Processing Illumina Pop Gen Data - update for 2019 CCN, KLB

Preface

This is a compilation of notes to accompany PERL and R scripts and Zach's C programs, and is intended as a rough guide to the steps required for processing Illumina data generated using the restriction-fragment based protocol developed in the Buerkle lab at U Wyoming. This is largely the product of instructions from Zach as captured by Lauren Lucas, Kate Bell and Laura Alberici Da Barbiano, with minor additions from Chris. Kate and Chris have organized and expanded much of this. Disclaimer: we deny everything.

More disclaimers: While we are working to clean things up, you should note that some annotations are not complete, or that scripts have changed leaving annotations somewhat inaccurate. This includes usage statements (we're working on it.) There are also several places where we are adding new things, meaning that this document is evolving. Embedded in these notes are some simple UNIX command line things, but this is uneven and might be frustrating if you are not already familiar with UNIX.

Update began Oct 4, 2014

This is an evolving doc. Please add notes, the only caveat being that you start any notes with your initials (e.g. "CCN: this part doesn't work."). We can formalize changes periodically. Or please send notes for additions or clarifications to Chris - ccnice@txstate.edu.

A final note about directory organization: comments in this vein are embedded and are designed to keep a homogenous directory structure across investigators and projects on the computers at TSU.

Overview

This is not a "pipeline" in the sense that this guide or the processing of sequence data is linear. There are multiple paths for various analyses and the appropriate path(s) for analyses will depend on the project and the questions (like any other population and evolutionary genetics project) (Fig. 1). There are, however, some commonalities. Every project will probably require the first several steps at minimum: steps 1-7 in Fig. 1 which include parsing, *de novo* assembly, reference-based assembly, and variant calling. Most investigators will also be very interested in quantifying various aspects of sequence coverage (step 8). From these initial steps, paths can diverge, or even loop back to earlier steps. In our schematic, we have divided things broadly into three categories in these later steps.

The blue boxes in Fig. 1 represent hierarchical Bayesian models that estimate posterior genotype probabilities for each individual at each locus. These approaches include the *Entropy* model (**??**) (which is very similar to *Structure* (**?**)) and produces assignment probabilities as well as genotype probabilities, and the *Popmod* model which assumes population structure and uses an allele frequency prior for each population (similar to the model described by **?**). In these methods, the genotype likelihoods calculated in variant calling are used directly as

input and updated for posterior probabilities of genotypes (see **?** for a good example). The genotype probabilities from these models can be used to get allele frequencies and be directly employed for illustrating patterns of variation (i.e. PCA, calculation of F_{ST}) or other kinds of analyses such as GWAS (**?**).

The dark blue boxes include various analyses that require SNP count files (snpcnt) as input. This is mostly an historical artifact (these methods were created before the methods outlined above). Here, the actual counts of alleles for each individual and locus are used in the calculation of posterior genotype probabilities. The *Afreq* model is very similar to the *Popmod* model, except for the form of the input data (**?**). Investigations of introgression using the *BGC* model (**?**) also require snpcnt files as input.

Lastly, it is also possible to work directly with genotype likelihoods from variant calling; the orange boxes in Fig. 1. This might be desirable when samples clearly violate Hardy-Weinberg assumptions, such as in situations with inbreeding or clonality, or when individuals are not randomly sampled from populations (e.g. mothers and offspring for paternity analysis or specific experimental situations). Using genotype likelihoods probably requires higher coverage than other analyses (above). This is the least explored area of analysis for us.

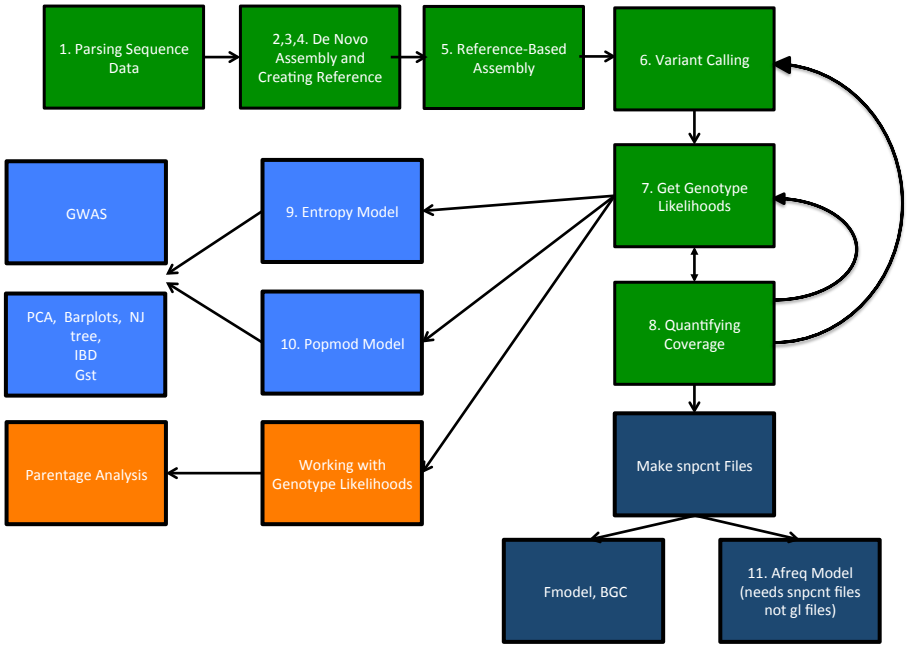

Figure 1: An overview of steps and paths for analysis of sequence data.

As a final note, please recognize that there are numerous alternatives for data analysis that are not included here. Genotyping-by-Sequencing (GBS) data forces us to think about uncertainty more than we used to with more conventional genetics data. Stochasticity arising from library preparation and the sequencing process means that there is substantial variation in coverage depth per locus and per individual in GBS data sets which means that there can be uncertainty about the genotype for an individual at a particular gene region. The

approaches adopted here are grounded in the philosophy that genotype uncertainty should be confronted. The central idea is that we are sampling the underlying genotype with GBS sequence reads, with all the attendant issues concerning sampling. Thought of this way, it makes some sense to treat the problem of genotyping from a modeling perspective like any other inference problem. This is the foundation for models like *Entropy*.

This is not the only way to deal with GBS data. A popular approach to deal with this uncertainty is to filter the data in a way to minimize it. This means throwing away data below a coverage threshold and keeping only those markers for which there are many sequence reads. This is essentially the way that the STACKS pipeline and others work (add refs). The advantage of this approach is that one might not worry about uncertainty if data have been filtered to only include loci with very high coverage across individuals and loci (or at least that is the idea). Computational time and overall complexity of analyses are reduced. The use of Bayesian inference does require more computational time than these other approaches to GBS data analysis. It also requires some familiarity with Bayesian methods and might not be applicable to all situations. On the other hand, another advantage of modeling genotype uncertainty is that you can potentially take advantage of lower coverage data, meaning that these methods that account for variable coverage allow researchers to use more of their data and thus maximize their research dollars (**?**).

What follows is a rough guide to Fig. 1. Throughout, commands to be typed after the prompt are in blue, with specific files, where your names might vary, are in red.

1 Assemble raw reads to phiX genome

The Point and Background: this is really an optional step, but one that can greatly reduce the time required for parsing (the next step). With the advent of sequencing on the HiSeq 4000 at the UT GSAF, the quantity of phiX used during sequencing increased substantially. phiX is a control sequence (aka "spike") used for Illumina sequencing. It is sequence from a small (5386bp) bacteriophage genome - apparently the first genome to be sequenced (and Fred Sanger did it). It is used for at least two purposes: it provides heterogeneity among clusters so that the camera can detect individual clusters as they are sequenced, and it provides a control for estimating sequencing error. From communicating with the GSAF, it seems that GBS libraries require more phiX than other sequencing projects and the 4000 requires more phiX than was used for 2500 sequencing. Given the increase in phiX sequences in raw reads, this optional first step uses the bowtie assembler to assemble raw reads to the phiX genome. The unaligned sequences (i.e. not phiX sequences) can then be parsed in the next step. Note that this is a time saver only - bowtie is a fast assembler, but the parsing script will also remove phiX sequences, albeit more slowly.

1. Create directory with project name (when in /Volumes/data: mkdir projectName)

2. Create directories within project directory (when in /Volumes/data/neophasia: mkdir barcodeKeys and do the same for rawReads and parsedReads)

3. Download/transfer/copy the file containing the rawreads.

Example:

scp rawreads.fastq.gz user@melissa.science.txstate.edu:/Volumes/data/projectName/rawReads/

You will then be asked for your password

4. Uncompress your rawReads:

gunzip rawreads.fastq.gz

5. copy the 6 files that comprise the indexed phiX genome to your rawReads directory. These files are:

e_phiX.1.ebwt
e_phiX.2.ebwt
e_phiX.3.ebwt
e_phiX.4.ebwt
e_phiX.rev.1.ebwt
e_phiX.rev.2.ebwt

These files are stored on melissa at: /Volumes/storage_m/phiX
On iris, they are at: /Volumes/data_i/scripts/phiX
On anna, they are at: /Volumes/programs/phiX

from your rawReads directory:

cp /Volumes/storage_m/phiX/e_phix* ./

6. Use the following command line to use bowtie to assemble your raw reads to the phiX genome:

bowtie -v 3 -a −−al aligned_phiX.txt −−un unaligned_NOTphiX.txt −−suppress 1,2,3,4,5,6,7,8 e_phiX rawreads.fastq hitFile.txt &

The -v option is the allowed mismatches, in this case 3. The "&" at the end puts the job in the background so that if you are disconnected, the job will continue. You can always use top to see if the job is still running.

[NOTE: if working remotely through TSU's VPN, you should exit after putting jobs in the background and reconnect. For some reason the & doesn't always work with the VPN - e.g. if you get disconnected, bowtie stops. Start this job and immediately exit by typing "exit" on the command line, or at least make sure you exit before being disconnected, then the job will continue. I have no idea why this happens.]

When the job is finished, move (using mv) the unaligned file to change its name. For example:

mv unaligned_NOTphiX.txt reads.fastq

2 PARSING

The Point: This is removing barcodes and cut sites from raw reads and attaching ind. IDs. We also check for short reads or reads with the Mse1 adapter seq.'s. See our protocol for library prep. This works with our set of 768 barcodes. There is also a subroutine that fixes mutations in the barcodes.

1. Transfer/copy over barcodekeys file.

Example:

scp barcodeKey.csv user@greenhouse.science.txstate.edu:/Volumes/data/projectName/barcodeKeys/

You will then be asked for your password

Your barcodekey file should look like this:

barcode_id,barcode,id
98,GATTAGGACCAATTC,uce_001
57,GCGCGGATACAATTC,uce_002
63,AGAGTCTGCCCAATTC,uce_003
83,GGATCCTTCAATTC,uce_004
59,CGTCGGAGACCAATTC,uce_005
5,AAGCCATAACCAATTC,uce_006
...

The first column is irrelevant (but needs to be there). The barcode sequences need to be uppercase letters. If that is not true, use the script "barcode_UPPERCASE.pl" to fix that:

perl barcodeUPPERCASE.pl standardBarcodeKey.csv

This will produce "UC_standardBarcodeKey.csv".

The barcode key file needs to be .csv and have unix line endings, the raw reads file needs to be a fastq file.

4. Once files have transferred, go to /Volumes/data/projectName/parsedreads - now the perl script to parse the reads can be run.

Type perl then the pathway to the perl script, then the pathway to the barcodes file and then the pathway to the raw reads.

Example:
perl /Volumes/data/scripts/parse_barcodes768.pl /Volumes/data/projectName/barcodekeys/barcodeKey.
/Volumes/data/projectName/rawreads/rawreads.fastq &

Our current parsing script is "parse_barcodes768.pl".

5. To run this job in the background add & to the end of the command line. The advantage of running this script in the background is that if your connection is broken, the program will continue running.

6. The script prints to screen the number of reads for each MID. Running the parsing script will produce two files, appropriately named: parsed_rarwreads.fastq and miderrors_rarwreads.fastq

7. If you want to capture the screen output, add ">parseOUT.txt" to the end of your command line

8. Alternatively, to get some information about your parsed data, copy the perl script "getIndCovParsed.pl" to your parsedreads directory so you can edit it.

To copy a file, when in the directory that you want to copy it to, use this command:

cp /Volumes/data/scripts/getIndCovParsed.pl ./

9. You will need to edit this perl script, changing the regular expressions to recognize your ID numbers. Example for IDs that look like this: abc_123. Change regular expression in script to [a-z]+_[0-9]+ This means that there are lower case letters, + means that there is one or more letters, then underscore, then [0-9] means that there are numbers and + means there is one or more numbers.

10. Then run the perl script on the parsed reads file, it will print to the screen the numbers of reads for each individual as well as the total number of reads and also creates a file called "coverage.txt" which contains the same information.

11. To count the total number of reads in the parsed file, you can use the grep command to count the number of "@" signs in the parsed read file:

grep @ parsed_rawreads.fastq | wc

3 *de novo* ASSEMBLY

The Point (and some notes): the following is a strategy to construct scaffolds for reference-based assembly for non-model systems that lack a genome. If you have a genome for your project, skip this section. You might also consider using a reference genome from a reasonably closely related organism.

The strategy here is adapted from Jon Puritz' dDocent wrapper (http://ddocent.com). Basically, the following are the command lines for *de novo* assembly of single-end read data extracted the dDocent wrapper. (Please recognize that you could also simply use the full dDocent wrapper for processing your data and skip this analysis guide.) This whole *de novo* assembly does not take very long. The longest step might be zipping the fastqs. However, many of the steps, especially step doing the cd-hit assembly (step 5), will take ALL of the RAM on your computer. Check top before initiating.

The scripts that do most of the work (steps 4 and 5 in this section) are set up to create a reference assembly that consists of sequences from reads that had at least 4 copies, were found in at least 4 individuals and have less than 80% homology among them. These values can all be adjusted in the scripts. These values are partly recommendations following from dDocent and supported by empirical observations. Keep in mind the overall goal here which is to cluster sequences and filter low frequency sequences and those that might be paralogs. We then use the consensus reads from these sequences as our "reference" for reference-based assembly.

1. Make individual .fastq files from your parsed reads. Make a list of all individuals from your barcodeKey (the uppercase version):

perl /Volumes/data m/scripts/IDfileMaker2.pl UC_barcodeKey.csv

This produces 'IDfile_UC_barcodeKey.csv' which looks like this:

abc_001
abc_002
acc_001
acc_002
bhe_001

cuc_001
cob_001 . . .

Next, split the parsed reads into individual .fastq files:

perl /Volumes/data m/scripts/splitFastqMF.pl IDfile_UC_barcodeKey.csv parsedReads.fastq &

[NOTE: Macintosh computers will only let you have 250 files open at one time, meaning that if you have >250 ind.s, this step will fail on a Mac. Solutions include splitting the IDfile into smaller lists or using a Linux machine.]

[NOTE: If your project includes reads from more than one lane, you will need to concatenate them before you run the splitFastq script. For example:

cat parsedReads_lane1.fastq parsedReads_lane2.fastq >combined_parsedReads.fastq]

[OPTIONAL STEP(s): For organizational simplicity, you could make a new dir for the individual .fastq files. We recommend also making another dir and copying the .fastq files. The following steps include compressing the .fastq files. To use them again, you would need to uncompress them. If you make a copy, you can avoid repeatedly compressing and uncompressing these files.]

2. make a list of fastq files:
ls *.fastq >namelist

Use a one liner to remove '.fastq' from namelist:
sed -i" -e 's/.fastq//g' namelist Note: use less to look at namelist and make sure the '.fastq' part of the file names is gone. Note also that the " is two single quotes, not one double quote

3. Compress fastq files (this takes some time - you could think about ways to speed this up):
gzip *fastq
Alternatively, you could use the script "zip_forks.pl".

4. Make a list of all the unique sequence reads for each individual and count them. For this, we will use a bash script. This script does the counting of unique sequences, makes a directory called "zippedFastqs" and moves all of your zipped fastq files into that directory, then it combines all the lists of unique sequences into one file, counts how many sequences are shared across various numbers of individuals, prints that to screen, and restricts sequences to those that have at least 4 copies. A number of files are created in the process that are worth investigating. [QUESTION TO USER: do we need to add explanations for all the files?] To use, copy the script to your directory and then run it as follows:
./ddocent.sh

5. In this step, we will restrict the set of seqs to those that are in at least 4 ind.s, meaning that the final set of sequences will have at least 4 copies and be shred by at least 4 individuals. We also convert these sequences to fasta format and perform a clustering/assembly at allowing up to 80% homology as a means of screening out potentially paralogous sequences. Copy another bash script into your working directory and execute as follows:

./ddocentPart2.sh

6. remove any tapeworm sequences:

perl tapewormRemover.pl reference.fasta
You will get a message that looks like this:
"We removed 26 tapeworms and kept 145299 good scaffolds"

4 REFERENCE-BASED ASSEMBLY USING BWA

The Point: now we will assemble all of the parsed reads to the scaffolds coming out of the *de novo* assembly. We use BWA (Burrows Wheeler Aligner). You will need the list of individuals you made at step 3-1 (i.e. 'IDfile_UC_barcodeKey.csv') and the individual .fastq files.

1. Index the reference .fasta file with BWA:

bwa index -p projectName_ref -a is reference.fasta

this produces:

projectName_ref.sa
projectName_ref.amb
projectName_ref.ann
projectName_ref.pac
projectName_ref.bwt

2. Do the assembly using the wrapper script "runbwa.pl". Copy this to your "assembly" directory and modify the last two lines to match your project. Specifically, you need to provide the name of the indexed reference files that you made in the previous step. Also, take a look at the command line options and modify. The -n option, which sets the "maximum edit distance" - presumably how much mismatch is allowed in the assembly - is perhaps the most important assembly parameter. In most circumstances, this can be set from 2 to 6, however, some experimenting with this parameter, and comparisons of different assemblies, is warranted.

perl runbwa.pl *fastq

This creates .sam and .sai files for every individual

3. Convert .sam files to .bam files

perl sam2bam.pl *.sam

[Note: you will see a bunch of error messages - don't worry about them, everything is running fine. These errors come from how this script is forking child processes and DO NOT impact how your .sam files are being compressed. I am working on tuning this up.]

5 Variant Calling and Filtering

1. Use bcftools to call variant SNP sites. 2 steps:

   bcftools mpileup -d 8000 -o out1.bcf -O b -I -f good_reference.fasta aln*sorted.bam &

   bcftools call -c -V indels -v -p 0.05 -P 0.001 -o variants.vcf out1.bcf

See the bcftools documentation and man pages for detailed info on these options.
[NOTE: for those of you familiar with older versions of bcftools, the -d parameter is no longer the proportion of individuals that must have data for a SNP to be called. Here -d is a maximum sequence depth allowed per individual and really is a limit to minimize using too much memory. We now do the equivalent of the old -d filtering in steps 2-4 in this section, below. You might also consider adding the -Q option which sets the minimum base quality. The default is 13, which seems quite low. Phred scores, Q, are quality scores embedded with your Illumina sequences and describe the probability, P, that the base in question is an error (i.e. sequencing error).

$P = 10^{(-Q/10)}$, so Q=13 means 5% chance. Q=20 translates to 1% (or 1/100), and 30 means 0.001. Q = 20 seems a good filter, Q= 30 is better. Think about it.]

[FURTHER NOTE: at this point, you have the option of using the clustering algorithm in the *Entropy* software (described in Gompert et al. 2014 (*Molecular Ecology* 23:4555-4573)), or using our older approach using the *Afreq* model (described in Gompert et al. 2012 (*Evolution* 66:2167-2181)) and the F-model. Or, you could think about using genotype likelihoods from variant-calling and skip the hierarchical models. Here, we continue with the path to *Entropy*. For the older path and other options, jump to Step 9 or 10.]

2. First round of filtering. Perform filtering of your SNPs for a number of parameters using the information in the variants.vcf file. There is some redundancy in these filtering steps, but I like it. Copy the script "vcfFilter_CCN_1.9.pl" to your working directory. You will need to modify the filtering criteria before using. Specifically, use a text editor (e.g. nano, emacs, vim) to edit the stringency variables on lines 18-27 in this script. Here is an example:

```
#### stringency variables, edits as desired
my $minCoverage = 350; # minimum number of sequences; DP
my $notFixed = 1.0; # removes loci fixed for alt; AF
my $bqrs = 3; # maximum absolute value of the base quality rank sum test; BaseQRankSum B
my $mqrs = 2.5; # maximum absolute value of the mapping quality rank sum test; MQRankSum
my $rprs = 2; # maximum absolute value of the read position rank sum test; ReadPosRankSu
my $afreqMin = 0.05; # minimum allele freq; AF1  #
my $afreqMax = 0.95; # maximum allele freq; AF1  #
my $mq = 30; # minimum mapping quality; MQ
my $miss = 35; # maximum number of individuals with no data
################################
```

minCoverage can be set to create an average coverage threshold. For example for 175 inds * 2 = 350 = mean of 2x coverage
notFixed simply screens out the weird loci that have no variants (they shouldn't be in there anyway)
bqrs sets the maximum limit for the difference in base quality between the reference allele and the alternative allele
mqrs same thing for mapping quality
rprs same thing for read position
afreqMin and **afreqMax** set the minor allele frequency threshold. If you want to stratify "common" and "rare" SNPs (see Gompert et al. 2014) you could run this script twice with different values here.
mq sets the minimum mapping quality score
miss sets the maximum number of individuals that can have no data. This is essentially the old -d but thinking about missing data and using numbers of inds rather than proportion. For example, the old -d set at 0.8 required at least 80 percent of individuals to have data before a SNP is called. The equivalent here for 170 individuals would be 34.

Once you have these filters set, run the script:

perl vcfFilter_CCN_1.9.pl variants.vcf

This produces a file called "filtered_firstRound_variants.vcf".

3. Next, inspect the coverage depth for the remaining loci. The strategy is to filter those loci which have extremely high sequencing depth because they are likely to be assemblies of paralogous loci. We use the "depthCollector.pl" script to make a list of all loci and their depth. Then we read that into R to calculate some statistics to use in the second round of filtering. Copy "depthCollector.pl" to your directory. Then:

perl depthCollector_1.9.pl filtered_firstRound_variants.vcf

Then, read the resulting file into R and calculate some stats:

```
> dat<-read.table("depth_filtered_firstRound_variants.vcf",header=F)
> dim(dat)
[1] 45350     3
> dat[1:5,]
  V1 V2      V3
1  2 40  253355
2  2 48  253361
3  5 62 1386064
4 11 14    1491
5 11 48    1491
> summary(dat$V3)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    350     607     955    2095    1588 1386000
> sd(dat$V3)
[1] 18090.79
> mean(dat$V3)+2*sd(dat$V3)
[1] 38276.33
> length(which(dat$V3 > 38277))
[1] 132
```

3. Second round of filtering (much like the first, except we also kick out loci with super high coverage depth). Copy "vcfFilter_CCN_1.9_more.pl" to your directory and modify as you did for the first round. Notice that the same filters are here in this script, but there is an additional one: **maxCoverage** sets the maximum allowable sequence depth. Edit this script:

```
#### stringency variables, edits as desired
my $minCoverage = 350; # minimum number of sequences; DP
my $maxCoverage =  38277; # max number of sequences; DP ; mean + 2sd; CCN addition - cal
my $notFixed = 1.0; # removes loci fixed for alt; AF
my $bqrs = 3; # maximum absolute value of the base quality rank sum test; BaseQRankSum B
my $mqrs = 2.5; # maximum absolute value of the mapping quality rank sum test; MQRankSum
my $rprs = 2; # maximum absolute value of the read position rank sum test; ReadPosRankSu
my $afreqMin = 0.05; # minimum allele freq; AF1  #
my $afreqMax = 0.95; # maximum allele freq; AF1  #
my $mq = 30; # minimum mapping quality; MQ
my $miss = 35; # maximum number of individuals with no data
######################
```

In the above example, I set the **maxCoverage** limit to be 2 standand deviations above the mean sequence depth. Feel free to set this to whatever you think is best. (You should probably also make sure that all the rest of the filters are the same as they were in the first round of filtering.) Then run the script:

perl vcfFilter_CCN_1.9_more.pl filtered_firstRound_variants.vcf

This produces a file called "filtered_secondRound_filtered_firstRound_variants.vcf".

Now the file names are getting annoying, let's change the name of this final filtered .vcf file:

mv filtered_secondRound_filtered_firstRound_variants.vcf doubleFiltered_variants.vcf

6 GET GENOTYPE LIKELIHOODS (AND SOME MORE FILTERING)

1. Pull out the genotype likelihoods from the variants.vcf file:

perl vcf2mpgl_CCN_1.9.pl doubleFiltered_variants.vcf

Prints to screen: "Number of loci: 153308; number of individuals 130" and makes file "doubleFiltered_variants50.mpgl". The suffix "mpgl" stands for "multiple population genotype likelihoods". Note: this script also produces "indiv_ids.txt" and the header which are needed later.

2. Generate allele frequency estimates:

perl vcf2af.pl doubleFiltered_variants50.vcf

This script retrieves the AF1 value from the vcf file, which is the maximum likelihood estimate of the alternative allele frequency assuming HWE. Produces "doubleFiltered_variants50.af" which has three columns: scaffold number, scaffold number:position, alternative allele frequency:

278 278:27 0.0197
695 695:81 0.04607
1126 1126:83 0.03265
2842 2842:42 0.04116
. . .

3. Next we select one variant site per scaffold/contig producing a set of SNPs that are all on different scaffolds and therefore less likely to be in LD. For this, we use read the .af file into R and randomly pick one SNP per contig and put those in a list:

```
> dat<-read.table("doubleFIltered_variants.af",header=F)
> dim(dat)
[1] 45218     3
> contigs<-unique(dat$V1)
> length(contigs)
[1] 8955
> loci<-matrix(NA,nrow=length(contigs),ncol=1)
> for(i in 1:length(contigs)){
+         sites <- dat[dat$V1 == contigs[i],]
+         rand <- sample(1:length(sites$V1), 1)
+         tmp <- as.character(sites$V2[rand])
+         loci[i,1] <- tmp
+ }
> length(loci)
[1] 8955
> write.table(loci,file="loci.txt", quote=FALSE, row.names=FALSE, col.names=FALSE)
```

4. Get the genotype likelihoods for the the specific loci that are in your list:

perl sort_mpgl_loci.pl loci.txt doubleFIltered_variants.mpgl

Produces "loci.txtdoubleFIltered_variants.mpgl"

5. Transform the genotype likelihood files into a genotype matrix with the mean genotype (point estimate of genotype) for each individual for each variant site (SNP).

perl gl2genest.pl loci.txtdoubleFIltered_variants.mpgl

Produces gl_loci.txtdoubleFIltered_variants.mpgl

This converts the phred score - scaled likelihoods into mean genotypes for each individual for each variable site. The results is a number between 0 and 2, with 0 being homozygous for the reference allele (referring back to bcftools variant calling), 1 (heterozygous) or 2 (homozygous for alternative allele). These mean genotypes are continuous scores that reflect the coverage depth and the sequence quality. Thus, a mean genotype of 0.3224 reflects uncertainty about whether the individual is homozygous or heterozygous, though there is higher likelihood (more data supporting) that the individual is homozygous. Similarly, a mean genotype of 1.9999 reflects relatively little uncertainty about the homozygous genotype of this individual.

7 Quantify Coverage

The point: Quantifying coverage will allow us to look at both the average number of reads per locus and the number of reads per locus for each individual. Deciding how to filter your data will depend on the downstream analyses you plan to use. For example, if working with genotype likelihoods it may be best to use only loci that have very high coverage so that there is less uncertainty in your data. [NOTE: CCN it needs to be clear that quantifying coverage is a flexible task - i.e can be done at different points]

1. Using the mpgl file created above, use "coverage_calc_bam_CCN.pl" to get a list of the number of reads for each individual for each loci. To run this script you will need a text file with a list of individual id's and you need to provide the pathway to the .bam files for your project. (Notice that this script does not work with the "gl_variants.mpgl" you just made, but rather with the "variants.mpgl" file from step 6-1.)

perl coverage_calc_bam_CCN.pl loci.txtdoubleFIltered_variants.mpgl indiv_ids.txt /path_to_bam_files/

2. Now R can be used to summarize coverage, read through the R script "coverage_updown.R" for more details. You can either summarize data based on average number of reads per individual, and remove individuals who do not meet your threshold for coverage, or remove loci that do not have enough coverage. If you decide to remove certain individuals, you may wish to go back to variant calling and re-do steps 5 and 6 with those individuals excluded, or even go back and re-do the ref-based assembly (Step 4) with those ind.s excluded.

3. To remove particular loci, print out a list of the loci you want to keep from R (see script "coverage_updown.R" for instructions). You can then use this list to create a new mpgl file which contains only the loci that have high enough coverage. To do this you will use the

script called "select_loci_mpgl_CCN.pl", you will need to include the list of high coverage loci that you made in R and your original mpgl file.

perl select_loci_mpgl_CCN.pl High_Cov_Loci.txt variants.mpgl

This will produce a file called "variants_focalloci.mpgl" which can be used in further downstream analyses.

8 PREPARING FILES FOR *Entropy*

1. Make another ID file for these steps. This is a list of all ind. IDs with "aln" added to each. These names appear all on one line, then there is a line of the "population" identifiers (i.e. the "abc" part of abc_123) all on one line. For example:

alnlfu_001 alnltr_004 alngme_001 . . .
lfu ltr gme . . .

Use the script "entropy_IDfileMaker2.pl":

perl entropy_IDfileMaker2.pl IDfile_barcodeKey_sam.txt

produces "entropy_IDfile_barcodeKey_sam.txt" and "header_ids.txt".

2. Concatenate the "header_ids.txt" file (from step 8-1 above) with the genotype likelihood matrix file. [NOTE: it is worth checking that the ind.s listed in "header_ids.txt" are in alpha/numeric order because that is the order of ind.s in your .mpgl files. Things will get very confusing if this is messed up!]

cat header_ids.txt gl_common.loci.txtvariants50.mpgl >f_gl_common.loci.txtvariants50.mpgl

3. Use R to transpose the genotype matrix. Start R by typing "R" on the command line and then using the following commands:

common< −read.table("f_gl_common.loci.txtvariants50.mpgl",header=F) [read in the file]

dim(common) [check the dimensions rows = number of loci plus the header line, cols = number of ind.s. should look like this:]

[1] 34547 130

tcommon< −t(common) [transpose matrix]

write.table(tcommon,file="tran_common_loci.txt",row.names=F,col.names=F,quote=F) [write the transposed matrix to file]

q() [quit R]

4. Add a column with population names. This column will be useful for making colored PCA plots etc. Note: you should inspect the perl script, depending on how you have coded the population information.

perl add_pop_info.pl tran_common_loci.txt
This produces 'names_tran_common_loci.txt'

5. Let's take a quick look at the data by doing a PCA on the mean genotype likelihoods in R. This provides a first, cursory look at the data before using the Bayesian clustering in *Entropy*.

Modify and use the R script "BWA_genotype_likelihood_PCA.R". This contains code for the PCA and 2-D and 3-D plotting. [Note: you can do this and the next step at the same time.]

6. Make the infiles for *Entropy*. This requires a PCA followed by k-means clustering which creates starting files for the Bayesian clustering algorithm.

Modify and use the R script "Entropy_infiles.R"

[NOTE: next you will return to your .mpgl file that you made at step 6-1. The files you made here for preparing for entropy are not used again, meaning the following were made for use with "Entropy_infiles.R", but are not used hereafter:
'gl_common.loci.txtvariants50.mpgl'
'f_gl_common.loci.txtvariants50.mpgl'
'tran_common_loci.txt'
'names_tran_common_loci.txt']

7. Add IDs to common loci genotype file. For this you will need to make another header file by adding a line to the "entropy_IDfile.txt" file. Open this file using an editor (vim, emacs, or nano, etc.) and add a new line at the top with three numbers: #ind.s #loci and "1". Name this file "headerT.txt". Then add this header to the genotype likelihood file:

cat headerT.txt common.loci.txtvariants50.mpgl > entropy_common_50.mpgl

[NOTE: CCN watch the "scaffold" issue here - might need to fix before running entropy. ## this should be corrected now]

Now you are ready for *Entropy*.

Philosophy of *Entropy*

Here is the Nice Lab philosophy about best practices regarding *Entropy*:
With STRUCTURE there was no way to check mixing, thus most people ran that model for 10 chains. With ENTROPY, you can check mixing in multiple ways, so we believe that the very first thing to do is check mixing carefully. If the model is working for your data, it is entirely possible to run a minimum of 2 chains and that will be sufficient.

Do these diagnositics for q with 'entropy_mixing_diagnostics-somethingScript.R' [CCN make this happen]

- ESS might be the most important diagnostic
- make history plots for 5% of your individuals
- gelman-rubin diagnostic for all the chains you have run (min 2)

Might be interesting to explore / calculate the corr of genotype probs across chains and across k's

plot q for k's from each chain per k and inspect carefully - look to see if the clusters are the same across chains, this is especially important when mixing is not great and usually at

higher k, make sure you use the same color scheme so that you can see if cluster assignments switch

If everything looks good, use estpost.entropy to combine chains which gives you the median q across the chains that you should use for plotting barplots

for genotype probs for PCA, average across k's
– take g2c0,g3c0,g4c0 ... (first chain from each k and average). This avoids the issue of "which k?". Here is code:

```
X <- list(g2c0,g3c0,g4c0,g5c0,g6c0,g7c0)
Y <- do.call(cbind,X)
Y <- array(Y,dim=c(dim(X[[1]]),length(X)))
dim(Y)
[1]   297 63339  6
meangprob <- apply(Y,c(1,2),mean)
pcaout <- prcomp(meangprob,center=T,scale=F)
summary(pcaout)
```

on DIC: DIC might not work, but, more important, there is no real or best K ever, despite many attempts to find it. You should make barplots for all k's stacked on top of each other because you want to know what the data tells you in full - i.e. all k's provide information about your data.

RUNNING *Entropy*

To look at options for the program, type "entropy" on the command line. You will see this:

entropy version 1.2 – 21 August 2013

Usage: entropy -i infile.txt [options]
-i Infile with genetic data for the population
-l Number of MCMC steps for the analysis [default = 10000]
-b Discard the first n MCMC samples as a burn-in [default = 1000]
-t Thin MCMC samples by recording every nth value [default = 1]
-k Number of population clusters [default = 2]
-e Probability of sequence error, set to '9' for locus-specific error rates [default = 0]
-Q Estimate intra- and interspecific ancestry and marginal q [0 or 1, default = 0]
-o HDF5 format outfile with .hdf5 suffix [default = mcmcout.hdf5]
-m Infile is in genotype likelihood format [default = 0]
-w Output includes population allele frequencies [default = 1]
-q File with expected starting values for admixture proportions
-s Scalar for Dirichlet init. of q, inversly prop. to variance [default = 1]
-p +/- proposal for ancestral allele frequency [default = 0.1]
-f +/- proposal for Fst [default = 0.01]
-y +/- proposal for gamma [default = 0.2]
-a +/- proposal for alpha [default = 0.1]

Here is an example of the command for running *Entropy*:

entropy -i entropy_neoall_common50.mpgl -o out_neoall_common50_entropy2ch0.hdf5 -l 50000 -b 5000 -t 10 -s 20 -e .01 -k 2 -q ldak2common50.txt -m 1 -w 0 &

In order, -i designates the infile, -o is the name of the outfile (make sure it ends in ".hdf5"), -l is the number of MCMC steps, -b -s the burnin, -t is the thinning, -s is the scalar [need to explore this parameter], -e sets the sequence error probability, -k is the number of clusters, -q is the starting values for the admixture proportion (q), -m tells the program that the infile has genotype likelihoods, -w is a toggle for including (or not) population allele frequencies. In this example, all other options were kept at the default (and thus not included on the command line). There is no option for multiple chains.

This gets repeated for each value of k that you are interested in.

Next, use the *estpost.entropy* to extract results from the hdf5 file. You will do separate commands to retrieve admixture proportions (q) and genotype probabilities (gprob) or DIC scores. Here are examples:

for admix. proportions, median, mean, upper and lower CIs:
estpost.entropy out_neoall_common50_entropy2ch0.hdf5 -p q -s 0 -o q2.txt

estpost reports:
file = out_neoall_common50_entropy2ch0.hdf5 parameter dimensions for q: ind = 186, populations = 2, samples = 4500, chains = 1

for gprobs, median, mean, upper and lower CIs:
estpost.entropy out_neoall_common50_entropy2ch0.hdf5 -p gprob -s 0 -o gprob2.txt

estpost reports:
file = out_neoall_common50_entropy2ch0.hdf5
parameter dimensions for gprob: loci = 12352, ind = 186, genotypes = 3, chains = 1

estpost.entropy options:

estpost.entropy version 1.2 - 23 September 2013

Usage: estpost.entropy [options] infile1.hdf5 infile2.hdf5
-o Outfile [default = postout.txt]
-p Name of parameter to summarize, e.g., 'q'
-c Credible interval to calculate [default = 0.95]
-b Number of additinal MCMC samples to discard for burn-in [default = 0]
-h Number of bins for posterior sample histogram [default = 20]
-s Which summary to perform: 0 = posterior estimates and credible intervals
1 = histogram of posterior samples [NICE: what is this??]
2 = convert to plain text [this is useful for checking mixing]
3 = calculate DIC
-w Write parameter identification to file, boolean [default = 1]
-v Display estpost.entropy software version

for option -p, The known parameters are: gprob, zprob, p, q, pi, fst, alpha, gamma, deviance

if you run two (separate) chains, you can retrieve results from both:
estpost.entropy out_neocommon50_entropy7c0.hdf5 out_neocommon50_entropy7c1.hdf5 -o gprob7.txt -p gprob -s 0

To check mixing (only for q at this point) in R:

library(coda) MCMC mixing for entropy

Starting with q: use estpost to get the plain text output of all the mcmc steps, this should be a matrix of each individual's q for all of your steps, for each 'population'.

An example: for k=3 I should have a matrix of 597 (199*3) by 4501 (#of steps +1 for ID column).

It should look somewhat like this:

> mcmc[1:5,1:10]
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
1 q_ind_0_pop_0 0 0.00000 0 0 0 0.00000 0.0000 0.00000 0.00000
2 q_ind_1_pop_0 0 0.00172 0 0 0 0.00000 0.0000 0.00000 0.00000
3 q_ind_2_pop_0 0 0.00000 0 0 0 0.00000 0.0000 0.00000 0.00000
4 q_ind_3_pop_0 0 0.00000 0 0 0 0.00000 0.0000 0.00000 0.00000
5 q_ind_4_pop_0 0 0.00000 0 0 0 0.00144 0.0051 0.00618 0.00093
> mcmc[195:205,1:10] V1 V2 V3 V4 V5 V6 V7 V8 V9
195 q_ind_194_pop_0 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
196 q_ind_195_pop_0 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
197 q_ind_196_pop_0 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
198 q_ind_197_pop_0 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
199 q_ind_198_pop_0 1 1.00000 1.00000 0.99997 1.00000 1.00000 1.0000 1.00000
200 q_ind_0_pop_1 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
201 q_ind_1_pop_1 1 0.99828 0.99874 1.00000 1.00000 1.00000 1.0000 1.00000
202 q_ind_2_pop_1 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
203 q_ind_3_pop_1 1 1.00000 1.00000 1.00000 1.00000 1.00000 1.0000 1.00000
204 q_ind_4_pop_1 1 1.00000 1.00000 1.00000 1.00000 0.99856 0.9949 0.99382
205 q_ind_5_pop_1 1 1.00000 0.99722 0.99734 0.99972 1.00000 1.0000 1.00000
V10
195 1.00000
196 1.00000
197 0.99979
198 1.00000
199 1.00000
200 1.00000
201 1.00000
202 1.00000
203 1.00000
204 0.99907
205 1.00000
> dim(mcmc)

597 4501

for each individual that you want to check, make a new vector with just their mcmc estimates for q. EXAMPLE:

for individual 0, population 0
> ind1_0<-mcmc[1,2:4501]
(it is 2 through 4501 because the first column is the ID info)

> ind1_0t<-t(ind1_0)
transpose the row into a column
> effectiveSize(ind1_0t)
1
731.3597

> plot(ind1_0t)

Now do the same for each of the 3 populations (k=3):
> ind1_1<-mcmc[200,1:4501]
* can keep the first column to check you have the right individual, and remove it in the next step

> ind1_1t<-t(ind1_1[2:4501])
> effectiveSize(ind1_1t)
200
636.9664
> ind1_2<-mcmc[399,1:4501]
> ind1_2[1:10]
V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
399 q_ind_0_pop_2 0 0 0 0 0 0 0 0 0
:

(NOTE: Kate and Chris, write an R script to collect ESS's etc. for all ind.s for q)
(NOTE: Question: are ESS's etc. correlated across multiple k - like is the ESS for q_ind_0_pop_0 equivalent to ESS for q_ind_0_pop_3??)
(NOTE: what other parameters do we need/ should we check mixing?? q seems good)
(NOTE: we need to start thinking carefully about "best practices")

To make barplots and PCAs:

For details see entropy_barplot&PCA.R script.

Below are sections that need to be created:

Running entropy on the cluster (TSU's star cluster - linux sun system [CCN check details]:

[see melissa configuration notes 26/03/14 and 31/03/14 and qsub_entropy_wrapper.pl]

Additions here: using allele frequencies to calculate Nei's D_a and D, making dendrograms
getting estimates of π and θ from SAMTOOLS
script for IBD

9 The *Popmod* Model

10 The *Afreq* Model

BGC

The F model

Other Potentially Useful Stuff

Split libraries

Use after parsing. This step is only needed if there are multiple libraries in one lane.

1. For example, 2 projects were in one Illumina lane (i.e. need to split neophasia from upstairs/downstairs.)

2. First copy splitfastq.pl script to the parsedreads directory (when in parsed read directory); cp /Volumes/data/scrips/splitfastq.pl ./

[NOTE: this is splitfastq.pl with lower case f - there is another script with a upper case F that is not for splitting parsed reads. Careful.]

3. Next open the script using emacs text editor; emacs splitfastq.pl or nano or any other editor (vi, etc.).

4. The perl script is used to identify which IDs belong to which project and transfer the appropriate reads into different outfiles. The script needs to be edited to change the names of the outfiles and change the id identifier (i.e. the sorting is hard-coded in this script).

To save in emacs: cntrl x and cntrl s or to save and exit cntrl x and cntrl c.

5. Now the perl script can be run: perl splitfastq.pl parsed_TSU3.fastq. When the script has finished two new files have been created, one with neophasia project parsed reads and one with u/d parsed reads. Each should only contain reads from individuals from the appropriate projects.

Getting Sequence Reads for Scaffolds (i.e. phylogenetics??)