Supplementary Information

Ava Hoffman, Jenny Cocciardi

Contents

Introduction

In this experiment, we used quaddRAD library prep to prepare the sample DNA. This means that there were both two unique outer barcodes (typical Illumina barcodes) *AND* two unique inner barcodes (random barcode bases inside the adapters) for each sample - over 1700 to be exact!

The sequencing facility demultiplexes samples based on the outer barcodes (typically called 5nn and i7nn). Once this is done, each file still contains a mix of the inner barcodes. We will refer to these as "sublibraries"

because they are sort of halfway demultiplexed. We separate them out bioinformatically later.

Raw Data File Naming - Sublibraries

Here's a bit of information on the file name convention. The typical raw file looks like this:

`AMH_macro_1_1_12px_S1_L001_R1_001.fastq.gz`

- These are author initials and "macro" stands for "Macrosystems". These are on every file.

  `AMH_macro`

- The first number is the *i5nn* barcode for the given sublibrary. We know all these samples have a i5nn barcode "1", so that narrows down what they can be. The second number is the *i7nn* barcode for the given sublibrary. We know all these samples have a i7nn barcode "1", so that further narrows down what they can be.

  `1_1`

- This refers to how many samples are in the sublibrary. "12px" means 12-plexed, or 12 samples. In other words, we will use the inner barcodes to further distinguish 12 unique samples in this sublibrary.

  `12px`

- This is a unique sublibrary name. S1 = 1 i5nn and 1 i7nn.

  `S1`

- This means this particular file came from lane 1 of the NovaSeq. There are four lanes. All samples should appear across all four lanes.

  `L001`

- This is the first (R1) of two paired-end reads (R1 and R2).

  `R1`

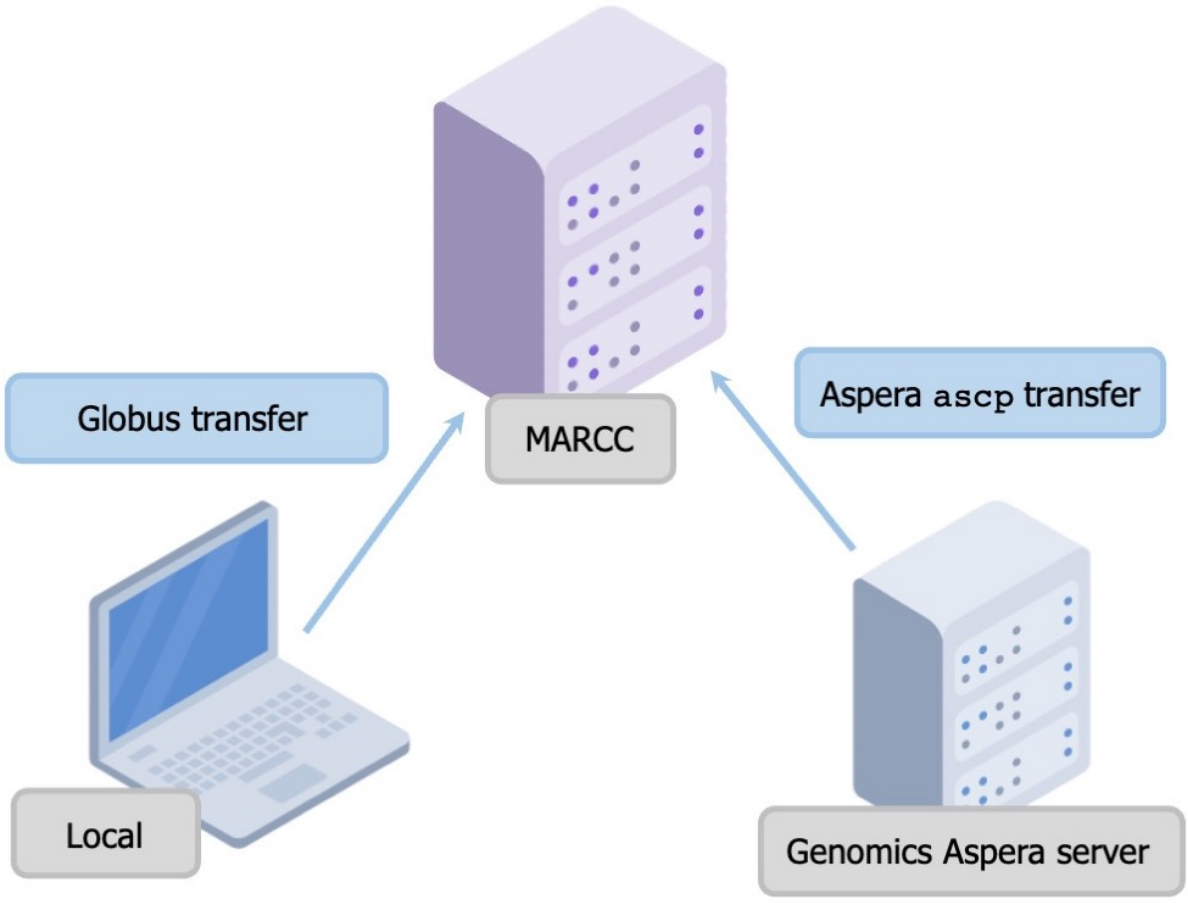
- The last part doesn't mean anything - it was just added automatically before the file suffix (`fastq.gz`)

  `001.fastq.gz`

A Note on File Transfers

There are three main systems at play for file transfer: the local machine, the sequencing facility's (GRCF) Aspera server, and MARCC. The Aspera server is where the data were/are stored immediately after sequencing. MARCC is where we plan to do preprocessing and analysis. Scripts and text files are easy for me to edit on my local machine. We used Globus to transfer these small files from my local machine to MARCC.

Midway through this analyses, we transitioned to another cluster, JHU's Rockfish. Scripts below, with the exception of file transfer from the Aspera server, should reflect the new filesystem, though you will have to adjust the file paths accordingly.



Preprocessing

Step 1 - Transfer Files

Files can be found in the `01_transfer_files/` directory.

`01-aspera_transfer_n.txt`

These are text files containing the *names* of `fastq.gz` files that we wanted to transfer from the sequencing facility's Aspera server to the computing cluster (MARCC). This was to maximize ease of transferring only certain files over at once, since transferring could take a long time. We definitely did this piecemeal. Possible file names shown in Aspera Transfer File Names. There are multiple of these files so that we could parallelize (replace n with the correct number in the command used below). This text file will need to be uploaded to your scratch directory in MARCC.

Files were then transferred using the following commands. Before starting, make sure you are in a data transfer node. Then, load the aspera module. Alternatively, you can install the Aspera transfer software and use that.

```
module load aspera
```

Initiate the transfer from within your scratch directory:

```
ascp -T -l8G -i /software/apps/aspera/3.9.1/etc/asperaweb_id_dsa.openssh
--file-list=01-aspera_transfer_n.txt
```

```
--mode=recv --user=<aspera-user> --host=<aspera-IP> /scratch/users/<me>@jhu.edu
```

Step 2 - Concatenate Files and Install Stacks

Files can be found in the `02_concatenate_and_check/` directory.

Step 2a - Concatenate Files for each Sublibrary

We ran my samples across the whole flow cell of the NovaSeq, so results came in 8 files for each demultiplexed sublibrary (4 lanes * paired reads). For example, for sublibrary 1_1, we'd see the following 8 files:

```
AMH_macro_1_1_12px_S1_L001_R1_001.fastq.gz
AMH_macro_1_1_12px_S1_L001_R2_001.fastq.gz
AMH_macro_1_1_12px_S1_L002_R1_001.fastq.gz
AMH_macro_1_1_12px_S1_L002_R2_001.fastq.gz
AMH_macro_1_1_12px_S1_L003_R1_001.fastq.gz
AMH_macro_1_1_12px_S1_L003_R2_001.fastq.gz
AMH_macro_1_1_12px_S1_L004_R1_001.fastq.gz
AMH_macro_1_1_12px_S1_L004_R2_001.fastq.gz
```

The `02_concatendate_and_check/02-concat_files_across4lanes.sh` script finds all files in the working directory with the name pattern `*_L001_*.fastq.gz` and then concatenates across lanes 001, 002, 003, and 004 so they can be managed further. The "L001" part of the filename is then eliminated. For example the 8 files above would become:

```
AMH_macro_1_1_12px_S1_R1.fastq.gz
AMH_macro_1_1_12px_S1_R2.fastq.gz
```

Rockfish uses slurm to manage jobs. To run the script, use the `sbatch` command. For example:

```
sbatch ~/code/02-concat_files_across4lanes.sh
```

This command will run the script from within the current directory, but will look for and pull the script from the code directory. This will concatenate all files within the current directory that match the loop pattern.

Step 2b – Download and Install Stacks

On Rockfish, Stacks will need to be downloaded to each user's code directory. Stacks, and software in general, should be compiled in an interactive mode or loaded via module. For more information on interactive mode, see `interact --usage`.

```
interact -p debug -g 1 -n 1 -c 1
module load gcc
```

Now download Stacks. We used version 2.60.

```
wget http://catchenlab.life.illinois.edu/stacks/source/stacks-2.60.tar.gz
tar xfvz stacks-2.60.tar.gz
```

Next, go into the stacks-2.60 directory and run the following commands:

```
./configure --prefix=/home/<your_username>/code4-<PI_username>
make
make install
export PATH=$PATH:/home/<your_username>/code4-<PI_username>/stacks-2.60
```

The filesystem patterns on your cluster might be different, and you should change these file paths accordingly.

Step 3 - Remove PCR Clones

Files can be found in the `03_clone_filter/` directory.

Step 3a - Run PCR Clone Removal Script

The `03-clone_filter.sh` script runs `clone_filter` from Stacks. The program was run with options `--inline_inline --oligo_len_1 4 --oligo_len_2 4`. The `--oligo_len_x 4` options indicate the 4-base pair degenerate sequence was included on the outside of the barcodes for detecting PCR duplicates. The script uses the file name prefixes listed for each single sub-pooled library in `03-clone_filter_file_names.txt` and loops to run `clone_filter` on all of them. Possible file names shown in `clone_filter` File Names.

Step 3b - Parse PCR Clone Removal Results

If you want to extract descriptive statistics from the `clone_filter` output, you can use the `03.5-parse_clone_filter.py` script to do so. It can be run on your local terminal after transferring the `clone_filter.out` logs to your local computer.

```
source("03_clone_filter/examine_clones.R")
make_cloneplot()
```

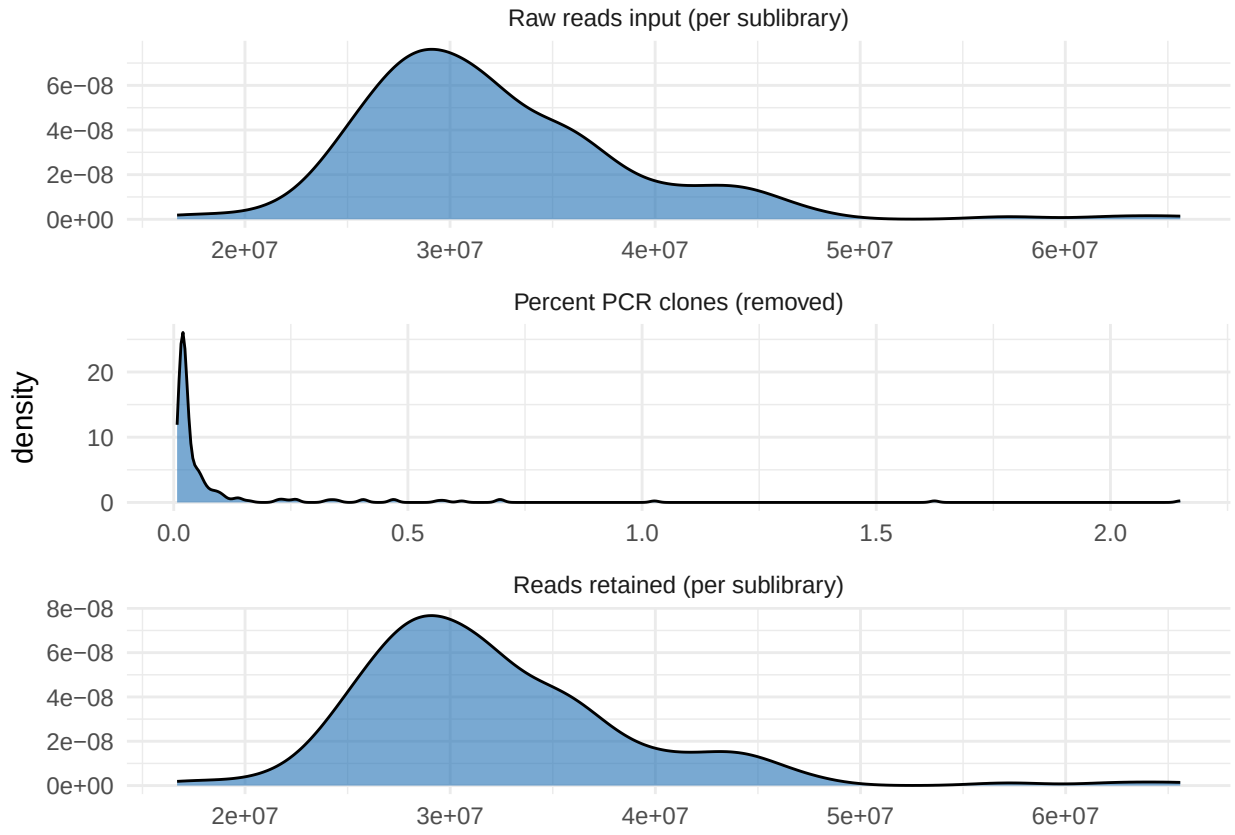


Figure S1: PCR clone removal statistics

Step 4 - Demultiplexing and Sample Filtering

Files can be found in the `04_demux_filter/` directory.

Step 4a - Demultiplex and Filter

The `04-process_radtags.sh` script runs `process_radtags` from Stacks. The program was run with options `-c -q --inline_inline --renz_1 pstI --renz_2 mspI --rescue --disable_rad_check`. The script uses the same file prefixes as Step 3 - `03-clone_filter.sh`. Each sub-pooled library has a forward

and reverse read file that was filtered in the previous step. Like the above section, the script uses the file name prefixes listed for each single sub-pooled library in `04-process_radtags_file_names.txt` and loops to run `process_radtags` on all of them. Possible file names shown in `clone_filter` File Names.

Each sub-pooled library also has a demultiplexing file (`04-demux/` directory) that contains the sample names and inner(i5 and i7) barcodes. For example, the sublibrary 1_1, we'd see the following barcode file:

```
ATCACG  AGTCAA  DS.BA.PIK.U.1
CGATGT  AGTTCC  DS.BA.PIK.U.2
TTAGGC  ATGTCA  DS.BA.PIK.U.3
TGACCA  CCGTCC  DS.BA.PIK.U.4
ACAGTG  GTCCGC  DS.BA.PIK.U.5
GCCAAT  GTGAAA  DS.BA.DHI.U.1
CAGATC  GTGGCC  DS.BA.DHI.U.2
ACTTGA  GTTTCG  DS.BA.DHI.U.3
GATCAG  CGTACG  DS.BA.DHI.U.4
TAGCTT  GAGTGG  DS.BA.DHI.U.5
GGCTAC  ACTGAT  DS.BA.GA.U.1
CTTGTA  ATTCCT  DS.BA.GA.U.2
```

The 'process_radtags' command will demultiplex the data by separating out each sublibrary into the individual samples. It will then clean the data, and will remove low quality reads and discard reads where a barcode was not found.

Step 4b - Organize files

In a new directory, make sure the files are organized by species. In the `process_radtags` script, we specified that files be sent to `~/scratch/demux/*sublibrary_name*` (reasoning for this is in Step 4c), but files should manually be organized into species folders (i.e., `~/scratch/demux/*SPP*`) after `process_radtags` is performed. For example, the file "DS.MN.L01-DS.M.1.1.fq.gz" should be sent to the `~/scratch/demux/DS` directory.

Note: this is not automated at this point but it would be nice to automate the file moving process so it's not forgotten at this point.

Step 4c - Assess the raw, processed, and cleaned data

In the script for Step 4, we have specified that a new output folder be created for each sublibrary. The output folder is where all sample files and the log file will be dumped for each sublibrary. It is important to specify a different output folder if you have multiple sublibraries because we will be assessing the output log for each sublibrary individually (and otherwise, the log is overwritten when the script loops to a new sublibrary).

The utility `stacks-dist-extract` can be used to extract data from the log file. First, we examined the library-wide statistics to identify sublibraries where barcodes may have been misentered or where sequencing error may have occurred. We used:

```
stacks-dist-extract process_radtags.log total_raw_read_counts
```

to pull out data on the total number of sequences, the number of low-quality reads, whether barcodes were found or not, and the total number of retained reads per sublibary. Look over these to make sure there are no outliers or sublibraries that need to be checked and rerun.

Next, we used:

```
stacks-dist-extract process_radtags.log per_barcode_raw_read_counts
```

to analyze how well each sample performed. There are three important statistics to consider for each sample.

1. *The proportion of reads per sample for each sublibrary* indicates the proportion that each individual was processed and sequenced within the overall library. This is important to consider as cases where a single sample dominates the sublibrary may indicate contamination.
2. *The number of reads retained for each sample* can be an indicator of coverage. It is most likely a good idea to remove samples with a very low number of reads. Where you decide to place the cutoff for low coverage samples is dependent on your dataset. For example, a threshold of 1 million reads is often used but this is not universal.
3. *The proportion of reads retained for each sample* can also indicate low-quality samples and will give an idea of the variation in coverage across samples.

Output for sublibraries for this step are summarized in `process_radtags-library_output.csv`.

Output for individual samples for this step are summarized in `process_radtags-sample_output.csv`.

The script `04c-process_radtags_stats.R` was used to create many plots for easily assessing each statistic. Output from this step can be found in `figures/process_radtags/` where figures are organized by species.

```
source("04_demux_filter/04c-radtags_filter_summary.R")
make_filterplot()
```

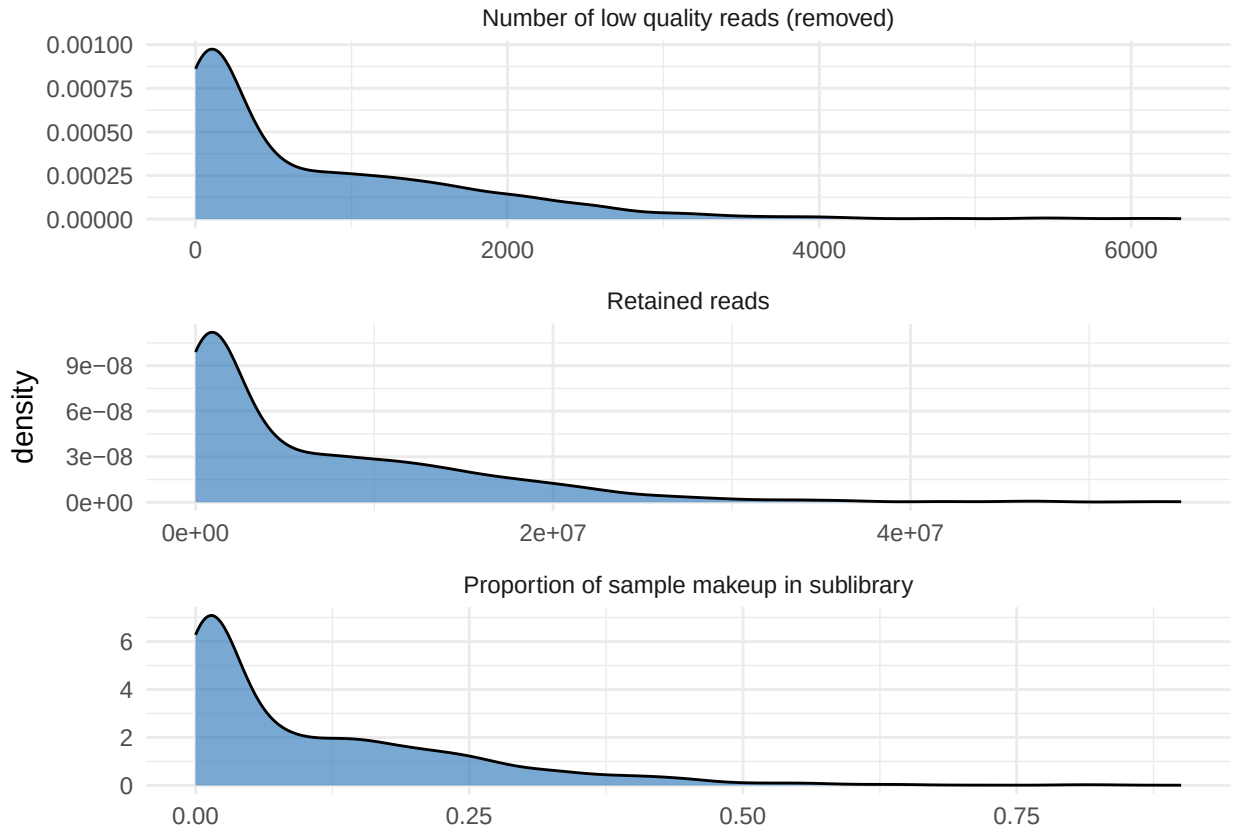


Figure S2: RAD tag processing statistics

Step 4d - Identify low-coverage and low-quality samples from `downstream analysis`

Using the same output log and the above statistics, we removed low-coverage and low-quality samples that may skew downstream analyses.

Samples were identified and removed via the following procedure:

1. First, samples that represented less than **1% of the sequenced sublibrary** were identified and removed. These samples correlate to low-read and low-coverage samples.
2. Next, a threshold of **1 million retained reads per sample** was used to remove any remaining low-read samples. Low-read samples correlate to low coverage and will lack enough raw reads to contribute to downstream analyses.

Good/kept samples are summarized in `process_radtags-kept_samples.csv`.

Discarded samples are summarized in `process_radtags-discarded_samples.csv`.

```
source("04_demux_filter/04c-radtags_filter_summary.R")
make_manual_discard_plot()
```

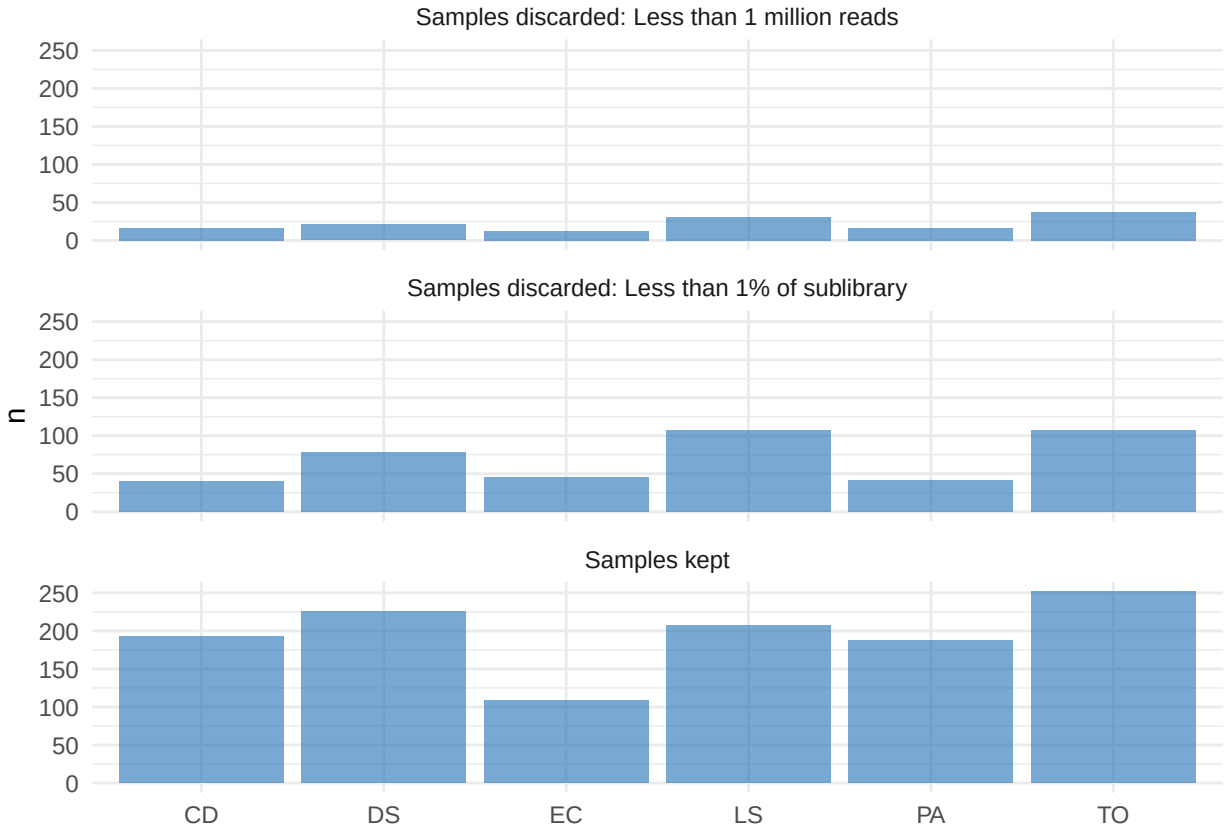


Figure S3: RAD tag manual filtering summary

Note: At this point, we started using Stacks 2.62 for its multi-threading capabilities. Functionality of the previous steps should be the same, however.

Generating Stacks Catalogs and Calling SNPs

Step 5 - Metapopulation Catalog Building and Parameter Search

Files can be found in the `05_ustacks_and_params/` directory.

Going forward, when we use the term **metapopulation**, we are referring to the collection of all samples within species among all cities where the species was present.

It is important to conduct preliminary analyses that will identify an optimal set of parameters for the dataset (see Step 5a). Following the parameter optimization, the program `ustacks` can be run to generate a catalog of loci.

Step 5a - Run `denovo_map.sh`

Stack assembly will differ based on several different aspects of the dataset(such as the study species, the RAD-seq method used, and/or the quality and quantity of DNA used). So it is important to use parameters that will maximize the amount of biological data obtained from stacks.

There are three main parameters to consider when doing this:

1. m = controls the minimum number of raw reads required to form a stack(implemented in `ustacks`)
2. M = controls the number of mismatches between stacks to to merge them into a putative locus (implemented in `ustacks`)
3. n = controls the number of mismatches allowed between stacks to merge into the catalog (implemented in `cstacks`)

There are two main ways to optimize parameterization:

1. an iterative method were you sequentially change each parameter while keeping the other parameters fixed (described in *Paris et al. 2017*), or
2. an iterative method were you sequentially change the values of M and n(keeping $M = n$) while fixing $m = 3$, and then test $m = 2, 4$ once the optimal $M = n$ is determined(described in *Rochette and Catchen 2017*, *Catchen 2020*).

We performed the second method and used the `denovo_map.sh` script to run the `denovo_map.pl` command to perform iterations. This script requires that we first choose a subset of samples to run the iterations on. The samples should be representative of the overall dataset; meaning they should include all populations and have similar read coverage numbers. Read coverage numbers can be assessed by looking at the descriptive statistics produced from Step 4c.

Place these samples in a text file (`popmap_test_samples.txt`) with the name of the sample and specify that all samples belong to the same population. For example, `popmap_test_samples.txt` should look like...

```
DS.BA.GA.U.1    A
DS.PX.BUF.M.5   A
DS.B0.HC4.M.1   A
...
```

It is important to have all representative samples treated as one population because you will assess outputs found across 80% of the individuals. The script will read this text file from the `--popmap` argument.

The script also requires that you specify an output directory after `-o`. This should be unique to the parameter you are testing... for example, if you are testing $M = 3$, then you could make a subdirectory labeled `stacks.M3` where all outputs from `denovo_map.sh` will be placed. Otherwise, for each iteration, the outputs will be overwritten and you will lose the log from the previous iteration. The `denovo_map.sh` script also requires that you direct it toward where your samples are stored, which is your directory built in Step 4b. Make sure to run the `--min-samples-per-pop 0.80` argument.

To decide which parameters to use, examine the following from each iteration:

1. the average sample coverage: This is obtained from the summary log in the `ustacks` section of `denovo_map.log`. If samples have a coverage <10x, you will have to rethink the parameters you use here.
2. the number of assembled loci shared by 80% of samples: This can be found in the `haplotypes.tsv` by counting the number of loci: `cat populations.haplotypes.tsv | grep -v ^"#" | wc -l`

3. the number of polymorphic loci shared by 80% of samples: This can be found in `populations.sumstats.tsv` or by counting `populations.hapstats.tsv`: `cat populations.hapstats.tsv | grep -v "^#" | wc -l`
4. the number of SNPs per locus shared by 80% of samples: found in `denovo_map.log` or by counting the number of SNPs in `populations.sumstats.tsv`: `populations.sumstats.tsv | grep -v ^"#" | wc -l`

The script `05a-param_opt-figures_script.R` was used to create plots for assessing the change in shared loci across parameter iterations.

Based on this optimization step, we used the following parameters:

Table S1: Final parameter optimization values for the Stacks pipeline.

Species	M (locus mismatches)	n (catalog mismatches)	m (minimum reads)
CD	8	8	3
DS	10	10	3
EC	8	8	3
LS	7	7	3
PA	5	5	3
TO	6	6	3

Step 5b - Run `ustacks`

`ustacks` builds *de novo* loci in each individual sample. We have designed the `ustacks` script so that the process requires three files:

- `05-ustacks_n.sh` : the shell script that executes `ustacks`
- `05-ustacks_id_n.txt` : the sample ID number
- `05-ustacks_samples_n.txt` : the sample names that correspond to the sample IDs

The sample ID should be derived from the `order_id` column(first column) on the master spreadsheet. It is unique (1-1736) across all of the samples.

The sample name is the corresponding name for each sample ID in the spreadsheet. E.g., sample ID "9" corresponds to sample name "DS.BA.DHI.U.4". Sample naming convention is species.city.site.management_type.replicate_plant.

`05-ustacks_n.sh` should have an out_directory (`-o` option) that will be used for all samples (e.g., `stacks/ustacks`). Files can be processed piecemeal into this directory. There should be three files for every sample in the output directory:

- `<samplename>.alleles.tsv.gz`
- `<samplename>.snps.tsv.gz`
- `<samplename>.tags.tsv.gz`

Multiple versions of the `05-ustacks_n.sh` script can be run in parallel (simply replace n in the three files above with the correct number).

A small number of samples (13) were discarded at this stage as the `ustacks` tool was unable to form any primary stacks corresponding to loci. See output/ustacks-discarded_samples.csv.

Step 5c - Correct File Names

This step contains a script `05b-fix_filenames.sh` which uses some simple regex to fix filenames that are output in previous steps. Stacks adds an extra "1" at some point at the end of the sample name which is not meaningful. The following files:

- DS.MN.L02-DS.M.3.1.alleles.tsv.gz
- DS.MN.L03-DS.U.2.1.tags.tsv.gz
- DS.MN.L09-DS.U.1.1.snps.tsv.gz

become:

- DS.MN.L02-DS.M.3.alleles.tsv.gz
- DS.MN.L03-DS.U.2.tags.tsv.gz
- DS.MN.L09-DS.U.1.snps.tsv.gz

The script currently gives some strange log output, so it can probably be optimized/improved. The script should be run from the directory where the changes need to be made. Files that have already been fixed will not be changed.

Step 5d - Choose catalog samples/files

In the next step, we will choose the files we want to go into the catalog. This involves a few steps:

1. Create a meaningful directory name. This could be the date (e.g., `stacks_22_01_25`).
2. Copy the `ustacks` output for all of the files you want to use in the reference from Step 5b. Remember this includes three files per sample. So if you have 20 samples you want to include in the reference catalog, you will transfer 3 x 20 = 60 files into the meaningful directory name. The three files per sample should follow this convention:

- `<samplename>.alleles.tsv.gz`
- `<samplename>.snps.tsv.gz`
- `<samplename>.tags.tsv.gz`

3. Remember the meaningful directory name. You will need it in Step 6.

Step 6 - Metapopulation catalog with `cstacks`

Files can be found in the `06_cstacks/` directory.

`cstacks` builds the locus catalog from all the samples specified. The accompanying script, `cstacks_SPECIES.sh` is relatively simple since it points to the directory containing all the sample files. It follows this format to point to that directory:

```
cstacks -P ~/directory ...
```

Make sure that you use the meaningful directory from Step 5c and that you have copied all the relevant files over. Otherwise this causes problems downstream. For example, you might edit the code to point to `~/scratch/stacks/stacks_22_01_25`.

```
cstacks -P ~/scratch/stacks/stacks_22_01_25 ...
```

The tricky thing is ensuring enough compute memory to run the entire process successfully. There is probably space to optimize this process.

The `cstacks` method uses a "population map" file, which in this project is `cstacks_popmap_SPECIES.txt`. This file specifies which samples to build the catalog from and categorizes them into your 'populations', or in this case, cities using two tab-delimited columns, e.g.:

```
DS.BA.GA.U.1    Baltimore
DS.BA.GA.U.2    Baltimore
DS.BA.GA.U.3    Baltimore
DS.BA.GA.U.4    Baltimore
DS.BA.GA.U.5    Baltimore
...
```

Make sure the samples in this file correspond to the input files located in e.g., `~/scratch/stacks/stacks_22_01_25`.

`cstacks` builds three files for use in all your samples (in this pipeline run), mirroring the sample files output by`ustacks`:

- `catalog.alleles.tsv.gz`
- `catalog.snps.tsv.gz`
- `catalog.tags.tsv.gz`

Table S2: Subset of samples used in SNP catalog creation.

Sample	Species	City
DS.BA.PIK.U.1	DS	BA
DS.BA.GA.U.4	DS	BA
DS.BA.LH-1.M.4	DS	BA
DS.BA.LH-3.M.1	DS	BA
DS.BA.WB.U.2	DS	BA
DS.BA.LL-4.M.5	DS	BA
DS.BA.LH-2.M.5	DS	BA
DS.BA.TRC.U.3	DS	BA
DS.BA.W3.M.2	DS	BA
DS.BA.RG-1.M.1	DS	BA
DS.BA.LL-3.M.3	DS	BA
DS.BA.RG-2.M.4	DS	BA
DS.BO.HC1.M.3	DS	BO
DS.BO.HC4.M.5	DS	BO
DS.BO.LC1.M.3	DS	BO
DS.BO.LC2.M.2	DS	BO
DS.BO.LC3.M.5	DS	BO
DS.BO.WL1.M.2	DS	BO
DS.BO.WL2.M.1	DS	BO
DS.BO.WL3.M.5	DS	BO
DS.BO.I4.U.1	DS	BO
DS.BO.R1.U.4	DS	BO
DS.BO.R2.U.2	DS	BO
DS.BO.R4.U.4	DS	BO
DS.MN.L05-DS.M.3	DS	MN
DS.MN.L09-DS.M.3	DS	MN
DS.MN.L11-DS.M.1	DS	MN
DS.MN.L02-DS.U.1	DS	MN
DS.MN.L02-DS.M.4	DS	MN
DS.MN.L03-DS.U.3	DS	MN
DS.MN.L04-DS.U.5	DS	MN
DS.MN.L06-DS.U.3	DS	MN
DS.MN.L07-DS.U.3	DS	MN
DS.MN.L09-DS.U.3	DS	MN
DS.MN.L11-DS.U.1	DS	MN
DS.MN.L11-DS.U.5	DS	MN
DS.PX.BUF.M.1	DS	PX
DS.PX.PIE.M.2	DS	PX
DS.PX.ALA.M.1	DS	PX
DS.PX.MTN.M.6	DS	PX
DS.PX.LAP.M.3	DS	PX
DS.PX.NUE.M.4	DS	PX
DS.PX.WES.M.2	DS	PX

Sample	Species	City
DS.PX.DF1.M.1	DS	PX
DS.PX.ENC.M.1	DS	PX
DS.PX.DOW.M.1	DS	PX
DS.PX.DOW.M.4	DS	PX
DS.PX.DF2.M.3	DS	PX
CD.BA.LA.U.2	CD	BA
CD.BA.TRC.U.3	CD	BA
CD.BA.WGP.M.2	CD	BA
CD.BA.LH-2.M.2	CD	BA
CD.BA.LL-4.M.1	CD	BA
CD.BA.PIK.U.2	CD	BA
CD.BA.WB.U.2	CD	BA
CD.BA.CP.U.4	CD	BA
CD.BA.FH.U.1	CD	BA
CD.BA.PSP.M.4	CD	BA
CD.BA.AA.U.4	CD	BA
CD.BA.RG-1.M.2	CD	BA
CD.BA.W3.M.3	CD	BA
CD.BA.GA.U.3	CD	BA
CD.BA.WBO.U.5	CD	BA
CD.LA.WHI.M.3	CD	LA
CD.LA.SEP.M.3	CD	LA
CD.LA.SEP.M.4	CD	LA
CD.LA.ROS.M.5	CD	LA
CD.LA.MR2.M.2	CD	LA
CD.LA.ALL.M.2	CD	LA
CD.LA.ALL.M.5	CD	LA
CD.LA.VAL.M.5	CD	LA
CD.LA.HAR.M.4	CD	LA
CD.LA.LUB.M.3	CD	LA
CD.LA.GLO.M.4	CD	LA
CD.LA.ZOO.M.3	CD	LA
CD.LA.NWH.M.5	CD	LA
CD.LA.KIN.M.3	CD	LA
CD.LA.KIN.M.5	CD	LA
CD.PX.CAM.U.5	CD	PX
CD.PX.MON.U.5	CD	PX
CD.PX.PKW.U.5	CD	PX
CD.PX.LAP.M.4	CD	PX
CD.PX.NES.U.4	CD	PX
CD.PX.PAL.M.3	CD	PX
CD.PX.ASU.M.1	CD	PX
CD.PX.NUE.M.5	CD	PX
CD.PX.WES.M.3	CD	PX
CD.PX.MAN.M.4	CD	PX
CD.PX.CLA.M.3	CD	PX
CD.PX.DF1.M.5	CD	PX
CD.PX.COY.M.5	CD	PX
CD.PX.RPC.M.3	CD	PX
CD.PX.ENC.M.2	CD	PX
EC.BA.LH-2.M.2	EC	BA
EC.BA.WBO.U.4	EC	BA

Sample	Species	City
EC.BA.WB.U.5	EC	BA
EC.BA.FH.U.3	EC	BA
EC.BA.CP.U.2	EC	BA
EC.BA.TRC.U.3	EC	BA
EC.BA.LL-4.M.4	EC	BA
EC.BA.WB.U.1	EC	BA
EC.BA.PIK.U.5	EC	BA
EC.BA.PSP.M.4	EC	BA
EC.BA.GA.U.2	EC	BA
EC.BA.LL-3.M.3	EC	BA
EC.BA.ML.U.1	EC	BA
EC.BA.TRC.U.5	EC	BA
EC.BA.ML.U.3	EC	BA
EC.LA.SGB.U.2	EC	LA
EC.LA.SGB.U.5	EC	LA
EC.LA.DUR.U.2	EC	LA
EC.LA.HOW.U.2	EC	LA
EC.LA.SAN.U.2	EC	LA
EC.LA.VER.U.1	EC	LA
EC.LA.VER.U.4	EC	LA
EC.LA.VB2.U.4	EC	LA
EC.LA.AC2.U.2	EC	LA
EC.LA.AC1.U.1	EC	LA
EC.LA.VB1.U.1	EC	LA
EC.LA.VB1.U.3	EC	LA
EC.LA.SGR.U.4	EC	LA
EC.LA.SGR.U.5	EC	LA
EC.LA.HOW.U.3	EC	LA
EC.PX.BUF.M.1	EC	PX
EC.PX.BUF.M.3	EC	PX
EC.PX.ALA.M.3	EC	PX
EC.PX.MTN.M.2	EC	PX
EC.PX.WES.M.1	EC	PX
EC.PX.WES.M.2	EC	PX
EC.PX.MAN.M.1	EC	PX
EC.PX.CLA.M.1	EC	PX
EC.PX.PSC.M.1	EC	PX
EC.PX.DF1.M.1	EC	PX
EC.PX.DOW.M.1	EC	PX
EC.PX.DOW.M.2	EC	PX
EC.PX.COY.M.2	EC	PX
EC.PX.COY.M.3	EC	PX
EC.PX.ALA.M.5	EC	PX
LS.BA.WB.U.1	LS	BA
LS.BA.WB.U.2	LS	BA
LS.BA.DHI.U.2	LS	BA
LS.BA.GA.U.1	LS	BA
LS.BA.PIK.U.3	LS	BA
LS.BA.PIK.U.5	LS	BA
LS.BA.CP.U.2	LS	BA
LS.BA.ML.U.2	LS	BA
LS.BA.WBO.U.3	LS	BA

Sample	Species	City
LS.BO.WL3.M.4	LS	BO
LS.BO.I1.U.1	LS	BO
LS.BO.I2.U.1	LS	BO
LS.BO.WL2.M.2	LS	BO
LS.BO.R1.U.2	LS	BO
LS.BO.R2.U.4	LS	BO
LS.BO.R3.U.3	LS	BO
LS.BO.HC4.M.3	LS	BO
LS.BO.LC4.M.2	LS	BO
LS.LA.VET.M.4	LS	LA
LS.LA.SSV.M.1	LS	LA
LS.LA.NAV.M.4	LS	LA
LS.LA.SHO.M.2	LS	LA
LS.LA.WES.M.3	LS	LA
LS.LA.GLO.M.3	LS	LA
LS.LA.HOW.U.5	LS	LA
LS.LA.SAN.U.2	LS	LA
LS.LA.ARR.U.2	LS	LA
LS.MN.L06-LS.U.2	LS	MN
LS.MN.L06-LS.U.5	LS	MN
LS.MN.L07-LS.U.4	LS	MN
LS.MN.L08-LS.U.5	LS	MN
LS.MN.L09-LS.U.3	LS	MN
LS.MN.L01-LS.M.4	LS	MN
LS.MN.L01-LS.U.3	LS	MN
LS.MN.L02-LS.U.1	LS	MN
LS.MN.L05-LS.U.2	LS	MN
LS.PX.MON.U.2	LS	PX
LS.PX.PKW.U.5	LS	PX
LS.PX.PIE.M.4	LS	PX
LS.PX.ALA.M.3	LS	PX
LS.PX.PAL.M.3	LS	PX
LS.PX.MAN.M.2	LS	PX
LS.PX.NUE.M.1	LS	PX
LS.PX.ENC.M.4	LS	PX
LS.PX.COY.M.3	LS	PX
PA.BA.PIK.U.1	PA	BA
PA.BA.LH-3.M.2	PA	BA
PA.BA.LH-3.M.3	PA	BA
PA.BA.WB.U.1	PA	BA
PA.BA.AA.U.1	PA	BA
PA.BA.WGP.M.3	PA	BA
PA.BA.LL-4.M.3	PA	BA
PA.BA.LA.U.2	PA	BA
PA.BA.LH-2.M.2	PA	BA
PA.BA.W3.M.3	PA	BA
PA.BA.RG-1.M.2	PA	BA
PA.BA.LL-3.M.5	PA	BA
PA.BO.I2.U.3	PA	BO
PA.BO.HC1.M.4	PA	BO
PA.BO.R3.U.2	PA	BO
PA.BO.HC4.M.5	PA	BO

Sample	Species	City
PA.BO.R4.U.2	PA	BO
PA.BO.WL2.M.5	PA	BO
PA.BO.WL4.M.4	PA	BO
PA.BO.LC4.M.4	PA	BO
PA.BO.HC2.M.1	PA	BO
PA.BO.R1.U.2	PA	BO
PA.BO.WL1.M.1	PA	BO
PA.BO.I1.U.5	PA	BO
PA.LA.ALL.M.5	PA	LA
PA.LA.SEP.M.1	PA	LA
PA.LA.SEP.M.5	PA	LA
PA.LA.WHI.M.2	PA	LA
PA.LA.ROS.M.5	PA	LA
PA.LA.LUB.M.2	PA	LA
PA.LA.GLO.M.2	PA	LA
PA.LA.ZOO.M.4	PA	LA
PA.LA.ZOO.M.5	PA	LA
PA.LA.NWH.M.2	PA	LA
PA.LA.KIN.M.4	PA	LA
PA.LA.POP.M.4	PA	LA
PA.PX.BUF.M.3	PA	PX
PA.PX.PIE.M.4	PA	PX
PA.PX.LAP.M.5	PA	PX
PA.PX.ALA.M.1	PA	PX
PA.PX.PAP.M.2	PA	PX
PA.PX.PAP.M.5	PA	PX
PA.PX.DF1.M.2	PA	PX
PA.PX.RPP.U.3	PA	PX
PA.PX.ENC.M.4	PA	PX
PA.PX.ENC.M.5	PA	PX
PA.PX.COY.M.1	PA	PX
PA.PX.BUF.M.2	PA	PX
TO.BA.WBO.U.4	TO	BA
TO.BA.CP.U.1	TO	BA
TO.BA.FH.U.1	TO	BA
TO.BA.LH-3.M.4	TO	BA
TO.BA.WGP.M.3	TO	BA
TO.BA.GA.U.4	TO	BA
TO.BA.PIK.U.4	TO	BA
TO.BA.PSP.M.1	TO	BA
TO.BA.RG-2.M.2	TO	BA
TO.BO.HC1.M.4	TO	BO
TO.BO.HC2.M.5	TO	BO
TO.BO.HC3.M.1	TO	BO
TO.BO.HC4.M.5	TO	BO
TO.BO.LC1.M.1	TO	BO
TO.BO.LC2.M.5	TO	BO
TO.BO.LC3.M.1	TO	BO
TO.BO.WL2.M.1	TO	BO
TO.BO.I2.U.3	TO	BO
TO.LA.WHI.M.5	TO	LA
TO.LA.HAR.M.4	TO	LA

Sample	Species	City
TO.LA.MR1.M.1	TO	LA
TO.LA.GLO.M.5	TO	LA
TO.LA.ZOO.M.1	TO	LA
TO.LA.NWH.M.4	TO	LA
TO.LA.VNS.M.2	TO	LA
TO.LA.PEP.M.5	TO	LA
TO.LA.COM.M.4	TO	LA
TO.MN.L11-TO.M.3	TO	MN
TO.MN.L02-TO.U.1	TO	MN
TO.MN.L04-TO.U.1	TO	MN
TO.MN.L06-TO.U.2	TO	MN
TO.MN.L08-TO.U.5	TO	MN
TO.MN.L09-TO.U.2	TO	MN
TO.MN.L11-TO.U.3	TO	MN
TO.MN.L05-TO.M.5	TO	MN
TO.MN.L08-TO.M.5	TO	MN
TO.PX.BUF.M.1	TO	PX
TO.PX.ALA.M.2	TO	PX
TO.PX.LAP.M.4	TO	PX
TO.PX.WES.M.1	TO	PX
TO.PX.CLA.M.1	TO	PX
TO.PX.DF1.M.1	TO	PX
TO.PX.DF2.M.1	TO	PX
TO.PX.COY.M.1	TO	PX
TO.PX.COY.M.6	TO	PX

Step 7 - Metapopulation locus matching with `sstacks`

Files can be found in the `07_sstacks/` directory.

All samples in the population (or all samples you want to include in the analysis) are matched against the catalog produced in[`cstacks`] (#step-6—cstacks) with `sstacks`, run in script `stacks_SPECIES.sh` and `stacks_SPECIES_additional.sh`. It runs off of the samples based in the output directory *and* the listed samples in `sstacks_samples_SPECIES.txt` and `sstacks_samples_SPECIES_additional.txt` (respectively), so make sure all your files (sample and catalog, etc.) are there and match. `sstacks_samples_SPECIES.txt` takes the form:

```
DS.BA.GA.U.1
DS.BA.GA.U.2
DS.BA.GA.U.3
DS.BA.GA.U.4
DS.BA.GA.U.5
...
```

There should be a new file produced at this step for every sample in the output directory:

- `<samplename>.matches.tsv.gz`

Step 8 - Metapopulation oriented by locus `tsv2bam`

Files can be found in the `08_tsv2bam/` directory.

`tsv2bam` and the proceeding programs in the pipeline use a populations map text file to specify which samples to include in the analysis. As such, a new population map (that differs from `06-cstacks_popmap.txt`) should

be created that includes all samples (including those used to create your catalog) that you want to include in the analysis. This file will include the same samples specified in `sstacks_samples_SPECIES.txt` with a colomn specifying population. Here, this file is `popmap_SPECIES.txt`.

We run `tsv2bam` using the script `tsv2bam_SPECIES.sh`.

This is the step at which it's usually discovered that some samples are bad(don't have any useable matches to the catalog). These samples were excluded from `popmap_SPECIES.txt`. For example we might simply cut out the following rows:

```
DS.MN.L10-DS.M.4    Minneapolis
DS.MN.L01-DS.M.4    Minneapolis
DS.BO.WL2.M.4   Boston
```

The following samples were discarded because they contained less than 300 sample loci matched to catalog loci:

Table S3: Samples discarded at the `tsv2bam` stage of the Stacks pipeline.

Sample	City
CD.BA.RG-1.M.4	Baltimore
CD.BA.RG-1.M.5	Baltimore
CD.BA.DHI.U.2	Baltimore
CD.BA.DHI.U.3	Baltimore
DS.MN.L04-DS.U.2	Minneapolis
DS.BO.WL3.M.4	Boston
DS.BO.WL1.M.4	Boston
DS.BO.LC1.M.2	Boston
DS.BO.HC4.M.4	Boston
EC.BO.R4.U.1	Boston
LS.PX.PAL.M.5	Phoenix
LS.PX.PAL.M.2	Phoenix
LS.PX.PAL.M.1	Phoenix
LS.MN.L01-LS.M.1	Minneapolis
LS.PX.PKW.U.3	Phoenix
LS.PX.ENC.M.2	Phoenix
LS.MN.L01-LS.U.1	Minneapolis
LS.LA.ARR.U.4	Los Angeles
LS.BO.WL2.M.3	Boston
LS.BO.WL2.M.1	Boston
LS.BO.R4.U.4	Boston
LS.BO.R4.U.2	Boston
LS.BO.R4.U.1	Boston
LS.BO.R3.U.2	Boston
LS.BO.R2.U.1	Boston
LS.BO.LC4.M.5	Boston
LS.BO.LC2.M.1	Boston
LS.BO.I4.U.1	Boston
LS.BO.HC4.M.4	Boston
LS.BO.HC4.M.2	Boston
LS.BA.WB.U.5	Baltimore
PA.BA.AA.U.3	Baltimore
PA.BA.AA.U.4	Baltimore
PA.BA.LH-3.M.1	Baltimore
PA.BA.LH-3.M.4	Baltimore

Sample	City
PA.BA.LL-4.M.1	Baltimore
PA.BA.RG-1.M.1	Baltimore
PA.BO.HC2.M.4	Boston
PA.BO.LC2.M.3	Boston
PA.BO.LC2.M.4	Boston
PA.BO.R3.U.3	Boston
PA.PX.ALA.M.2	Phoenix
PA.PX.ALA.M.3	Phoenix
PA.PX.RPP.U.1	Phoenix
PA.PX.RPP.U.2	Phoenix
TO.BA.DHI.U.5	Baltimore
TO.BA.TRC.U.1	Baltimore
TO.BA.TRC.U.2	Baltimore
TO.BA.TRC.U.3	Baltimore
TO.BO.R2.U.2	Boston
TO.BO.R4.U.1	Boston
TO.BO.R4.U.2	Boston

Step 9 - Metapopulation SNP calling with `gstacks`

Files can be found in the `09_gstacks/` directory.

The script `gstacks_SPECIES.sh` also uses the population map specified in Step 8, `popmap_SPECIES.txt`. The `gstacks` program aligns the paired-end reads and assembles the paired-end contigs and then calls SNPs.

Produces the following:

- `catalog.fa.gz` : consensus catalog loci, contains the consensus sequence for each locus as produced by `gstacks` in a standard FASTA file
- `catalog.calls` : per-nucleotide genotypes, contains the output of

Step 10 - Metapopulation summaries with `populations`

Files can be found in the `10_populations/` directory.

The populations program will use the script `species_populations.sh` and the population maps specified in Step 9 (`popmap_SPECIES.txt`) to calculate population-level summary statistics. Specifically, the script iterates through all species and several levels of the parameter `--min-samples-overall`, the minimum percentage of individuals across populations required to process a locus

You will most likely run the populations program multiple times if you are looking at different 'sub-populations'. A new directory should be created and the population program should run out of that directory for each iteration of the population program. Alternativley, you can specify a new directory as the output folder in the script using the command `-O`.

Ultimately, the key parameters we used for running `populations` were as follows:

- `--write-random-snp`: restrict data analysis to one random SNP per locus
- `--min-samples-overall`: 20 - locus must be in 20% of individuals
- `--min-maf`: 0.05 - minimum minor allele frequency required to process a nucleotide site at a locus

Step 11 - Examine Within-city Catalogs and Populations

Files can be found in the `11_city_catalogs/` directory.

For any given species within city, there is likely to be a slightly different set of SNPs compared to the whole metapopulation of five cities. We examined 24 sets of species-city combinations. Within the folder `11_city_catalogs` there are 24 folders:

```
CD_BA_22_09_14
CD_LA_22_09_14
CD_PX_22_09_14
DS_BA_22_09_14
DS_BO_22_09_14
DS_MN_22_09_14
DS_PX_22_09_14
EC_BA_22_09_14
EC_LA_22_09_14
EC_PX_22_09_14
LS_BA_22_09_14
LS_BO_22_09_14
LS_LA_22_09_14
LS_MN_22_09_14
LS_PX_22_09_14
PA_BA_22_09_14
PA_BO_22_09_14
PA_LA_22_09_14
PA_PX_22_09_14
TO_BA_22_09_14
TO_BO_22_09_14
TO_LA_22_09_14
TO_MN_22_09_14
TO_PX_22_09_14
```

All folders contain a script and set of samples used to build the species-city catalog. Because parameter search had already been performed at the metapopulation level, we ran all species-city combinations using `denovo_map.pl`. Input samples came from step 4. For example, input files should look like this:

```
CD.BA.AA.U.1.1.fq.gz
CD.BA.AA.U.1.2.fq.gz
CD.BA.AA.U.1.rem.1.fq.gz
CD.BA.AA.U.1.rem.2.fq.gz
```

A population map is also included within each directory to ensure only appropriate samples (i.e., ok coverage) are included in the catalog.

Step 11b - City Level Population Analysis

The city-level populations program will use the script `11-city_catalogs/11-city_populations.sh`. The goal of this script is to run the Stacks function `populations` on each city-species combination while toggling the `--min-samples-overall` option. Toggling this option allows us to hone in on a reasonable number of SNPs while also getting rid of SNPs with a lot of missing data, therefore creating a more robust population structure snapshot. As described in step 11, each city-species combination with adequate data is represented by a folder. The `11-city_populations.sh` script iterates through folders, within which it runs several `--min-samples-overall` levels and outputs each parameter permutation to its own new directory. This script uses the `11-city_catalogs/11-city_catalog_names.txt` to iterate rather than doing all folders by default. This is convenient if you only want to iterate through a subset of the folders (city-species combinations).

Analysis

Fst - Variance among populations

We used `hierfstat::wc()` to calculate an Fst statistic for each species.

```
source("R/Fst_rbarD.R")
```

```
calc_fst()
```

```
read.csv("output/population_stats-fst.csv")
```

```
##   spp Diploid   Fst
## 1  CD   FALSE 0.322
## 2  DS   FALSE 0.355
## 3  EC    TRUE 0.606
## 4  LS    TRUE 0.536
## 5  PA   FALSE 0.244
## 6  TO   FALSE 0.108
```

rbarD - Clonality of populations

We used `poppr::ia()` to calculate the standardized index of association of loci in the dataset.

From the documentation:

> It has been widely used as a tool to detect clonal reproduction within populations. Populations whose members are undergoing sexual reproduction, whether it be selfing or out-crossing, will produce gametes via meiosis, and thus have a chance to shuffle alleles in the next generation. Populations whose members are undergoing clonal reproduction, however, generally do so via mitosis. This means that the most likely mechanism for a change in genotype is via mutation. The rate of mutation varies from species to species, but it is rarely sufficiently high to approximate a random shuffling of alleles. The index of association is a calculation based on the ratio of the variance of the raw number of differences between individuals and the sum of those variances over each locus. You can also think of it as the observed variance over the expected variance.

When `p.rD` is small (<0.05) and rbarD is (relatively) higher, that is a sign that the population could be more clonal than sexual.

There is a nice description here.

```
calc_rbarD()
```

Table S4: rbarD statistics for species and nested populations in this study.

spp	city	n	Ia	p.Ia	rbarD	p.rD
CD	BA	56	21.2938393	1.000	0.0227603	1.000
CD	LA	48	27.6161960	0.127	0.0152745	0.008
CD	PX	82	26.6954409	0.001	0.0137806	0.001
DS	BA	55	29.6256245	1.000	0.0194295	1.000
DS	BO	50	48.6300407	1.000	0.0299887	1.000
DS	MN	80	14.6413798	1.000	0.0121960	1.000
DS	PX	36	0.1930822	0.169	0.0238413	0.063
EC	BA	41	10.3852195	1.000	0.0147025	1.000
EC	PX	29	6.3919146	1.000	0.0163922	1.000
EC	LA	37	14.2293626	1.000	0.0196589	1.000
LS	BA	19	11.0254296	1.000	0.0241329	1.000
LS	BO	26	19.3364324	0.056	0.1211653	0.076
LS	LA	59	14.9797370	0.722	0.0201264	0.001

spp	city	n	Ia	p.Ia	rbarD	p.rD
LS	MN	28	12.9925894	0.993	0.0262154	0.001
LS	PX	35	4.2572897	1.000	0.0200827	1.000
PA	BA	44	11.1791383	1.000	0.0301568	1.000
PA	BO	58	11.3444510	1.000	0.0379816	0.983
PA	LA	38	10.8013156	1.000	0.0228542	1.000
PA	MN	2	NA	NA	NA	NA
PA	PX	28	25.1886585	1.000	0.0785711	1.000
TO	BA	59	7.8175800	1.000	0.0092886	0.999
TO	BO	58	9.6635713	0.001	0.0102150	0.001
TO	LA	28	10.8244447	0.001	0.0156813	0.001
TO	MN	75	7.9147773	1.000	0.0084373	1.000
TO	PX	16	13.3835215	0.001	0.0489003	0.001

Trim spatial files

From the USGS:

> The U.S. Geological Survey (USGS), in partnership with several federal agencies, has developed and released four National Land Cover Database (NLCD) products over the past two decades: NLCD 1992, 2001, 2006, and 2011. This one is for data from 2016 and describes urban imperviousness.
>
> https://www.mrlc.gov/data/type/urban-imperviousness
>
> NLCD imperviousness products represent urban impervious surfaces as a percentage of developed surface over every 30-meter pixel in the United States. NLCD 2016 updates all previously released versions of impervious products for CONUS (NLCD 2001, NLCD 2006, NLCD 2011) along with a new date of impervious surface for 2016. New for NLCD 2016 is an impervious surface descriptor layer. This descriptor layer identifies types of roads, core urban areas, and energy production sites for each impervious pixel to allow deeper analysis of developed features.

First, we trimmed the large data. This makes a smaller `.rds` file for each city.

```
source("R/trim_NLCD_spatial_data.R")
```

```
create_spatial_rds_files()
```

Make Sampling Maps

Next, we made plots for each city's sampling locations. Note that these only include sites that had viable SNPs.

```
source("R/plot_map_of_samples.R")
```

```
make_all_urban_site_plots()
```

Archetypal Analysis

We performed archetypal analysis following https://github.com/AI-sandbox/archetypal-analysis.

There does not appear to be versioning for this software, but we used the tool by cloning it at this commit point: https://github.com/AI-sandbox/archetypal-analysis/commit/7ae1c25c41bbe97ac2ff2b62c4cbe1814d0dbc27 .

The software does something wonky and loads an old version of pandas (1.1.5) but a newer version is required (>=1.2.0). This has to be updated in the `requirements.txt` file and the `setup.py` file in the archetypal-analysis repo that is cloned.

```
numpy==1.19.5
pandas==1.2.0 <<<<< Fix this line when cloned
pandas-plink==2.2.4
```

Update pandas if needed.

```
pip install pandas -U
```

```
source("R/archetype_analysis.R")
```

We ran the archetypal analysis algorithm looking for k = the number of cities in which the species was ID'd as well as k * 2.

```
run_archetype_analysis(spp = "CD", k_vals = c(3, 6))  # 3 populations
run_archetype_analysis(spp = "DS", k_vals = c(4, 8))  # 4 populations
run_archetype_analysis(spp = "EC", k_vals = c(3, 6))  # 3 populations
run_archetype_analysis(spp = "LS", k_vals = c(5, 10))  # 5 populations
run_archetype_analysis(spp = "PA", k_vals = c(5, 10))  # 5 populations
run_archetype_analysis(spp = "TO", k_vals = c(5, 10))  # 5 populations
```

Visualizing Archetypes

Before proceeding, we wanted to extract urban cover from the NLCD data and add it to the site data.

```
source("R/extract_envt_data.R")
```

```
write_urban_cover_data()
```

Plot the archetypes in one big plot.

```
source("R/plot_archetypes.R")
```

```
make_archetype_multi_plot()
```

Isolation by State

```
source("R/plot_isolation_by_state.R")
```

```
plot_dendrograms()
```

Appendix

SessionInfo()

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Big Sur 11.7
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] SNPRelate_1.32.0    gdsfmt_1.34.0       ggrepel_0.9.2
##  [4] ggtreeExtra_1.6.1   ggtree_3.4.4        treeio_1.20.2
##  [7] tidytree_0.4.1      ape_5.6-2           treedataverse_0.0.1
## [10] ggh4x_0.2.3         forcats_0.5.2       here_1.0.1
## [13] cowplot_1.1.1       viridis_0.6.2       viridisLite_0.4.1
## [16] readr_2.1.3         stringr_1.4.1       raster_3.6-11
## [19] sp_1.5-1            poppr_2.9.3         hierfstat_0.5-11
## [22] adegenet_2.1.8      ade4_1.7-20         dplyr_1.0.10
## [25] magrittr_2.0.3      tidyr_1.2.1         ggplot2_3.4.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-160       bit64_4.0.5        rprojroot_2.0.3    tools_4.2.2
##  [5] polysat_1.7-7      utf8_1.2.2         R6_2.5.1           vegan_2.6-4
##  [9] lazyeval_0.2.2     DBI_1.1.3          mgcv_1.8-41        colorspace_2.0-3
## [13] permute_0.9-7      withr_2.5.0        tidyselect_1.2.0   gridExtra_2.3
## [17] bit_4.0.5          compiler_4.2.2     textshaping_0.3.6  cli_3.4.1
## [21] formatR_1.12       labeling_0.4.2     scales_1.2.1       yulab.utils_0.0.5
## [25] systemfonts_1.0.4  digest_0.6.30      rmarkdown_2.18     pkgconfig_2.0.3
## [29] htmltools_0.5.3    pegas_1.1          fastmap_1.1.0      highr_0.9
## [33] rlang_1.0.6        rstudioapi_0.14    shiny_1.7.3        gridGraphics_0.5-1
## [37] farver_2.1.1       generics_0.1.3     jsonlite_1.8.3     vroom_1.6.0
## [41] ggplotify_0.1.0    patchwork_1.1.2    Matrix_1.5-3       Rcpp_1.0.9
## [45] munsell_0.5.0      fansi_1.0.3        ggnewscale_0.4.8   lifecycle_1.0.3
## [49] terra_1.6-41       stringi_1.7.8      yaml_2.3.6         MASS_7.3-58.1
## [53] plyr_1.8.8         grid_4.2.2         parallel_4.2.2     promises_1.2.0.1
## [57] crayon_1.5.2       lattice_0.20-45    splines_4.2.2      hms_1.1.2
## [61] knitr_1.41         pillar_1.8.1       igraph_1.3.5       boot_1.3-28.1
## [65] seqinr_4.2-23      reshape2_1.4.4     codetools_0.2-18   glue_1.6.2
## [69] evaluate_0.18      ggfun_0.0.9        vctrs_0.5.1        tzdb_0.3.0
## [73] httpuv_1.6.6       gtable_0.3.1       purrr_0.3.5        assertthat_0.2.1
## [77] xfun_0.35          mime_0.12          xtable_1.8-4       later_1.3.0
## [81] ragg_1.2.4         tibble_3.1.8       aplot_0.1.9        cluster_2.1.4
## [85] ellipsis_0.3.2
```

File Organization

All data files for the Macrosystems project are permanently stored under Meghan Avolio's group resources in the Johns Hopkins University Rockfish computing cluster. Files are stored under the 'data' directory under the following subdirectories:

- `01-raw_data`: This folder contains the raw, unprocessed data files that were obtained directly from the sequencing server. There are eight `fastq.gz` files per sublibrary that correspond to the four sequencing lanes for each read direction.
- `02-concatenated_data`: This folder contains the concatenated, unprocessed files for each sublibrary (i.e., the files containing the sequences for each lane were combined to create one file per read direction).
- `03-pcr_filtered_data`: Here, you will find the resulting data files from the `clone_filter` program, where pcr replicates/clones have been removed from the raw sequences. There are two `fq.gz` files per sublibrary.
- `04-process_radtags`: This folder contains various subdirectories that correspond to the

`process_radtags` program that demultiplexes and cleans the data. The `demux_txt_files` folder contains the .txt files used to identify barcodes and separate out the individual samples from each sublibrary. The resulting data files from the `process-radtags` program are separated by individual and can be found in the relevant species folder(i.e., CD, DS, EC, LS, PA, TE, TO). Each individual sample has four data files; `sampleID.1.fq.gz` and `sampleID.2.fq.gz` correspond to the forward and reverse reads for each sample and `sampleID.rem,1/2.fq.gz` contain the remainder reads that were cleaned and removed from the data sequence.

- `05-ustacks-denovo_data`: This folder contains species subdirectories that store the resulting data files from the `ustacks` program for each individual. There are three files per individual; `sampleID.allelles.tsv.gz`, `sampleID.snps.tsv.gz`, and, `sampleID.tags.tsv.gz`. These files should be permanently stored here and copied to a new directory for any new catalogs and/or when a group of samples are being aligned to a new catalog.
- `catalogs_by_city`: For any given species within city, there is likely to be a slightly different set of SNPs compared to the whole metapopulation of five cities. We examined 24 sets of species-city combinations. These catalogs are permanently stored here.
- `catalogs_by_species`: Metapopulation catalogs are stored within this folder for each species. The metapopulation catalog was created using samples from all populations to create a national catalog.

  Some notes about catalog directories:

  - Catalogs contain three files; `catalog.alleles.tsv.gz`, `catalog.snps.tsv.gz`, and `catalog.tafs.tsv.gz`. If you would like to use the catalog on a new project, you will need to copy all three files to a new project folder.
  - You can determine which individuals were used to create the catalog by looking at the `cstacks_popmap.txt` found within each folder. Specifically for the metapopulation catalogs, this information is also found in the cstacks-metapop-catalog_samples-included.csv
  - You can determine which individuals were subsequently aligned to the catalog and used in the subsequent stacks analysis by looking at the `popmap*.txt` found within each folder.
  - Each folder also contains the relevant ustacks and stacks pipeline scripts and output files (i.e., from `cstacks`, `gstacks`, `stacks`, `tsv2bam`, and `populations`),

Aspera Transfer File Names

See data/aspera_transfer_file_names.csv. Preview:

```
readLines("data/aspera_transfer_file_names.csv", 10)
```

```
##  [1] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L001_R1_001.fastq.gz"
##  [2] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L001_R2_001.fastq.gz"
##  [3] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L002_R1_001.fastq.gz"
##  [4] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L002_R2_001.fastq.gz"
##  [5] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L003_R1_001.fastq.gz"
##  [6] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L003_R2_001.fastq.gz"
##  [7] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L004_R1_001.fastq.gz"
##  [8] "/Hoffman_macrosystems/AMH_macro_1_1_12px_S1_L004_R2_001.fastq.gz"
##  [9] "/Hoffman_macrosystems/AMH_macro_1_10_8px_S10_L001_R1_001.fastq.gz"
## [10] "/Hoffman_macrosystems/AMH_macro_1_10_8px_S10_L001_R2_001.fastq.gz"
```

`clone_filter` File Names

See data/clone_filter_file_names.csv. Preview:

```
readLines("data/clone_filter_file_names.csv", 10)
```

```
##  [1] "AMH_macro_1_1_12px_S1"  "AMH_macro_1_10_8px_S10" "AMH_macro_1_11_8px_S11"
```

```
##  [4] "AMH_macro_1_12_8px_S12" "AMH_macro_1_13_8px_S13" "AMH_macro_1_14_8px_S14"
##  [7] "AMH_macro_1_2_12px_S2"  "AMH_macro_1_3_12px_S3"  "AMH_macro_1_4_12px_S4"
## [10] "AMH_macro_1_5_8px_S5"
```