourceforge.net/projects/bio-bwa/files/ (tested in 0.6.2-r126)
- Picard http://picard.sourceforge.net/ (tested in 1.113, java version 1.7.0_09-icedtea)
- SAMtools http://sourceforge.net/projects/samtools/files/ (tested in 0.1.19-44428cd)
- BEDtools https://github.com/arq5x/bedtools2 (tested in 2.19.1)
- R http://www.r-project.org (tested in 3.1.0). Used R libraries are:
  - gplots http://cran.r-project.org/web/packages/gplots/index.html
  - RColorBrewer http://cran.r-project.org/web/packages/RColorBrewer/index.html
- perl (tested on 5.10.1). Used modules are: “Math::CDF qw(:all)”, “POSIX qw(ceil floor)”, “List::Util”.
- awk, zcat, sort

Please set the path for bwa, Picard, SAMtools, Bedtools, R, and PERL5LIB (e.g. path to Math::CDF) in `hippie.ini` (e.g. `path/to/hippie/hippie.ini`).

Directory structure for HIPPIE execution [↑top]

HIPPIE operates on a per-library (sample) level. A “project” can contain multiple libraries (samples) and each library resides in one directory. Each library directory has sub-directories for the command scripts and output files (`cmd/`) and intermediate files (`sai/`, `sam/`, `bam/`) as in Figure2. The user is required to prepare and maintain the input files based this directory structure, as well as describe information of each library of the project in a configuration file, including paths for the project, reference genome, and reference epigenetics (ENCODE) data, etc.

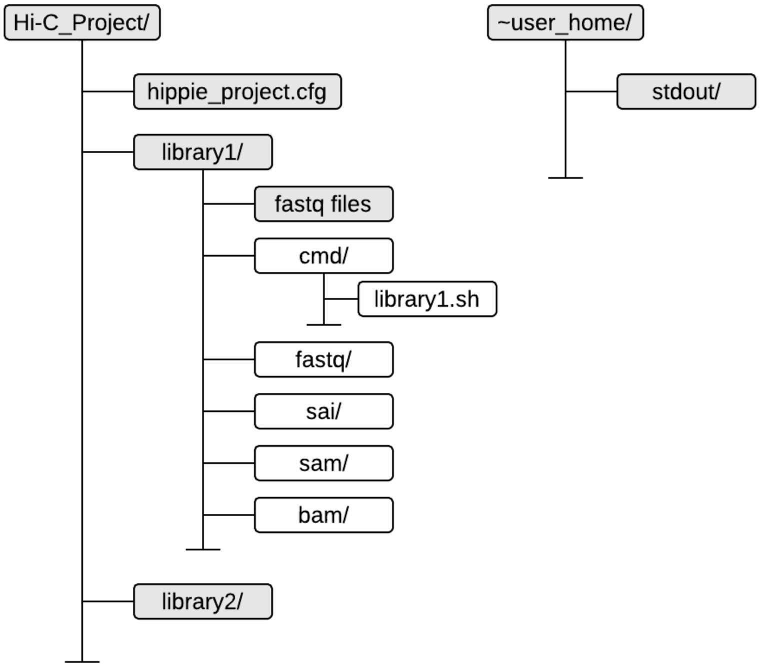

Figure 2. The directory structure up of HIPPIE. The shaded (grey) directories have to be prepared by the users, and the white directories are automatically generated by HIPPIE based on the configuration file.

In this manual, we will use "*Hi-C_Project*" as an example project. The names of the libraries sequenced are “*library1*”, “*library2*”, “*library3*”, etc.

Library (sample) directory [↑top]

Each library can contain multiple paired-end fastq files with as long as the reads contained are from the same library. The fastq files can be compressed (*.fastq.gz) or uncompressed (*.fastq). The files generated subsequently are stored in each individual sample's sub-directories (eg. test_data/Hi-C_project/library1/).

Log directory: $HOME/stdout [↑top]

Under Open Grid Scheduler or job distributing environment, the screen output from running jobs is redirected to a log file. HIPPIE stores all such log files in a “`stdout/`” directory under user’s home directory. You need to create it before running HIPPIE.

```
$ mkdir -p ~/stdout
```

Create the configuration file [↑top]

Please refer to the template file named “project_configure.cfg”. This file contains information of the libraries sequenced, including cell types, restriction enzyme, etc. Common attributes, such as reference genome, other epigenetics (ChIP-seq peaks, DNase-seq hotspots) data, and research project name are also described.

We suggest users separate the analyses of Hi-C for different organisms or species by creating different project directories. This can prevent confusion of the reference genome and clarify the epigenetics data usage.

Prepare the reference genome [↑top]

Please first download the reference genome sequence in FASTA format (.fa file), and run bwa index to generate the index of the reference genome. See below example for human hg19:

1. Please find the reference genome can be found at: http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.2bit and use the UCSC utility program, twoBitToFa, to extract the .fa from this file (from http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/).

2. Please generate the index file of the reference genome (eg. hg19.fa) for bwa alignment.

```
bwa index -a bwtsw hg19.fa
```

The index will be generated under the same directory of hg19.fa. After the index is generated, set the path to the hg19.fa GENOME_REF under in your configuration file (eg. “project_configure.cfg”).

INSTALL MATH::CDF PERL PACKAGE [↑top]

HIPPIE uses perl package MATH::CDF to estimate the p-value of each interaction. To install the package, use either CPAN or follow the following steps:

1. Please download the module file (Math-CDF-0.1.tar.gz) from CPAN at: http://search.cpan.org/~callahan/Math-CDF-0.1/CDF.pm.

2. Untar the tar.gz file and follow the installation instruction. We suggest installing the module under user’s home directory ($HOME), so that all cluster nodes can locate the it.

```
tar jzvf Math-CDF-0.1.tar.gz
cd Math-CDF-0.1
perl Makefile.PL PREFIX=$HOME
make
make install
```

2. Update the path to MATH::CDF to PERL5LIB by simply adding the path in hippie.ini:

```
export PERL5LIB=$PERL5LIB:$HOME/lib64/perl5
```

RUN HIPPIE [↑top]

Each library has its own individual directory and sub-directories as in Figure 2. Once execute the configuration file, a tailored bash script file (.sh) that contains all the commands to complete the analysis. To achieve this, we describe how to execute this with the configuration file.

Once you have prepared the configuration file (e.g. `project_configure.cfg`), the following are the steps to execute it.

i. Change directory to the project directory (e.g. `Hi-C_Project/`).

```
$ cd path/to/Hi-C_Project
```

ii. Evoke the environment paths by “source” the `hippie.ini` file from HIPPIE package (e.g. `path/to/hippie/hippie.ini`). One needs to consult the person who installed HIPPIE, if you cannot find it.

```
$ source path/to/HIPPIE/hippie/hippie.ini
```

Users can try "`echo $HIPPIE_HOME`" to check whether the path set-up has worked correctly. It should display where HIPPIE locates.

```
$ echo $HIPPIE_HOME
```

iii. Make sure `$HOME/stdout/` directory is made.

```
$ ls $HOME/stdout/
```

iv. Run HIPPIE with a configuration file (`-f`) to generate the tailored bash script for each library.

```
$ ./hippie.sh -f project_configure.cfg [-h HIPPIE_HOME_DIR] [-p1] [-p2] [-p3] [-p4]
```

- **`-f`** - specifies the location of the project configuration file
- **`-h` HIPPIE_HOME_DIR** - optionally specify the location of the HIPPIE home directory. Otherwise it would use the environment variable `$HIPPIE_HOME` in hippie.ini
- **`-p1 -p2 -p3 –p4`** - optionally submit all tasks from each phase for all samples specified in the configuration file.

This would create the library-level directories (`cmd/, fastq/, sai/, sam/, and bam/`). Under each sample's cmd/ directory would be a launching script with a name in this format: “`library1.sh`”, and under each library’s fastq/ directory would be the soft link of the pair-end read files (fastq, or fastq.gz).

In the next several sections we describe the different ways of using the launching script for each library.

THE PHASE MODE: PIPELINE EXECUTION [↑top]

After executing `hippie.sh`, each library will have its own individual sub-directories generated as well as the bash script (e.g. `library1.sh`) propagated under its cmd/ directory. The entire HIPPIE pipeline can be divided into five phases, which can be run individually and sequentially.

The procedure of running each of the four phases is the same: First change directory to a library's cmd/ sub-directory and begin the analysis process by the phase (p1), phase 2, phase 3, phase 4, and phase 5. The tasks of the phases are chained together and the submitted jobs are designed to run only after the prerequisite jobs are finished.

```
$ cd path/to/Hi-C_Project/library1/cmd/

$ ./library1.sh -p1

$ ./library1.sh -p2

$ ./library1.sh -p3

$ ./library1.sh –p4

$ ./library1.sh –p5
```

“Multiple phases” is also acceptable. That is, users can submit phase 1 through phase 4 all at once.

```
$ ./library1.sh -p1 –p2 –p3 –p4 -p5
```

The phase mode can operate with the debug mode (details see below).

THE TASK MODE: RUNNING SELECTED PORTIONS OF THE PIPELINE [↑top]

Task mode allows you to run any single tasks of HIPPIE.

```
$ cd path/to/Hi-C_Project/library1/cmd

$ ./library1.sh -t TASKNAME
```

- `-t TASKNAME` - the single task to run

The task mode can also operate along with the debug mode. The tasks are chained together, thus the following jobs will only run after their consecutively prerequisite job is finished. Thus, one can skip the first task, but consecutively submit jobs for the second task and the third task. For example:

```
$ ./library1.sh -t SECOND_TASKNAME

$ ./library1.sh -t THIRD_TASKNAME
```

THE DEBUG MODE: DEBUGGING OPTION [↑top]

Debug mode does not submit jobs. Instead, the full command(s) that would be submitted is displayed. The -d option must be followed by any flag that would normally submit a task or sequence of tasks such as `-p1`, `-p2`, `-p3`, or `-t`.

```
$ cd path/to/Hi-C_Project/library1/cmd

$ ./library1.sh -d -p1
```

- **`-d`** - debug mode; previews the qsub command that would be submitted
- **`-p`** - runs the steps for phase 2, when preceded by `-d`, the jobs would not be submitted. Instead, the full command(s) would be displayed.

Debug mode works with any combination of task submitting flags

```
$ cd path/to/Hi-C_Project/library1/cmd

$ ./library1.sh -d -p2 -p3

$ ./library1.sh -d -t " annotateFragment"
```

PHASES AND TASKS [↑top]

Phase 1: Read Mapping [↑top]

Phase 1 takes the unmapped reads received and aligns them to the reference genome.

i. Aligning reads of all fastq files to the reference genome.

```
$ cd path/to/Hi-C_Project/library1/cmd

$ ./library1.sh -t bwaAln
```

The output files are stored in the `sai/` directory.

ii. Combining mate pairs. Task `bwaAln` is its prerequisite task.

```
$ ./library1.sh -t bwaSamp
```

The output files are stored in the `sam/` directory.

iii. Adding readgroup information to all reads. Task `bwaSamp` is its prerequisite task.
If the project configuration file (`project_configure.cfg`) has `DATA_TYPE` with "`ONEFASTQ`" or "`ONEFASTQSE`"

```
$ source library1.sh

$ ./library1.sh -t "addReadGroup $SAM_DIR/s_${LINE}_sequence.aligned.sam.gz
$BAM_DIR/s_${LINE} Samp${RGID}"
```

else

```
$ ./library1.sh -t addReadGroupTasks
```

The output files are stored in the `bam/` directory.

iv. Merging multiple alignment files (*.bam) from the same library into one alignment file. Task addReadGroupTasks is its prerequisite task.
If the flowcell configuration file's `DATA_TYPE` is "`ONEFASTQ`" or "`ONEFASTQSE`"

```
$ source library1.sh

$ ./library1.sh -t "mgBamSoftLink $BAM_DIR/s_${LINE}_rg.bam s_${LINE}_merged.bam"
```

else

```
./library1.sh -t samtoolsMergeBam
```

The output file is stored in the `cmd/` directory with file name in the format of `s_library1_merged.bam`.

Phase 2: Quality Control [↑top]

Phase 2 takes the aligned reads and further processes them with mapping quality control. First, it transforms the bam file to bed file. Then, it removes the duplicates that may be due to PCR artifacts, discards read mapped to random contigs; and finally, it filters the read pair that has worse mapping quality than user-defined mapping quality criteria.

i. Basic statistics for the mapped/alignment file. Task `samtoolsMergeBam` of Phase 1 is its prerequisite task.

```
$ cd path/to/Hi-C_Project/library1/cmd

$ source library1.sh

$ ./library1.sh -t doFlagStat "s_${LINE}_merged"
```

The output file is stored in the `cmd/stat/` directory: `s_library1_merged.flagstat`.

ii. Transform bam file to bed file. Task `samtoolsMergeBam` of Phase 1 is its prerequisite task.

```
$ ./library1.sh -t bam2Bed
```

The output file is stored in the `cmd/` directory: `s_library1_merged.bed`.

iii. Remove PCR artifact duplicate read pairs. Task `bam2Bed` is its prerequisite task.

```
$ ./library1.sh -t rmdupBed
```

The output file is stored in the `cmd/` directory: `s_library1_merged_rmdup.bed`.

iv. Filter out the reads that do not meet user-defined mapping quality. Only both reads resides on autosomal and sex chromosomes are retained. Task `rmdupBed` is its prerequisite task.

```
$ ./library1.sh -t rmBadMapped
```

The output file is stored in the `cmd/` directory: `s_library1.bed`.

Phase 3: Peak identification and functional annotation [↑top]

i. Calculate the distance of each pair of reads (forward and reverse) to their closest restriction sites and classify the reads to specific read pairs or non-specific read pairs. Task `rmBadMapped` of Phase 2 is their prerequisite task.

```
$ cd path/to/Hi-C_Project/library1/cmd

$ ./library1.sh -t getDistancetoRSLeft

$ ./library1.sh -t getDistancetoRSRight

$ ./library1.sh -t getDistancePairBed
```

The output file is stored in the cmd/ directory: `s_library1_specific.bed` and `s_library1_nonspecific.bed`.

ii. Sort out each read by if it participates in a specific read or non-specific read pair. Task `getDistancePairBed` is its prerequisite task.

```
$ ./library1.sh -t consecutiveReadsS

$ ./library1.sh -t consecutiveReadsNS
```

The output file is stored in the `cmd/` directory: `library1_consecutive_m500.bed` and `library_consecutive_NS.bed`.

iii. Get list of restriction fragments with number of reads. Tasks `consecutiveReadsS` and `consecutiveReadsNS` are both its prerequisite tasks.

```
$ ./library1.sh -t getFragmentsRead
```

The output file is stored in the cmd/ directory: `HindIII_fragment_S_reads.bed` and `HindIII_fragment_NS_reads.bed`. Depends on the restriction enzyme used, here we use HindIII as an example.

iv. Call Hi-C peaks in the unit of restriction fragment. Task `getFragmentsRead` is its prerequisite task.

```
$ ./library1.sh -t getPeakFragment
```

The output file is stored in the cmd/ directory: `library1_HindIIIfragment_S_reads_95.bed`. Here we use 95% upperbound threshold as an example.

v. Annotate the genetics feature of the Hi-C peaks. Task `getPeakFragment` is its prerequisite task.

```
$ ./library1.sh -t annotateFragment
```

The output file is stored in the cmd/directory: library1_HindIIIfragment_95_annotated.bed.

Phase 4: Prediction of enhancer-target gene interaction [↑top]

i. Identify peak–peak interactions. Task annotateFragment of Phase 3 is its prerequisite task.

```
$ ./library1.sh -t findPeakInteraction
```

The output file is stored in the cmd/ directory: for intra- chromosomal interactions: chr*_95_reads_interaction.txt and for inter- chromosomal interactions: library1_interChrm_95_reads_interaction.txt.

ii. Correct Hi-C bias from the interacting contacts. findPeakInteraction of Phase 4 is its prerequisite task.

```
$ ./library1.sh -t correctHiCBias
```

The output file is stored in the cmd/ directory: for intra- chromosomal interactions: chr*_95_reads_interaction_pvalue.txt and for inter- chromosomal interactions: library1_interChrm_95_reads_interaction_pvalue.txt.

iii. Identify promoter annotated peak–peak interaction. correctHiCBias of Phase 4 is its prerequisite task.

```
$ ./library1.sh -t findPromoterInteraction
```

The output file is stored in the cmd/ directory: for intra- chromosomal interactions: chr*_95_promoter_annotated_interaction_promoterAnno.txt and chr*_95_promoter_annotated_promoterInteraction_pvalue.txt and for inter-chromosomal interactions: library1_95_interChrm_promoterInteraction.txt and library1_interChrm_95_reads_interaction_pvalue.txt.

iv. Identify promoter interacting enhancer elements (also known as candidate enhancer elements, CEE). Task findPromoterInteraction is its prerequisite task.

```
$ ./library1.sh -t getCeeTarget
```

The output file is stored in the cmd/ directory: library1_95_CEE_gene.bed.

Phase 5: Characters analysis of enhancer-target gene interactions [↑top]

i. Calculate distance distribution between enhancers and their targets/closest genes. Task getCeeTarget is its prerequisite task.

```
$ ./library1.sh -t ETdistance
```

The output file is stored in the `cmd/` directory: `library1_95_ET_distance.txt`.

ii. Enrichment analyses for regulatory associated histone marks within enhancer elements and other interactions, and plot the enrichment bar figure. Task `getCeeTarget` is its prerequisite task.

```
$ ./library1.sh -t histoneEnrichment

$ ./library1.sh –t plotHisEnrichment
```

The output file is stored in the `cmd/` directory: `histone_enrichment.txt` and `library1_histone_enrichment.jpg`.

iii. Enrichment analyses of GWAS hit within the enhancer elements. Task `getCeeTarget` is its prerequisite task.

```
$ ./library1.sh -t GWASEnrichment
```

The output file is stored in the cmd/ directory: `library1_95_GWAS_enrichment.txt`.