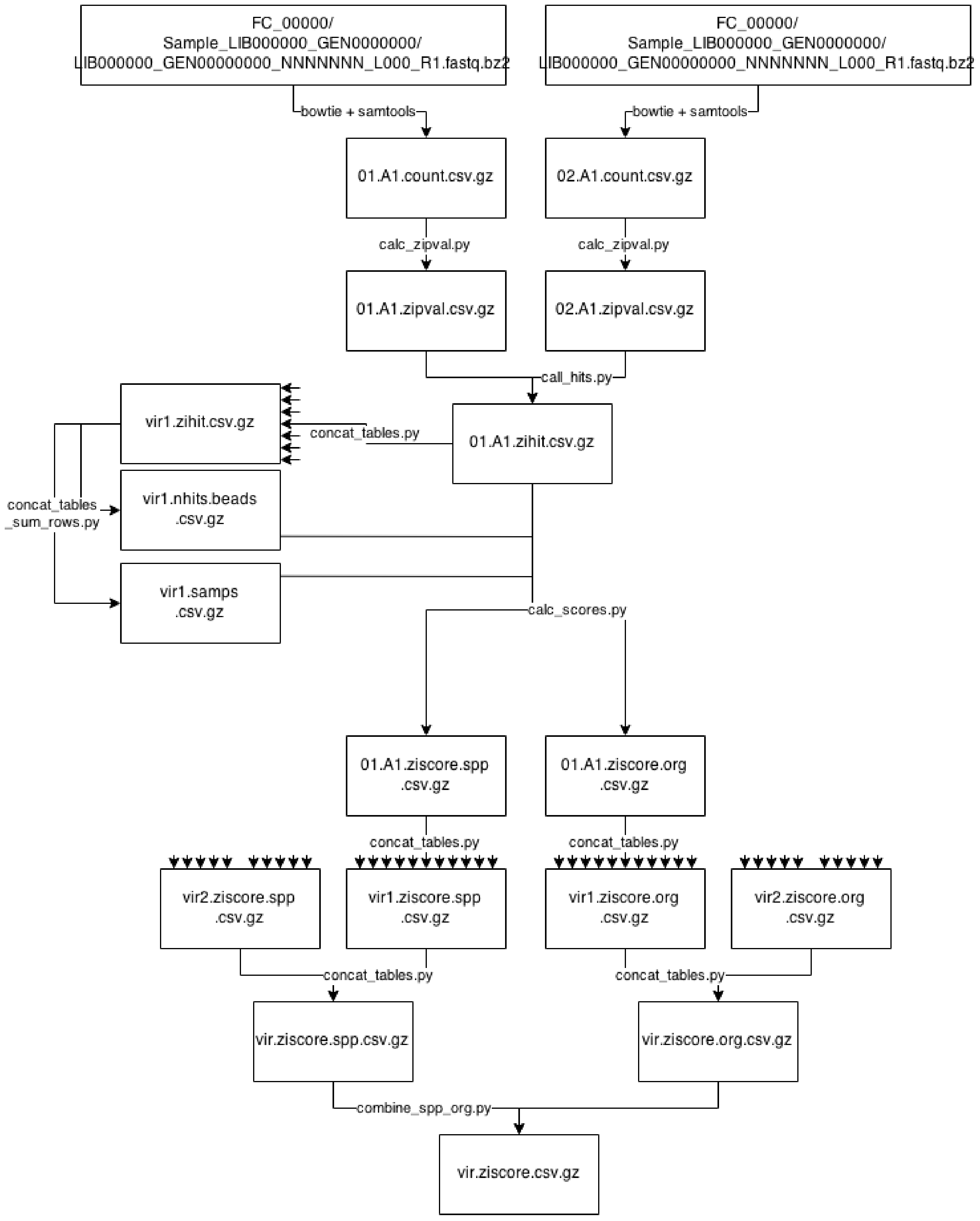
Data Analysis Pipeline

Overview Flowchart



Get on Orchestra

You need an account on the HMS Orchestra cluster. Access the cluster using SSH from the Terminal on Mac OS or PuTTY on Windows.

I recommend doing everything in an interactive session

```
bsub -q interactive -W 12:00 –n 12 bash
```

If you run long, computationally intensive commands on one of the logins shells (mezannine, balcony), it will likely be terminated and you will get a warning. The alternative is to submit jobs, but I think it's easier to run interactively and unless you need to process 1000s of samples of data, it doesn't save that much time to parallelize by submitting separate jobs.

Download data to Orchestra

Follow the instructions from the Biopolymers email to download the data. Since there is limited space on personal accounts, I store my data in the shared Elledge lab drive at `/n/data2/hms/genetics/elledge/`. You may need to ask Eric to give you access.

Using the automated pipeline I set up

Fill out metadata.xslx

Add rows to the “screens” worksheet in the “metadata.xslx” file for each sample you screened.

The “bpf_fc” and “bpf_lib” are the flowcell (fc) and library (lib) identifiers from the Biopolymers Facility (bpf). The “idx” is the sequence of the barcode used for that sample. You can find the index sequences in the “96_idx” and “IDX001..096” and “IDX097..192” worksheets.

The “library” and “amp_date” are the library used and the date the library was amplified. This is necessary because each amplification generates a new input count distribution. Make sure each (“library”, “amp_date”) has a single row labeled “input” for the “plate”.

“plate”, “row”, and “col” together uniquely identify a screen. Note that the column “screen” is automatically calculated based on these values.

“beads” identifies what kind of beads were used for the screen. For example, the same sample could be screened on both Protein A+G and anti-IgM beads.

“provider” and “label” together uniquely identify a sample. “Provider” refers to whom we got the sample from and “label” generally refers to the label on the tube.

Generate dependency makefiles

After updating “metadata.xlsx”, upload it to `/n/data2/hms/genetics/elledge/gjx1/screens/results` and replace the old version already there.

Then run the following command to automatically generate the dependencies in the `/n/data2/hms/genetics/elledge/gjx1/screens/results/mk` folder:

```
python generate_dependencies.py
```

Run make

Now you can use the makefile (`/n/data2/hms/genetics/elledge/gjx1/screens/results/makefile`) to automatically create files.

For example, to create the count file for the 01.A1 screen (all individual files are stored in the directory `/n/data2/hms/genetics/elledge/gjx1/screens/results/parts`)

```
make parts/01.A1.count.csv.gz
```

Or to create the combined zipval file for all the vir2 screens (all combined files are stored directly in `/n/data2/hms/genetics/elledge/gjx1/screens/results/`)

```
make vir2.zipval.csv.gz
```

The makefile should automatically figure out what steps need to be run. You can also parallelize using six cores using `make -j 6`.

Documentation for each step of data analysis pipeline

Align sequencing data to reference

Use Bowtie to align the reads to a reference sequence. Bowtie should automatically be installed on Orchestra at `/opt/bowtie/`.

Upload reference sequences FASTA file

Before you can align, you need to build an index for the reference sequences. To do this, create a FASTA file containing all the sequences you want to align to. Upload this file to Orchestra using SCP from the Terminal on Mac OSX or WinSCP on Windows.

Note: if you upload from Windows, you will have to convert the line endings using the command

```
sed -i 's/\r$//' REFERENCE.FASTA
```

where REFERENCE.fasta is the name of your reference files. If you don't do this, nothing will align, I think because bowtie-build doesn't read the sequence correctly when the line endings are wrong.

Build Bowtie index

To build the bowtie index use the bowtie-build command. For example:

```
/opt/bowtie/bowtie-build REFERENCE.fasta REFERENCE_OUTPUT_NAME
```

This will create a bowtie index from the file *REFERENCE.fasta*. It will save the output as *REFERENCE _OUTPUT_NAME.1.ebwt*, *REFERENCE_OUTPUT_NAME.2.ebwt*, etc.

Align

I keep the full alignment results as BAM files, generated using samtools. Samtools should automatically be installed. I use `/opt/samtools-1.1/bin/samtools`.

Use this pipeline to decompress the bzipped sequence data, uses bowtie to align, and finally outputs a sorted compressed BAM alignment file.

```
bzip2 -dc path/to/FASTQ_SEQUENCES.fastq.bz2 | /opt/bowtie/bowtie -n 3
-l 30 -e 1000 --tryhard --nomaqround --norc --best --sam --quiet
path/to/REFERENCE_OUTPUT_NAME - | /opt/samtools-1.1/bin/samtools view
-u - | /opt/samtools-1.1/bin/samtools sort -T BAM_OUTPUT_NAME.bam - -o
$@
```

where

path/to/FASTQ_SEQUENCES.fastq.bz2 is the path to the FASTQ file containing the sequencing reads

path/to/REFERENCE_OUTPUT_NAME is the path to bowtie index (without the .1.ebwt suffix)

BAM_OUTPUT_NAME.bam is the name of the bam file to which you will save the alignment results

For additional information on the parameters, see the Bowtie manual and the Samtools manual. In particular, you can use the -5 and -3 parameters on bowtie to trim adapter sequences from the 5' or 3' ends. However, I prefer to include the adapter sequences in the reference sequence to avoid such trimming.

Count reads

To count the number of reads for each sequence, I use the samtools idxstats command. Before using idxstats, it is necessary to index the bam file by running the command:

/opt/samtools-1.1/bin/samtools index BAM_OUTPUT_NAME.bam

This will create a file *BAM_OUTPUT_NAME.bai*, which is the index of *BAM_OUTPUT_NAME.bam*.

Now use the following pipeline to count the number of reads for each reference sequence and convert it into a csv file. The first column of the csv file will be the oligo id's, taken from the sequence names in the *REFERENCE.fasta* file. The second column is the number of times that oligo was read.

```
/opt/samtools-1.1/bin/samtool idxstats BAM_OUTPUT_NAME.bam | cut -f
1,3 | sed -e '/^\*\t/d' -e '1 i id\tSAMPLE_ID' | tr "\\t" ","
>COUNT_FILE.csv
```

Where SAMPLE_ID is the id of the sample and COUNT_FILE.csv is the name of the count file.

I compress the csv files with gzip to save space. To do this, run the command:

```
gzip COUNT_FILE.csv
```

which will compress the file and rename is as *COUNT_FILE.csv.gz*

Install Python packages

The rest of the analysis requires custom Python scripts that depend on various packages. The easiest way to install them is to use the Anaconda package manager.

Download the appropriate Anaconda installer:

```
wget https://repo.continuum.io/archive/Anaconda3-2.2.0-Linux-x86_64.sh
```

Then, install Anaconda by calling

```
bash Anaconda3-2.2.0-Linux-x86_64.sh
```

Anaconda should automatically install the necessary dependencies (numpy, scipy, matplotlib) by default.

Calculate zero-inflated p-values from counts

The python script *calc_zipval.py* that will calculate zero-inflated p-values for a set of output counts based on a set of input counts. Note that the directory *calc_zipval.py* sits in must also contain the file *zigp.py*, which contains the definition of the zero-inflated p-value model.

Run the script using this command

```
python calc_zipval.py OUTPUT.count.csv.gz INPUT.count.csv.gz
log_directory >OUTPUT.zipval.csv
```

OUTPUT.count.csv is the output read count

INTPUT.count.csv is the input read count

log_directory is a directory where the script will save several plots showing model fits

OUTPUT.zipval.csv is the resulting zero-inflated p-values. First column is oligo id, second is zipvalue

Again, I gzip these csv files to save space.

Call hits from replicate zero-inflated p-values

The python script *call_hits.py* will call hits based on replicate zero-inflated p-values.

```
python call_hits.py REPLICATE1.zipval.csv.gz REPLICATE2.zipval.csv.gz
THRESHOLD log_directory >OUTPUT.zihit.csv
```

REPLICATE1.zipval.csv.gz and *REPLICATE2.zipval.csv.gz* are the two replicate zero-inflated pvalue files

THRESHOLD is the threshold zero-inflated p-value for calling hits (2.3 for VirScan) *log_directory* is a directory where the script will save plots showing correlation between the replicates

OUTPUT.zihit.csv is the resulting hits. First column is oligo id, second is True/False

Again, I gzip these csv files to save space.

Calculate virus scores from hits

The python script *calc_scores.py* calculates virus scores using the maximum parsimony approach. In addition, it will filter out any hits found in at least 3 of the beads samples or only one of the serum samples.

```
python calc_scores.py ZIHITS.zihit.csv.gz METADATA.csv.gz
NHITS.BEADS.csv.gz NHITS.SAMPS.csv.gz GROUPING_LEVEL EPITOPE_LEN
>OUTPUT.ziscore.spp.csv
```

ZIHITS.zihit.csv.gz is the gzipped results of call_hits.py

METADATA.csv.gz is the file containing the metadata for the virus library

NHITS.BEADS.csv.gz is a two column gzipped csv file, column 1 is oligo id, column 2 is the number of beads samples in which that oligo was a hit

NHITS.SAMPS.csv.gz is a two column gzipped csv file, column 1 is oligo id, column 2 is the number of non-beads samples in which that oligo was a hit

GROUPING_LEVEL can be *Species* or *Organism*, depending on

EPITOPE_LEN should be 7, the length of a linear epitope

OUTPUT.ziscore.spp.csv is a two column csv file. First column is either Species or Organism, depending on *GROUPING_LEVEL*, and second column is the score.

Again, I gzip these csv files to save space.

Combine multiple csvs into one table

It's easier to work with one file containing data for all the samples rather than with 100s of files for each sample. The *concat_tables.py* script combines multiple csv files into one.

```
python concat_tables.py TABLE1.csv.gz TABLE2.csv.gz TABLE3.csv.gz …
>COMBINED.csv.gz
```

You can list as many tables as you want. The script will combine then using the first column as the index.