**DGRP RNA-seq Alignment and Analysis Pipeline – User Guide**

**Logan Everett, PhD – May 2017**

**I. OVERVIEW**

The DGRP-seq “pipeline” is a collection of scripts to rapidly process hundreds of samples through the primary alignment and analysis steps. These scripts were initially developed for the DGRP Baseline RNA-seq project, but should be ready to apply to similar data sets, especially RNA-seq data that will be directly compared to the Baseline data. Each script is designed to be flexible, but these commands may require further adaptation to other projects.

In particular, these scripts are designed for RNA-seq libraries with the following properties:

* Whole fly RNA
* Individual line and sex per sample (no line or sex pooling)
* RiboRNA-depleted libraries
* Single-end reads (NOT paired-end data)
* Strand-specific libraries using 2nd strand synthesis dUTP labeling and digestion (Meaning that the sequence reads are reverse complement to the actual transcript RNA sequence)
* Samples are barcoded and pooled
* Illumina v1.9 Fastq file naming schema and quality score encoding
* Replicate numbers refer to the **biological** replicate
* All technical replicates are resequenced from the same library prep with the same barcode (no replicate library preps from the same biological sample) – technical replicates should be given the same replicate number

In many cases, there are parameters already setup to handle other types of data (e.g. paired-end data), but these have not been fully tested, and are outside the scope of this document. Consult the header documentation in each script for available parameters that might be used to accommodate other types of RNA-seq libraries.

Additionally, the pipeline was designed to do several things well, at the cost of a straight-forward “single script” that does everything:

* At each step, many samples are run in parallel, to take advantage of multi-node HPC environments like hyperion
* Each step is run separately, for easy troubleshooting – otherwise the entire pipeline would need to be re-run if it hits an error towards the end. However it may be useful in the future to create a script that runs the complete pipeline on each individual sample.
* Additional samples can be added later, which allows rapid QC of each flowcell of data as it is sequenced, and rapid addition of new data (e.g. if some libraries need to be resequenced to get sufficient depth).

This guide assumes basic knowledge of a Unix-based server/HPC environment, such as the Hyperion cluster used by the Mackay/Anholt labs, including how to move files back and forth to the server, navigate the server file system, use the Unix command line through an SSH connection, and configure the .bash\_profile file to change PATH and other environment variables.

**II. INSTALLATION**

Most of the scripts are wrappers for other public open-source tools that are used at each step of the pipeline. Therefore, there are a number of tools that must be installed prior to running this analysis pipeline. If you are running the pipeline on Hyperion, some of the tools are already installed for all users, and you can use many of the other tools already installed in in the ljeveret user account (until that account is eventually deleted). However, some tools need to be installed separately for each user. If you are running this pipeline on a new server (not Hyperion), then you will need to make sure all required tools are installed. The version used for the Baseline project is noted in each case, and for data that is meant to be compared directly to Baseline it is best to use the same version of all tools. Installation instructions for each individual tool are listed below.

Obtaining the DGRPseq package of scripts

The first step is to copy the DGRPseq package of scripts into your own directory. This package of scripts is managed using git, a distributed version management system. Git allows the user to easily update the scripts whenever changes are available, but having your own copy will prevent you from having issues with active development, and also allows you to make your own personal changes to the scripts as needed. Git is already installed on Hyperion, and the official repository of the DGRPseq scripts is also stored on Hyperion, therefore all you need to do is identify where in your own user directory you want to store the scripts. For example, if you wish to store the scripts in a folder called Tools under your home directory, you would do the following:

mkdir ~/Tools (If the directory doesn’t exist yet)

cd ~/Tools

git clone /home/ljeveret/Tools/DGRPseq/.git/

This will create a new directory under your Tools directory, called DGRPseq. You should only run this command once, and it will copy the most up to date version of the packages. However, in the future, there may be bug fixes or additional useful features added to the pipeline, in which case you may want to sync your copy of the scripts to the newest version. To do this, run the following commands:

cd ~/Tools/DGRPseq

git pull

This will sync all updates to the packages to your private copy. If you wish to make your own changes to the DGRPseq scripts, you should read up on how to manage multiple branches of a repository using git.

SAMTools v1.1

The pipeline currently uses SAMTools v1.1. It has NOT been tested with other versions of SAMTools. On Hyperion, you can guarantee you are using this required version of SAMTools by adding the following to your PATH variable in your .bash\_profile:

/home/ljeveret/Tools/samtools-1.1/bin

CutAdapt v1.6

The pipeline currently requires CutAdapt v1.6, which is NOT the most recent version. A newer version has now been installed on Hyperion, therefore the user must install the older version in their home directory.

Instructions for Hyperion users:

First run the following command:

python /home/ljeveret/Tools/cutadapt-1.6/setup.py install --user

Then add the following to your PATH variable in your .bash\_profile:

/home/ljeveret/Tools/cutadapt-1.6/bin

For other computing environments, you can download v1.6 here:

<https://pypi.python.org/pypi/cutadapt/1.6>

Download the source tar.gz file, upload to your server, and decompress. Then cd into the cutadapt-1.6 directory, and run:

python setup.py build

python setup.py install –user

Then change your path to to point to the this version of cutadapt.

Note: It may also be possible to install specific versions of cutadapt using Anaconda, but I haven’t yet tested that.

BWA v0.7.10-r789

The pipeline has currently been tested with v0.7.10-r789. Different versions may give slightly different alignment results, and therefore when the goal is to compare to Baseline results, this exact version should be used. You can use the installation of this version already installed on Hyperion by adding the following to your PATH variable:

/home/ljeveret/Tools/bwa-0.7.10

STAR v2.4.0e

On Hyperion, you can use the version of STAR installed here by adding to your PATH variable:

/home/ljeveret/.local/bin

HTSeq v0.6.1p1

Each user must individually install HTSeq in their account, as follows. On Hyperion, copy the file /home/ljeveret/Tools/HTSeq-0.6.1p1.tar.gz into a location in your own home directory. Then decompress the archive and enter the resulting directory using these commands:

tar –xzf HTSeq-0.6.1p1.tar.gz

cd HTSeq-0.6.1p1.tar.gz

Then install HTSeq for your account with this command:

python setup.py install --user

R v3.1.1 and related packages

The pipeline has been tested using R v3.1.1, and requires a number of other packages to be installed, including Bioconductor. However, all users can use the version of R installed on the ljeveret user directory, which already has all necessary packages. Just add the following to your PATH variable:

/home/ljeveret/Tools/R-3.1.1/bin

This path is also hard-coded at the top of all R scripts in the pipeline. This will be need to be changed for porting the pipeline to other systems, but is sufficient for ensuring the correct installation of R is used on Hyperion.

FastQC v0.11.2 (Optional)

Several optional QC steps use FastQC. The version tested was v0.11.2. This version is already installed for all users on Hyperion.

Cufflinks v2.2.1 (Optional)

If you plan to search for new novel transcripts (beyond those already identified in the Baseline analysis), you will need cufflinks. The version used in the Baseline analysis and tested with this pipeline is v2.2.1, which is currently installed on Hyperion for all users.

Trinity v2.1.1 (Optional)

The Trinity de novo assembly tool is only needed if you wish to mine your unaligned reads for additional microbial or other exogenous sequences. The pipeline has been tested with Trinity v2.1.1, which you can use on Hyperion by adding the following to your PATH variable:

/home/ljeveret/Tools/trinityrnaseq-2.1.1

**III. ANALYSIS SETUP**

Project Setup

For the purposes of this pipeline, a “project” is defined as a complete and newly sequenced set of libraries that are meant to be analyzed together. A project may consist of multiple flowcells, conditions, and samples. In most cases, you do NOT need to completely reanalyze the Baseline data, as it can be incorporated in later steps. These instructions and the accompanying scripts are designed to help other people process new RNA-seq studies in a way that is directly comparable to the Baseline analysis. However, if you are planning to change major steps in the alignment portion of the pipeline, all data, including the Baseline data, should be processed the same way – such adaptation of the pipeline is outside the scope of this document.

To setup a new project, you should create a new directory, under your user directory, on Hyperion (or whichever server you are using). The folder should be given a name that is descriptive and specific to your project, e.g. if you are analyzing DGRP lines that have been sent into space, a good project directory name would be “DGRP\_Space\_RNA”. This folder will hold all intermediate and output files from running the pipeline. Throughout this document, the main project directory will be referred to as PROJECT\_HOME.

Next, copy the file “example\_project\_info.sh” into PROJECT\_HOME, and rename the file to “project\_info.sh”. This is a standard file that sets many project-level parameters. Many of these parameters can be left at their default value, but there are a few you need to set for your project, specifically:

**[WHAT ARE PROJECT PARAMS TO SET?]**

**[DETAILS ON OPTIONAL PARAMS TO ADJUST]**

Batch Setup

Each project is further subdivided into batches. Each batch corresponds to a unique set of fastq sequence files, and it is intended to organize libraries that were sequenced at the same time. This allows each run to be processed up to the QC steps as they are sequenced, and then additional sequencing data can be added later without having to re-run analysis on the previous data. However, if all sequencing data has already been collected, they can all go in a single batch for simplicity.

NOTE: Each fastq file (corresponding to a specific flowcell, lane, and barcode) should be listed in exactly ONE batch file. Resequenced libraries (the same sample with the same barcode, sequenced again on a new flowcell) can be in a new batch from the original sequence data from that same library. Replicate data from the same library is combined at a later step in the pipeline when all resequencing is complete.

Each batch is described in a separate file in the PROJECT\_HOME directory. The name of the file should always end in “.txt”, and the batch name will be everything before the .txt suffix. A good standard practice is to just name batches by their run date, for example “batch\_160104.txt” would contain all data sequenced on 1/4/16. Or they can be numbered in the order they are run, e.g. “batch1.txt”, “batch2.txt”, etc. Ultimately, batch names can be anything that makes sense to you as long as the file names end with .txt.

Each batch file should be tab-delimited text with **[7??]** columns, as follows:

**[PUT IN BATCH FORMAT]**

The file should have no header line, and no additional blank or comment lines. If you create the batch file in Excel, make sure to save as tab-delimited text, and then open in text editor such as TextWrangler (Mac) or JEdit (PC), and change the line delimiter to UNIX style – otherwise the file will not behave properly in the Unix environment on the server. Also make sure the last line of the file ends with a new line (most editors should show a single blank line at the end), otherwise the last sample will be missed by the batch processing script.

**[ALSO NEED INSTRUCTIONS HERE ON HOW TO SETUP THE TABLE THAT MAPS SAMPLE NAMES TO LINE, SEX, REP – could provide parse\_sample\_names.R as an example, table can also be created manually]**

Session Setup

Each time you log on to Hyperion or whatever server you are using, you are starting a new “session”. If you are running the tools locally, every time you open a new Terminal or Cygwin window, you are also starting a new session. The DGRP-seq tools require certain environment variables to be set, so that each tool has a consistent set of parameters and knows where to pick up the appropriate files. Many of these environment variables are defined in the project\_info.sh script described above, but they must be reloaded in each session by changing your working directory to PROJECT\_HOME, and then running the following command:

source project\_info.sh

This command should run with no output or errors. There are also several environment variables that you should define at the beginning of each new session that are NOT stored in project\_info.sh because they regularly need to be adapted to what step of the analysis you are on, and how heavily the server is currently being used.

If you are using Hyperion or any other slurm-based HPC cluster, then you should set the USENODE environment variable to specify which job nodes to use. As a general rule, you should always leave at least one node open for other users, so for example:

USENODE=”-x node1”

This will queue your jobs onto all nodes EXCEPT FOR node 1, ensuring that there is a node free for other users. Here are several other possible options for this environment variable:

USENODE=”-x node[1,2]”

The above setting leaves both nodes 1 and 2 open for other jobs.

USENODE=”-w node4”

The above setting will queue your jobs ONLY on node4. You can exclude multiple nodes with the -x option, but you should only specify a single node with the -w option. Also note, setting this environment variable does not do anything automatically, the same variable is referenced in all of the pipeline commands to enforce this rule, but it gives the user a simple way to specify how to use cluster resources at the start of each session.

Additionally, for the alignment phase of the pipeline, you need to specify the RUNBATCHES environment variable. This is an array listing which batches are currently being aligned or processed. If you are running the pipeline on a project for the first time, or if you are running a step that has not been run on any batches yet, set RUNBATCHES as follows:

RUNBATCHES=${ALLBATCHES[@]}

If you are adding one or more additional batches to a project, and need to process ONLY the new batches, run this command instead:

RUNBATCHES=("newbatch1" "newbatch2")

Where newbatch1 is the name of the first batch you are adding, and so on. You can list any number of batches here, just make sure each one is in quotes and separated by a space within the parentheses.

**IV. PRIMARY ALIGNMENT PIPELINE**

This section of the documentation walks the user through the primary alignment steps, starting from raw sequencing data in FastQ format, and ending with reads aligned to several target sequence databases, including the Drosophila reference genome. This portion also includes several quality control steps, which are optional but highly recommended. It is advisable to run this portion of the pipeline on each new flowcell (sample batch) as the data becomes available, rather than waiting for all sequence data to be collected. This allows the user to detect problems early in the process, and prioritize libraries for re-sequencing in cases of insufficient read depth.

**Step 1. (Optional) Initial QC Assessment**

First, it is advisable to run the following command to make sure that your project\_info.sh file contains the correct path to the raw fastq files, and that your batch files are set up appropriately:

for BATCH in ${RUNBATCHES[@]}

do

run\_batch\_task.sh \

--batch=$BATCH \

--task='ls -l $RAW\_FASTQ\_PATH/$sampleID.fastq.gz' \

> ${BATCH}\_precheck.log

done

This should run with no errors, and should produce a file named (BATCH)\_precheck.log for each of your batches. This verifies that all expected fastq files are present.

It is also a good idea to run FastQC on the raw fastq files to make sure there are no obvious problems with sequence quality or formatting. Run the following command block to run the FastQC tool on all libraries in your current set of batches:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p fastqc/$BATCH

logfile="fastqc/$BATCH/run\_fastqc\_slurm\_%A\_%a.out"

run\_batch\_task.sh --array \

--threads=1 \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE" \

--task='fastqc --extract --outdir fastqc/$argBatch $RAW\_FASTQ\_PATH/$sampleID.fastq.gz'

done

If you run squeue immediately after this command block, you should see multiple jobs in the queue (one for each sample in each of your current batches). WAIT until all jobs have finished before proceeding to the next command. Once all FastQC jobs have finished, run this command:

fastqc\_summary.sh fastqc

The output should start with the following two messages:

No key failures - GOOD!

All fastq files are same encoding type - GOOD!

These will be followed by a summary of the %GC distribution across libraries, the distribution of the read length across libraries, and the number of known and other over-represented sequences. It will also output tables (in the fastqc/ directory) of these values across all libraries that can be analyzed or visualized in Excel.

**Step 2. Adapter Trimming with CutAdapt**

The first required alignment step is to remove adapter sequences. Depending on read length and the quality of the size selection step, it is possible to have cases where the actual insert sequence is shorter than the read length, in which case the 3’ end of the read will contain part of the 3’ adapter sequence. This is a well-known problem in Illumina sequencing data and many tools have been developed to deal with it. This pipeline currently uses CutAdapt, and the parameters here have been tailored specifically to single-end reads using our standard adapter sequences.

Run the following block of commands to queue up a CutAdapt job for each library in your current batches:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $TRIMMED\_FASTQ\_PATH/$BATCH

logfile="$TRIMMED\_FASTQ\_PATH/$BATCH/run\_cutadapt\_slurm\_%A\_%a.log"

run\_batch\_task.sh --array \

--threads=1 \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE" \

--task='run\_cutadapt.sh --output=$TRIMMED\_FASTQ\_PATH/$argBatch --sample=$sampleID'

done

Immediately after this block of commands completes, the squeue command should show a new set of jobs in the queue. Once ALL of these jobs have finished running (that is, CutAdapt finishes running on ALL libraries), run the following command:

for BATCH in ${RUNBATCHES[@]}

do

summarize\_cutadapt.sh \

$TRIMMED\_FASTQ\_PATH/$BATCH/run\_cutadapt\_slurm\_\*.log \

> $TRIMMED\_FASTQ\_PATH/$BATCH.cutadapt.summary.txt

done

This should finish running with no output, but for each batch there should be a new file in the trimmed/ directory with tabulated data summarizing the cutadapt results (should have a line for each library with information on number of reads processed, number of reads filtered, etc.)

**Step 3. Cleanup of Residual Ribosomal RNA**

The next step is to filter out ribosomal RNA (rRNA) contamination. This step was designed for libraries prepared using the RiboZero kit or some other rRNA depletion kit (for polyA-amplified libraries, this step is probably not necessary, but also won’t hurt). rRNA depletion removes the majority of rRNA molecules in a library, but the depletion is not 100% complete, and therefore there will be some residual sequences in the library. Because rRNA is such a large portion of the total RNA initially extracted from a sample, small fluctuations in depletion efficiency can produce large fluctuations in the % of rRNA sequences seen in the final library. Thus, this filtering step was designed to remove all remaining rRNA sequences, quantify them (for troubleshooting purposes), and use only the remaining non-rRNA sequences for all further analysis, including library depth for normalization purposes.

The filtering is accomplished by aligning all sequences to a small database of the primary Drosophila melanogaster rRNA transcripts using the BWA alignment tool. This provides a much cleaner quantification of rRNA content in each library, as compared to genomic alignment, because the FlyBase/BDGP version 5 of the D. melanogaster genome contains incomplete assemblies of some rRNA sequences and many copies of others (This appears to have been corrected in version 6 of the reference genome, but we are not currently using that version). The library was produced by extracting the known rRNA transcripts from FlyBase and GenBank. The sequences that do not align to the special rRNA filtering database are then used for all subsequent analysis.

By default, the DGRP-seq pipeline is set up to use 4 threads per BWA job, which provides a good balance of CPU and memory usage on hyperion. You can adjust this by changing the DEFAULT\_THREAD\_COUNT parameter in project\_info.sh. For example, you can set this parameter to 1 to remove all multi-threading (more library alignment jobs will run in parallel, but they will run slower due to using only a single CPU thread). Conversely, setting this parameter to 20 will cause each BWA job to use an entire node, but each job should run very fast. Note that changing this parameter will affect all subsequent steps that use multi-threading.

To run the rRNA filtering step, run this command block:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $BWA\_RIBO\_PATH/$BATCH

logfile="$BWA\_RIBO\_PATH/$BATCH/run\_bwa\_batch\_ribo\_slurm\_%A\_%a.out"

run\_batch\_task.sh --array \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE -c $DEFAULT\_THREAD\_COUNT" \

--task='run\_bwa.sh --sample=$sampleID --batch=$argBatch'

done

After all resulting jobs from this block have finished running, you should check the log files to make sure there were no errors. You can also generate files that summarize the number of reads aligning to each rRNA sequence or set of sequences, by running the following command block (OPTIONAL):

for BATCH in ${RUNBATCHES[@]}

do

logfile="$BWA\_RIBO\_PATH/$BATCH/run\_bwa\_count\_slurm\_%A\_%a.out"

run\_batch\_task.sh --array \

--threads=1 \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE" \

--task='bwa\_count.py --verbose --nosplit bwa\_rRNA/$argBatch/$sampleID/output.bam > bwa\_rRNA/$argBatch/$sampleID/target\_counts.txt'

done

**Step 4. Alignment to Microbiome Database**

The next step is to separate out reads from RNA transcribed from microbial genomes rather than from the *D. melanogaster* genome. This step was designed for libraries prepared from whole flies, although it may be worth checking on libraries derived from specific fly organs as well. Whole flies harbor microbial species in their cut and on their exterior, and may also carry other latent infections (viral or Wolbachia) in other tissues. Previous analysis of the Baseline RNA-seq data has produced a library of the most common microbes seen in whole fly RNA-seq libraries thus far. This filtering step was designed to identify reads aligning to those microbial genomes for separate analysis (see Analysis of Genetically Variable Microbial Species in later portion of this document).

The filtering is accomplished by aligning all sequences to a database of the DGRP Microbiome identified to date. The sequences that do not align to the microbial database are then used for all subsequent analysis. To run the microbe filtering step, run this command block:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $BWA\_MICROBE\_PATH/$BATCH

logfile="$BWA\_MICROBE\_PATH/$BATCH/run\_bwa\_microbe\_slurm\_%A\_%a.out"

run\_batch\_task.sh --array \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE -c $DEFAULT\_THREAD\_COUNT" \

--task='run\_bwa.sh --sample=$sampleID --batch=$argBatch --mode=microbe'

done

Wait until all resulting jobs have completed to run the next step.

**Step 5. Alignment to Transposon Sequences (RepBase)**

The next step is to separate out reads transcribed from transposons. These sequences may not be fully assembled in the reference genome, and may appear at multiple locations, which complicates the quantification of transposons from the primary reference genome alignment. A database of known transposon sequences has already been compiled as part of RepBase [REFERENCE]. This filtering step was designed to identify reads aligning to transposon sequences for separate normalization and analysis (see Analysis of Genetically Variable Transposon Activity in later portion of this document).

The filtering is accomplished by aligning all sequences to the RepBase database of the transposons for *D. melanogaster*. The sequences that do not align to RepBase are then used for all subsequent analysis. To run the transposon filtering step, run this command block:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $BWA\_REPEAT\_PATH/$BATCH

logfile="$BWA\_REPEAT\_PATH/$BATCH/run\_bwa\_repeat\_slurm\_%A\_%a.out"

run\_batch\_task.sh --array \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE -c $DEFAULT\_THREAD\_COUNT" \

--task='run\_bwa.sh --sample=$sampleID --batch=$argBatch --mode=repeat'

done

Wait until all resulting jobs have completed to run the next step.

**Step 6. (Optional) Intermediate QC Assessment**

At this point in the pipeline, the fully filtered FastQ files (adapter trimmed, with rRNA, microbial, and transposon sequences removed) can be re-assessed with FastQC. The main purpose is to confirm that no libraries had major issues with undersized inserts after adapter trimming, and that most repetitive sequences have been resolved after filtering. But overall this step can probably be removed from the pipeline.

To run FastQC again on the fully filtered FastQ files, run this command block:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p fastqc\_repfiltered/$BATCH

logfile="fastqc\_repfiltered/$BATCH/run\_fastqc\_slurm\_%A\_%a.out"

run\_batch\_task.sh\

--array\

--threads=1\

--batch=$BATCH\

--sbatchopt="-o $logfile $USENODE "\

--task='fastqc --extract --outdir fastqc\_repfiltered/$argBatch $REPEAT\_FILTERED\_PATH/$argBatch/${sampleID}\_filtered.fastq.gz'

done

When all resulting jobs have completed, run this final step to tabulate the results and check for any major problems:

fastqc\_summary.sh fastqc\_repfiltered

The results should look similar to what was obtained at Step 1, except that there should be fewer over-represented sequences, and the read lengths will now be a range for each library. Note that at this step, each library should still have an upper limit of read length that is close to what was sequenced. If a library went from, e.g., 125bp read length in the raw FastQ file (step 1) to 50-75bp read length here, it indicates that inserts were much smaller than intended and this may create a batch effect if it is observed for an entire flowcell.

**Step 7. Alignment to Reference Genome**

The next step is to align filtered reads to the *D. melanogaster* reference genome. This alignment step is done using the STAR algorithm, rather than BWA, because we must account for the possibility that some reads span splice junctions. That is, portions of a read may align to two regions many kb apart on either side of an intron, which is a case that BWA is not designed to handle. The STAR algorithm on the other hand is designed to handle spliced alignments very rapidly.

The other important consideration here is that our reads are derived from DGRP lines, which contain many genetic variants compared to the line used for the reference genome. Thus, our alignments must also be tolerant to a relatively high number of mismatches. The parameters used for the STAR alignment step have already been optimized for this issue, and an additional step later in the pipeline tests for any remaining alignment bias and prevents these effects from inflating the estimation of expression level heritability (see Analysis of Genetically Variable Gene Expression).

To run STAR on the fully filtered FastQ files, run this command block:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $STAR\_PATH/$BATCH

logfile="$STAR\_PATH/$BATCH/run\_star\_batch\_slurm\_%A\_%a.log"

run\_batch\_task.sh\

--array\

--batch=$BATCH\

--sbatchopt="-o $logfile $USENODE -c $DEFAULT\_THREAD\_COUNT"\

--task='run\_star.sh --sample=$sampleID --batch=$argBatch --lane=$lane --name=$sampleName --flowcell=$flowcell'

done

Wait until all resulting jobs have completed to run the next step.

**Step 8. Count Reads in Known Gene Models**

STAR identifies the best alignments between each read and the reference genome. However, to get gene-level quantification, we need to assign each read to the gene model it best overlaps. This is done using a tool called HTSeq-count, which compares read alignments to gene models, and applies several simple rules for removing reads with ambiguous assignments. A detailed explanation of the rules that HTSeq-count applies and the logic behind these rules is available here: <http://www-huber.embl.de/HTSeq/doc/count.html>

To run HTSeq-count on all STAR alignments, run this command block:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $HTSEQ\_COUNT\_PATH/$BATCH

logfile="$HTSEQ\_COUNT\_PATH/$BATCH/run\_htseq\_batch\_star\_slurm\_%A\_%a.log"

run\_batch\_task.sh --array \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE -c 1" \

--task='run\_htseq.sh --sample=$sampleID --batch=$argBatch --suffix=STAR\_counts'

done

Note that HTSeq-count will compare the read alignments to whatever set of gene models is specified in the FLY\_KNOWN\_GFF parameter of the project\_info.sh file. If you alter this parameter, and need to update the results of this step, just add the parameter --overwrite immediately after run\_htseq.sh in the command block above, and re-run. Make sure to re-run for ALL samples.

Wait until all jobs finish running before continuing to the next step.

**Step 9. Tabulation of Quality Control Parameters**

Although this is partly considered a QC step, it is required because the read totals computed here are used to build important tables needed for normalization in the subsequent analysis phases. This step is also important to determine that libraries have been sequenced to the desired depth even after filtering out rRNA, microbial, and transposon reads. To tabulate important QC stats for all current batches, first run this command block:

for BATCH in ${RUNBATCHES[@]}

do

sbatch $USENODE \

-o $TRIMMED\_FASTQ\_PATH/$BATCH/fastq.gz.stats \

-e $TRIMMED\_FASTQ\_PATH/$BATCH/fastq.gz.stats.err \

file\_stats.sh $TRIMMED\_FASTQ\_PATH/$BATCH/\*.fastq.gz

sbatch $USENODE \

-o $RIBO\_FILTERED\_PATH/$BATCH/fastq.gz.stats \

-e $RIBO\_FILTERED\_PATH/$BATCH/fastq.gz.stats.err \

file\_stats.sh $RIBO\_FILTERED\_PATH/$BATCH/\*.fastq.gz

sbatch $USENODE \

-o $MICROBE\_FILTERED\_PATH/$BATCH/fastq.gz.stats \

-e $MICROBE\_FILTERED\_PATH/$BATCH/fastq.gz.stats.err \

file\_stats.sh $MICROBE\_FILTERED\_PATH/$BATCH/\*.fastq.gz

sbatch $USENODE \

-o $REPEAT\_FILTERED\_PATH/$BATCH/fastq.gz.stats \

-e $REPEAT\_FILTERED\_PATH/$BATCH/fastq.gz.stats.err \

file\_stats.sh $REPEAT\_FILTERED\_PATH/$BATCH/\*.fastq.gz

done

Wait until all resulting jobs have finished (these will take a long time for batches that consist of whole flow cells). Then run the following commands:

cat $TRIMMED\_FASTQ\_PATH/\*/fastq.gz.stats \

> $TRIMMED\_FASTQ\_PATH/fastq.gz.stats

cat $RIBO\_FILTERED\_PATH/\*/fastq.gz.stats \

> $RIBO\_FILTERED\_PATH/fastq.gz.stats

cat $MICROBE\_FILTERED\_PATH/\*/fastq.gz.stats \

> $MICROBE\_FILTERED\_PATH/fastq.gz.stats

cat $REPEAT\_FILTERED\_PATH/\*/fastq.gz.stats \

> $REPEAT\_FILTERED\_PATH/fastq.gz.stats

When these commands finish, run the following command to tabulate all QC parameters into a single table (note: this will tabulate all batches indicated in project\_info.sh regardless of what current batches were run in previous steps):

sbatch -o collect\_sample\_QC\_stats.Rout $USENODE \

collect\_sample\_QC\_stats.R ${ALLBATCHES[@]}

When the step above finishes, there will be a new file called sample\_QC\_stats.txt, which is a tab-delimited text file that can be viewed in Excel. You should look at several important QC parameters here. The most important is to look at the UNIQUE.ALIGN.READS column and make sure that every library has at least ~5 million reads. Libraries that are substantially below this cutoff should either be resequenced on another flowcell, or the library should be remade (in which case, the low-depth library should be removed from analysis).

To identify the cause of low-depth libraries, you can first look at TOTAL.READS column – this shows how deeply the library was sequenced overall. If this value is similar to other libraries, then the library prep is good but contains a high proportion of rRNA, microbial, and/or transposon sequences (you can look at FRAC.READS.RIBO, FRAC.READS.MICROBE, and FRAC.READS.REPEAT to assess which type of read was an issue).

You should also check that all samples have similar values in READ.LEN.MEAN and READ.LEN.MEDIAN columns – lower values here indicate problems with insert lengths. For other columns, such as FRAC.READS.TRIMMED, FRAC.BASES.TRIMMED, FRAC.MULTI.ALIGN, FRAC.TOO.SHORT, and FRAC.KNOWN.GENES, you should not see drastic differences between flowcells/batches. You can visualize the values in each column, both within each flowcell (grouped by lane) and across all flowcells, by running this command:

sbatch -o plot\_sample\_QC\_stats.Rout $USENODE \

plot\_sample\_QC\_stats.R

**Step10. Sample Label Validation (Optional)**

This step is technically optional (it does not produce anything required for subsequent steps), however, it is HIGHLY RECOMMENDED. This step runs several scripts that validate each sample is labeled with the correct genotype and sex.

The first step is to count alleles at all known DGRP SNPs based on reads in the RNA-seq libraries. Run this command block to run this process on each sample in the current batches:

for BATCH in ${RUNBATCHES[@]}

do

mkdir -p $PILEUP\_PATH/$BATCH/

logfile="$PILEUP\_PATH/$BATCH/run\_count\_alleles\_slurm\_%A\_%a.log"

run\_batch\_task.sh --array \

--batch=$BATCH \

--sbatchopt="-o $logfile $USENODE" \

--task='count\_alleles\_rna.sh --sample=$sampleID --outpath='$PILEUP\_PATH/$BATCH/' --header '$STAR\_PATH/$BATCH/'$sampleID/Aligned.sortedByCoord.out.bam'

done

Wait until all resulting jobs have finished. The next step is to run a script comparing the observed alleles in each RNA-seq library against the known alleles at each SNP from the DGRP genotyping study. The following command block runs this script on all current batches, details of what the script does are provided below:

for BATCH in ${RUNBATCHES[@]}

do

logfile="$PILEUP\_PATH/$BATCH/validate\_line\_labels.Rout"

sbatch -o $logfile $USENODE validate\_line\_labels.R $BATCH

done

For each RNA-seq library, this script looks only at alleles with multiple supporting reads, and a clear homozygous call. For each of these alleles, it computes the percentage of SNPs with a mismatched allele against each DGRP line. If the sample is labeled correctly, it will usually score best (lowest error rate) against that line compared to all other DGRP lines by at least an order of magnitude. There are exceptions, most notably samples from lines 303 and 306 which are genetically very similar. The results of these comparisons are reported in the genotype\_check.txt file as follows:

BEST.LINE indicates the DGRP line with the best (lowest) error rate of mismatched SNPs, and BEST.ERR indicates the error rate (should generally be < 0.001). If BEST.LINE does not match the currently labeled line, the sample is marked as having a possible labeling error.

NEXT.LINE and NEXT.ERR give the line and error rate respectively of the second best match. If the line is labeled correctly, NEXT.ERR should generally be > BEST.ERR\*10. When BEST.LINE is correct, but NEXT.ERR is < BEST.ERR\*10, the library is marked with a WARN flag indicating the need to take a closer look. In some cases, NEXT.ERR may be close to BEST.ERR, which usually indicates cross-contamination of another line stock or inadvertent mixing of two libraries. The notable exception of this is lines 303 and 306 which are genetically very similar.

An additional test is to compute the ratio of mismatched SNPs for the labeled line vs each other possible line. For each other possible line, the comparison is limited to only SNPs that distinguish the currently labeled line from the alternate line. The RATIO.LINE column in genotype\_check.txt indicates the best guess for an alternate line compares to the currently labeled line based on this comparison. RATIO.ERR gives the ratio of the error rates, and when this value is >1, it indicates that the current line label is most likely correct (ideally this value will be >10, but for closely related lines or low coverage libraries this may not be the case). A RATIO.ERR < 1 indicates that RATIO.LINE is likely to be the correct line label, and that the library is currently mislabeled.

To draw several plots of the distributions of BEST.ERR and RATIO.ERR, run this command:

genotype\_validation\_plots.R ${ALLBATCHES[@]} \

&> genotype\_validation\_plots.Rout

The tests above should be sufficient to identify any libraries that have been labeled with the wrong DGRP line, but it does not tell us whether the library is labeled with the correct sex. To validate the sex labels, run the following command:

validate\_sex\_labels.R ${ALLBATCHES[@]} \

&> validate\_sex\_labels.Rout

This script uses the observation that in previous studies, the first principal component was heavily influenced by sex-specific expression. Thus, this script takes the gene-level read counts for all libraries, applies some basic read-depth normalization, and then performs PCA. PC1 is assumed to capture sex-specific differences. The script then performs Linear Discriminant Analysis (LDA) on PC1 using the current sex labels as the groups to be used for discrimination. If all samples are labeled correctly with regards to sex, then PC1 will fully separate all male and female libraries from each other, and the LDA will work with 100% precision. If a small number of samples are misclassified by the LDA, that is a strong indication that they have been given the wrong sex label. If PC1 does not separate male and female samples in general, that suggests there is a major batch effect or other technical artifact in the data that needs to be dealt with first. Finally, this script identifies edge cases based on the probability score from the LDA. That is, if the LDA classifies the sex correctly, but gives a lower confidence probability (<0.95), the script will issue a warning in validate\_sex\_labels.Rout. The script also produces a PCA plot with any misclassified and borderline cases highlighted. An example plot is show below:

Macintosh HD:Users:loganje:Projects:DGRP_3WK_RNAseq_Align:sample_QC_figures:libsex_PCA_plot.pdf

In this plot, red dots indicate female libraries, blue dots indicate male libraries. It is clear from the plot that PC1 fully separates male and female libraries, indicating that all libraries are labeled correctly. The one red dot with an extra circle around it indicates a borderline case where the LDA probability of classification was <0.95 – but in this case it was 0.93, which is still very strong and it is clear from the positioning that is similar to other female libraries. If the red and blue dots intermingle overall, there is a more serious problem with your libraries involving some other technical artifact such as a batch effect. If just a few dots are in the wrong cloud (e.g. a couple red dots in the blue cloud), these should be highlighted with extra circles, and you should look at the validate\_sex\_labels.Rout log file to see which libraries are mislabeled.

**Step 11. Clean Up Intermediate Files (Optional)**

This step is also optional but highly recommended to conserve space on Hyperion or any other server that you are using. You should only do this step AFTER completing all Step 9 above for all batches, and ideally after you have done all optional QC steps and confirmed that the Gene Quantification step in next section is working correctly. The main large files that are safe to clean-up are the intermediate fastq files generated at each step of the alignment procedure. Clean these files up as follows:

for BATCH in ${ALLBATCHES[@]}

do

rm trimmed/$BATCH/\*.fastq.gz

rm bwa\_rRNA/$BATCH/\*/\*\_mapped.fastq.gz

rm ribofiltered/$BATCH/\*.fastq.gz

rm bwa\_microbe/$BATCH/\*/\*\_mapped.fastq.gz

rm microbefiltered/$BATCH/\*.fastq.gz

rm repfiltered/$BATCH/\*.fastq.gz

done

**V. ANALYSIS OF GENETICALLY VARIABLE GENE EXPRESSION**

This section covers the joint analysis of all samples in a project. It can be run on a partial set of batches (before all sequencing has been completed) but all steps here need to be re-run whenever additional sequence data is added. It is generally best to wait until all sequencing and QC is completed before proceeding with this portion of the analysis. However, the same rules of project, batch, and session setup apply here as they do for the alignment portion of the pipeline.

**GENE QUANTIFICATION**

In order to facilitate combined analysis of gene expression jointly across all samples, the first steps are to build a table of all unique samples (including which libraries represent resequencing of the same sample and thus should be combined), as well as additional information about each sample needed for normalization (e.g. read depth at each stage of the alignment pipeline). This command first builds the summarization table of all samples from all batches:

sbatch -o build\_sample\_table.Rout \

build\_sample\_table.R ${ALLBATCHES[@]}

Make sure this script completes successfully by looking at the log file in build\_sample\_table.Rout. You should also see a new file called sample\_master\_table.txt which contains a list of all unique samples in your data set. The most common cause of error here is having two or more samples with the same name but different barcodes. This is not allowed because the only replicates that should exist in the dataset should be resequencing of the same library (and thus should have the same barcode). Typical causes of samples with the same sample name but different barcodes are:

1. You may have collected more biological replicates than intended from one or more lines, and/or given them the same replicate number. This is OK, but you should first give each sample the appropriate replicate number and sample name.
2. You prepared two or more libraries from the same biological sample. In this case, remove the sample with lower read depth before continuing (combining technical replicates that representative different library preps is currently not handled in this pipeline).
3. One or more samples were labeled incorrectly, make sure you have run genotype validation QC step and resolved any errors and warnings there.

The next step is to build a table with read counts for all genes across all samples. This is the table that will be used for further normalization, and then ultimate for the genetic variance models. At this step, replicate libraries (same library sequenced on multiple flowcells or lanes to get sufficient depth) will be combined to get total read counts at the level of biological replicates only.

sbatch -o build\_count\_table.Rout build\_count\_table.R

Make sure this step completes successfully by checking the build\_count\_table.Rout log file. You should also see a new file in htseq directory called combined\_samples\_gene\_counts.txt. This file contains the final tabulated read counts for each gene across all samples.

**SAMPLE NORMALIZATION**

# Standard normalization (now includes TMM!)

# Normalizes to column sums

# TO DO: Keep LOW and RARE features in at this stage?

# Do normalized output files still get FLAG column?

LOGFILE=expression/normalize\_expression.Rout

normalize\_expression.R COUNTS=expression/$COUNTSTUB"\_counts.txt" OUTDIR=expression GENES=expression/combined-gene-info.txt CUTOFF=FIT &> $LOGFILE

# ONLY NEED TO RUN THIS ONCE:

# Variant density of the KNOWN genes

sbatch $USENODE -o compute\_variant\_density\_known.Rout compute\_variant\_density.R ~/Resources/FlyBase/Dmel\_r5.57\_FB2014\_03/gff/dmel-all-transcriptome-r5.57-plus-r6.11-backport.gff known\_gene

**GENETIC VARIANCE MODELS**

mkdir -p expression/genVar

# Primary H^2 analysis (Within Sex) - the main analysis uses FPKM values, skips NQ transform, corrects for variant rate AND Wolbachia

# NOTE: Parallelization auto-detects the number of CPUs given to jobs

sbatch $USENODE -c 20 -o expression/genVar/gen\_var\_model\_fpkm\_VR\_WolAdj.Rout gen\_var\_model.R EXPR=expression/$COUNTSTUB"\_fpkm.txt" FILTER=LOW LINEREG=expression/combined\_gene\_variant\_rates.txt TAG=VR OUTDIR=expression/genVar/

# Pooled Sex H^2 Analysis

# RE-RUNNING WITH FLAG VERSION ON 1/18/17

# TO DO: Should do VR correction within sex? Or include a SEX:LINEREG ixn term?

sbatch $USENODE -c 20 -o expression/genVar/gen\_var\_model\_fpkm\_VR\_Pooled\_Wol.Rout gen\_var\_model.R POOLED=TRUE EXPR=expression/$COUNTSTUB"\_fpkm.txt" FILTER=LOW LINEREG=expression/combined\_gene\_variant\_rates.txt TAG=VR OUTDIR=expression/genVar/

# eQTL H^2 Analysis (Within Sex) - Same as primary analysis but includes additional model terms for major inversions and cryptic relatedness PCs

# RE-RUNNING WITH FLAG VERSION ON 1/18/17

# These line means should be used for eQTL mapping

sbatch $USENODE -c 20 -o expression/genVar/gen\_var\_model\_fpkm\_VR\_eQTL.Rout gen\_var\_model.R EXPR=expression/$COUNTSTUB"\_fpkm.txt" FILTER=LOW EQTL=T LINEREG=expression/combined\_gene\_variant\_rates.txt TAG=VR OUTDIR=expression/genVar/

# Plots from primary model

grep '^[FG]' expression/genVar/$COUNTSTUB"\_fpkm\_VR\_WolAdj\_model\_results.txt" > expression/genVar/combined\_samples\_split\_known\_fpkm\_VR\_WolAdj\_model\_results.txt

grep '^[XG]' expression/genVar/$COUNTSTUB"\_fpkm\_VR\_WolAdj\_model\_results.txt" > expression/genVar/combined\_samples\_split\_novel\_fpkm\_VR\_WolAdj\_model\_results.txt

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_known\_novel\_fpkm\_VR\_WolAdj\_model\_results.txt &> expression/genVar/plot\_all\_heritability\_FPKM.Rout

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_split\_known\_fpkm\_VR\_WolAdj\_model\_results.txt &> expression/genVar/plot\_known\_heritability\_FPKM.Rout

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_split\_novel\_fpkm\_VR\_WolAdj\_model\_results.txt &> expression/genVar/plot\_novel\_heritability\_FPKM.Rout

# Plots from pooled model

grep '^[FG]' expression/genVar/$COUNTSTUB"\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt" > expression/genVar/combined\_samples\_split\_known\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt

grep '^[XG]' expression/genVar/$COUNTSTUB"\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt" > expression/genVar/combined\_samples\_split\_novel\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_known\_novel\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt &> expression/genVar/plot\_all\_pooled\_FPKM.Rout

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_split\_known\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt &> expression/genVar/plot\_known\_pooled\_FPKM.Rout

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_split\_novel\_fpkm\_VR\_WolAdj\_Pooled\_model\_results.txt &> expression/genVar/plot\_novel\_pooled\_FPKM.Rout

# Plots from eQTL model

grep '^[FG]' expression/genVar/$COUNTSTUB"\_fpkm\_VR\_eQTL\_model\_results.txt" > expression/genVar/combined\_samples\_split\_known\_fpkm\_VR\_eQTL\_model\_results.txt

grep '^[XG]' expression/genVar/$COUNTSTUB"\_fpkm\_VR\_eQTL\_model\_results.txt" > expression/genVar/combined\_samples\_split\_novel\_fpkm\_VR\_eQTL\_model\_results.txt

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_known\_novel\_fpkm\_VR\_eQTL\_model\_results.txt &> expression/genVar/plot\_all\_eQTL\_FPKM.Rout

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_split\_known\_fpkm\_VR\_eQTL\_model\_results.txt &> expression/genVar/plot\_known\_eQTL\_FPKM.Rout

Rscript plot\_bicolor\_hist.R expression/genVar/combined\_samples\_split\_novel\_fpkm\_VR\_eQTL\_model\_results.txt &> expression/genVar/plot\_novel\_eQTL\_FPKM.Rout

# --- Principal Component Analysis --- #

# TO DO: Copy this over to microbiome and transposons

mkdir -p expression/pca

sbatch $USENODE -c 20 -o expression/pca/expression\_pca\_fpkm\_VR\_WolAdj\_F.Rout expression\_PCA.R EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_line\_means.txt

sbatch $USENODE -c 20 -o expression/pca/expression\_pca\_fpkm\_VR\_WolAdj\_M.Rout expression\_PCA.R EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_line\_means.txt

# --- Trait Correlation --- #

# This uses a more standardized script and does linear regression

# TO DO: Add spearman correlations into this version?

# TO DO: Add plots to the core script here?

# TO DO: Lots of other potential improvements, see TO DO items in the script itself

mkdir -p expression/traitreg

# Run against individual microbe expression features

sbatch $USENODE -o expression/traitreg/trait\_regression\_${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F.Rout trait\_regression.R EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_line\_means.txt

sbatch $USENODE -o expression/traitreg/trait\_regression\_${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M.Rout trait\_regression.R EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_line\_means.txt

# Run against PCs

sbatch $USENODE -o expression/traitreg/trait\_regression\_${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_PCs.Rout trait\_regression.R EXPR=expression/pca/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_PCs.txt

sbatch $USENODE -o expression/traitreg/trait\_regression\_${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_PCs.Rout trait\_regression.R EXPR=expression/pca/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_PCs.txt

# --- Run WCGNA --- #

# This is an alternate Clustering Method to MMC

mkdir -p expression/wgcna

# Filter out LOW/RARE genes, and filter by avg expr, H2, and VAR as well:

filter\_line\_means.R MODE=wgcna EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_line\_means.txt H2FILE=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_model\_results.txt FILTER=RARE,LOW AVGEXPR=0 VAR=0.05 &> expression/wgcna/filter\_all\_F.Rout

filter\_line\_means.R MODE=wgcna EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_line\_means.txt H2FILE=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_model\_results.txt FILTER=RARE,LOW AVGEXPR=0 VAR=0.05 &> expression/wgcna/filter\_all\_M.Rout

# Run WGCNA on F and M

# TO DO: Plenty of room for improvement or alternate things to try here,

# See TO DO list on run\_wgcna.R

sbatch $USENODE -c 20 -o expression/wgcna/run\_wgcna\_F.Rout run\_wgcna.R EXPR=expression/wgcna/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_line\_means\_wgcna.txt

sbatch $USENODE -c 20 -o expression/wgcna/run\_wgcna\_M.Rout run\_wgcna.R EXPR=expression/wgcna/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_line\_means\_wgcna.txt

# Plot results

plot\_mmc\_results.R EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_line\_means.txt MMC=expression/wgcna/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_F\_wgcna.csv &> expression/wgcna/${COUNTSTUB}\_fpkm\_Wol\_F\_plot\_wgcna\_results.Rout &

plot\_mmc\_results.R EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_line\_means.txt MMC=expression/wgcna/${COUNTSTUB}\_fpkm\_VR\_WolAdj\_M\_wgcna.csv &> expression/wgcna/${COUNTSTUB}\_fpkm\_Wol\_M\_plot\_wgcna\_results.Rout &

eQTL Analysis and Network Construction

This part of the pipeline maps eQTLs for all genes, and constructs a regulatory network based on overlapping cis and trans eQTLs. Mapping eQTLs is first achieved using the fast QTL mapping tool Plink, and permutations are used to determine appropriate FDR cutoffs for each expression feature.

The first step is to setup genotype files that are properly formatted for plink. These files can be be re-used for eQTL mapping of microbiome and transposon features below, and therefore this step only needs to be run once for all 3 analyses. This script sets up the appropriate plink genotype files for the lines used in your project:

setup\_plink.sh

This should produce a new file called freeze2.200line.common.snp. If you are using fewer than 200 lines, the file name will be slightly different (e.g. 190line), make sure to edit all commands below to use the filename appropriate for your project.

This script maps all SNPs to nearby genes:

sbatch $USENODE -c 20 -o build\_SNP\_gene\_map.Rout \

build\_SNP\_gene\_map.R SNP=freeze2.200line.common.snp

Wait until this script finishes (it may take a long time, but it is also parallelized to take advantage of an entire node). It will speed up all subsequent steps because the SNP-gene mapping does not have to be re-done after this.

Next, setup plink “phenotype” files containing line means for all genetically variable expression features. The instructions below show how to do this for male and female expression patterns separately, since gene expression patterns usually show pervasive sex effects and sex by line interactions. This script also produces permuted versions of the phenotype files (scrambled line labels) which are necessary for computing appropriate FDR cutoffs. The number of permutations to be performed is stored in the PNUM environment variable. All commands using this variable should be run in a single session. PNUM=10 is sufficient for preliminary testing, but PNUM=100 is ideal for more robust FDR estimates.

These scripts should run quickly and log files should not indicate any errors. You should now see several .pheno files in expression/plink and matching files in expression/plink/Perm\* subdirectories.

PNUM=10

mkdir -p expression/plink

filter\_line\_means.R \

EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_eQTL\_F\_line\_means.txt \

FILTER=RARE,LOW MODE=eqtl PERM=$PNUM \

&> expression/plink/plink\_setup\_F.Rout

filter\_line\_means.R \

EXPR=expression/genVar/${COUNTSTUB}\_fpkm\_VR\_eQTL\_M\_line\_means.txt \

FILTER=RARE,LOW MODE=eqtl PERM=$PNUM \

&> expression/plink/plink\_setup\_M.Rout

Next, submit the individual plink jobs (primary and permutations) to the job queue (these can all be run in parallel):

sbatch $USENODE \

-o expression/plink/plink\_run\_F.log \

run\_plink.sh \

--genotypes=freeze2.200line.common \

--phenotypes=expression/plink/${COUNTSTUB}\_fpkm\_VR\_eQTL\_F.pheno \

--output=expression/plink/ExprF

sbatch $USENODE \

-o expression/plink/plink\_run\_M.log run\_plink.sh \

--genotypes=freeze2.200line.common \

--phenotypes=expression/plink/${COUNTSTUB}\_fpkm\_VR\_eQTL\_M.pheno \

--output=expression/plink/ExprM

for ((perm=1;perm<=PNUM;perm++))

do

sbatch $USENODE \

-o expression/plink/Perm${perm}/plink\_run\_F.log \

run\_plink.sh \

--genotypes=freeze2.200line.common \

--phenotypes=expression/plink/Perm${perm}/${COUNTSTUB}\_fpkm\_VR\_eQTL\_F.pheno \

--output=expression/plink/Perm${perm}/ExprF

sbatch $USENODE \

-o expression/plink/Perm${perm}/plink\_run\_M.log \

run\_plink.sh \

--genotypes=freeze2.200line.common \

--phenotypes=expression/plink/Perm${perm}/${COUNTSTUB}\_fpkm\_VR\_eQTL\_M.pheno \

--output=expression/plink/Perm${perm}/ExprM

done

Depending on the number of genetically variable expression features selected for analysis, these jobs may take a long time (possibly a day or more) to run. Wait until all jobs have finished before proceeding.

Plink produces a separate .qassoc file for each expression feature in each analysis. This next step combines all .qassoc files into a single file for each analysis, and then uses the combined files for FDR analysis against the permutations.

sbatch $USENODE \

-o expression/plink/eQTL\_tabulate.Rout \

eQTL\_tabulate.R expression/plink/

When the job above finishes, run this command to compute which eQTLs pass an FDR cutoff of 5%:

sbatch $USENODE \

-o expression/plink/eQTL\_perm\_fdr.Rout \

eQTL\_perm\_fdr.R expression/plink/

When you are sure these steps have run successfully (no error messages in logs, new files tabulating all eQTLs), it is safe to remove all individual qassoc files from expression/plink/

Next, run the following scripts to separate the eQTLs into different categories based on their proximity to genes. These can be run in parallel (the two commands are just running on each sex separately):

sbatch -c 20 $USENODE \

-o expression/plink/eQTL\_summary\_F.Rout \

eQTL\_summary.R \

EQTL=expression/plink/ExprF.0.05.fdr.results.txt \

MAP=freeze2.200line.common.snp.gene.map

sbatch -c 20 $USENODE \

-o expression/plink/eQTL\_summary\_M.Rout \

eQTL\_summary.R \

EQTL=expression/plink/ExprM.0.05.fdr.results.txt \

MAP=freeze2.200line.common.snp.gene.map

These scripts take the full set of eQTLs passing 5% FDR threshold for each sex, and split them into the following categories:

1. cis eQTLs – all eQTLs within 1kb of the gene body of the affected expression feature
2. trans eQTLs – all eQTLs within 1kb of one or more other gene bodies, but >1kb from the gene body of the affected expression feature
3. distal eQTLs – all eQTLs >1kb from all gene bodies – these are most likely to correspond to distal regulatory elements such as enhancers.

The next set of commands use the cis and trans eQTL tables to construct a gene regulatory network. Again, these can be run in parallel (first command is female eQTLs, second is males):

sbatch -c 20 $USENODE \

-o expression/plink/eQTL\_network\_F.Rout eQTL\_network.R \

EQTL=expression/plink/ExprF.0.05.fdr.trans.eqtls.txt

sbatch -c 20 $USENODE \

-o expression/plink/eQTL\_network\_M.Rout eQTL\_network.R \

EQTL=expression/plink/ExprM.0.05.fdr.trans.eqtls.txt

The resulting network is determined as follows. First, all trans eQTLs are overlapped with cis eQTLs. For every case where the same SNP is both a cis eQTL for gene X and a trans eQTL for gene Y, an edge is drawn in the regulatory network indicating that gene X is likely to regulate gene Y (or “X 🡪 Y”). Note that a single edge in the network can be based on multiple SNPs, if there are multiple SNPs meeting the criteria (cis eQTL for X, trans eQTL for Y). All relevant SNPs are recorded in the network table.

Next, the resulting network is split into disjoint subnetworks, such that there are no edges between any of the subnetworks. This typically results in one primary subnetwork covering most of the genes with eQTLs, and then a handful of much smaller subnetworks. In order to determine “hub genes” in the network, it is necessary to break any loops (e.g. cases where X 🡪 Y and Y 🡪 X, although loops can involve any number of genes). To do this, edges are prioritized based on the strongest p-value among all relevant SNPs. Edges with the weakest (largest) p-values are dropped to break loops until none remain.

Once all loops are broken, each gene is analyzed to determine:

1. The number of direct targets, e.g. the number of genes inferred to be directly regulated by X in the network.
2. The total number of downstream targets, e.g. the number of direct targets of X, plus the number of their direct targets, etc.
3. The number of direct regulators, e.g. the number of genes inferred to be directly regulating X in the network.
4. The total number of upstream regulators, e.g. the number of direct regulators, plus the number of their direct regulators, etc.

Genes that are “master regulators” or hubs of the network should have large numbers of total downstream targets and a small number of total upstream regulators. The script outputs a table describing these properties for all genes in the eQTL network, and another table describing the overall size and density of each disjoint subnetwork.

**VI. IDENTIFICATION OF NOVEL TRANSCRIPTS**

**VII. ANALYSIS OF GENETICALLY VARIABLE MICROBIAL SPECIES**

**VIII. ANALYSIS OF GENETICALLY VARIABLE TRANSPOSON ACTIVITY**