S1: CallHap Manual

CallHap: A Pipeline for Analysis of Pooled Whole-genome Haplotypes

Last edited: 3/13/2017

By: Elizabeth Hendrickson

Licensing information

With the exception of the Genome Analysis Toolkit, all programs are freely available under either the Gnu Public License or the MIT License. The Genome Analysis Toolkit is free for non-commercial use; other use should contact the Broad Institute at softwarelicensing@broadinstitute.org. Python and bash scripts for the CallHap pipeline are available at https://github.com/cruzan-lab/CallHap.

Introduction

CallHap is a pipeline designed for the analysis of pooled haplotype data. It depends on the presence of two types of sequencing libraries; either single sample libraries (SSLs) or pooled libraries (Pool). Ideally, a Pool should contain equimolar genetic material from 20 individuals, and one of those individuals should be prepared separately as a SSL. This pipeline picks up following sequencing on an Illumina HiSeq or similar high-throughput sequencer.

Requirements

- A LINUX/UNIX/MacOS system with the following programs installed:
  - Cutadapt (http://cutadapt.readthedocs.io/en/stable/index.html)
  - Sickle (http://bioinformatics.ucdavis.edu/research-computing/software/)
  - BWA (http://bio-bwa.sourceforge.net/)
  - Samtools (http://samtools.sourceforge.net/)
  - PicardTools (https://broadinstitute.github.io/picard/)
  - GATK (https://software.broadinstitute.org/gatk/)
  - Freebayes (https://github.com/ekg/freebayes)
  - Python 2.7x (https://www.python.org/) with NumPy (http://www.numpy.org/)
  - Java Development Kit

Example Data Set

An example data set is available on the GitHub. This data set was artificially constructed and represents the state of data after variant filtering but before haplotype calling.

Contents

CallHap: A Pipeline for Analysis of Pooled Whole-genome Haplotypes

Quick start

Detailed Instructions

Quick Start

Setup:

program -config.sh:

Edit program-config.sh so that each of the variables is set to the absolute path of the program in question.

Reference Preparation:

Obtain a reference genome (in FASTA format) for your species of interest (or closely related other species), and prepare it for use by using the following commands:

```
$ bwa index {reference}.fasta
$ samtools faidx {reference}.fasta
$ java -jar /path/to/picardtools/picard.jar \
  CreateSequenceDictionary \
  R={reference}.fasta \
  O={reference}.dict
```

Preprocessing:

There are two basic processing pipelines provided; one with automated trimming (CallHap_Preproc_0.01.23.sh) and one without (CallHap_Preproc_NoTrimming_0.01.23.sh). It is strongly suggested that at least a few (2-5) samples per flow cell be run manually (one step at a time), at least through trimming for quality control and to see if those samples need any additional trimming beyond the basic trimming steps (adapter and quality trimming).
If you are doing trimming separately, be sure to use the locations of the trimmed files in the preconfig instead of the locations of the raw files.

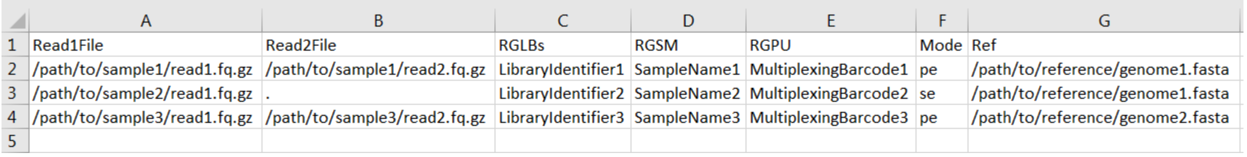
Create a preconfig file in Excel with the following columns:

- Read1File
- Read2File
- RGLB
- RGSM
- RGPU
- Mode
- Reference

Each row should represent one sequencing library (SSL or Pooled).

- Read1File and Read2File should give the absolute path to the locations of the raw data for the Read 1 and Read 2 files (in the case of single end data, give the file location under Read1File, and put a period (.) for Read2File).
- RGLB should be some identifier for the library (e.g. library number).
- RGSM should be a sample name, preferably indicating the species of the library, the location the sample came from, and whether the sample is a SSL or Pool (Example: SpenamLocS#SSL, SpenamLocS#Pool).
- RGPU should indicate the barcoding used for this library during library prep (Example: ATTACTCG-TATAGCCT).
- Mode should be one of se (single-end) or pe (paired-end).
- Reference should indicate the reference genome you would like this library aligned to.

An example preconfig format would be:



	A	B	C	D	E	F	G
1	Read1File	Read2File	RGLBs	RGSM	RGPU	Mode	Ref
2	/path/to/sample1/read1.fq.gz	/path/to/sample1/read2.fq.gz	LibraryIdentifier1	SampleName1	MultiplexingBarcode1	pe	/path/to/reference/genome1.fasta
3	/path/to/sample2/read1.fq.gz	.	LibraryIdentifier2	SampleName2	MultiplexingBarcode2	se	/path/to/reference/genome1.fasta
4	/path/to/sample3/read1.fq.gz	/path/to/sample3/read2.fq.gz	LibraryIdentifier3	SampleName3	MultiplexingBarcode3	pe	/path/to/reference/genome2.fasta
5							

Note: Not all samples have to have the same sequencing mode (se or pe), or same reference genome. If all samples are of the same species, the reference genomes for all libraries should be the same. This example can be found on the GitHub.

Save the preconfig file as a .csv. Convert it to a config file using:

```
$ python /path/to/CallHap/CallHap_ConfigCreator.py \
  --input preconfig.csv \
  --adapt1 {SequencingAdapter} \
  --adapt2 {SequencingAdapter} \
  --sequencer {Sequencer used to produce data} \
  --minBaseQual 30 \
  --minReadQual 30 \
  --runID {Identifier for this run}
```

This will output a .sh file with the run ID as the name (for example, of you put --runID {runID}, the file would be called runID.sh)
Then use the following command to run the rest of the pre-processing (replacing the script name if you did trimming separately):

```
$ bash /path/to/CallHap/CallHap_Preproc_0.01.23.sh \
  program-config.sh {runID}.sh
```

SNP Calling:
Set up an input list of files using:

```
$ ls -1 /path/to/files/*SSL*.rg.ra.bam > {RunID}.txt
$ ls -1 /path/to/files/*Pool*.rg.ra.bam >> {RunID}.txt
```

Call FreeBayes using:

```
$ /path/to/freebayes/freebayes -L {RunID}.txt \
  -p 1 -f /path/to/reference/{reference}.fasta \
  -v {RunID}_SNPs.vcf --use-best-n-alleles 2 \
  --min-repeat-entropy 1 --no-partial-observations \
  --min-alternate-fraction 0.03
```

This step may take a while, and while running, may look like it isn't doing anything

Preparation for SNP Filtering and Haplotype Calling:
Create a text file (poolSizeFile.txt) containing two tab-separated columns in the following format:

    PoolName	numIndividualsInPool

This text file will inform later analyses and must be tab-delineated. The list should contain one pool name per line as well as the size (#) of that specific pool.
This will be used as the argument to -p in both the SNP filtering and Haplotype Calling steps.

SNP Filtering:
SNP filtering is accomplished by use of a custom python script, which can be run with the command:

```
$ python /path/to/CallHap/CallHap_VCF_Filt.py \
  -i {RunID}_SNPs.vcf -o {RunID}_d{600}q{20}_Haps.vcf \
  -O {RunID}_d{600}q{20}_Pools.vcf -n <number of SSLs> \
  -N <number of Pools> -d {600} -q {20} \
  -p {poolSizeFile.txt} --dropLowDepth {200}
```

You may need to trim off one or more columns from the VCF file if one sample was not called at a majority of positions; if a single sample is not called at a particular position, the variant at that position will be discarded. To determine if a column needs to be removed, look at your VCF file in Excel, and see if there are any columns that are periods (“.”) for the majority of rows. Removing the column can also be done in Excel, but you need to be careful because Excel likes to add quotes when it saves files with commas in the cells, as do most spreadsheet editors I’ve found. Alternatively, set `--dropLowDepth` to a low value (say, 100) and any libraries with an average depth less than that will be automatically dropped.

Haplotype Calling:
Before running this step, check how many cores are available on the system you’re using with htop. Make sure you don’t overload the system you’re working on; don’t set -t to higher than the number of available cores, and don’t take up all the cores on the machine.

Haplotype calling can be run using:

```
$ python /path/to/CallHap/CallHap_HapCallr.py \
  --inputHaps {RunID}_d{600}q{20}_Haps.vcf \
  --inputFreqs {RunID}_d{600}q{20}_Pools.vcf \
  -o {RunID} -p {poolSizeFile.txt} -t {5} -l {2} \
  --numRandom {100} --numTopRSS {3} --genpop --structure
```

This program generates four to six output file per solution output (within the minimum number of RSS values):

- A NEXUS file (RunID_solNum_haps.nex) for network phylogeny creation; PopART (http://popart.otago.ac.nz/index.shtml) works fairly well. I’ve been using the TCS algorithm.
- A VCF file (RunID_solNum_PredFreqs.vcf) containing the estimated SNP frequencies based

on the estimated haplotype frequencies, and the per-SNP average residuals in the INFO field

- A CSV file (RunID_solNum_freqs.csv) containing the per-pool haplotype frequencies and RSS values for each pool.
- A CSV file (RunID_solNum_Regression.csv) containing paired observed and predicted SNP frequencies from the Least Squared algorithm.
- (Optional): A Structure formatted file (RunID_solNum_iterNum.str) containing the expanded haplotype frequencies
- (Optional): A Genpop file containing the haplotype frequencies for use in Adegenet.

In addition, outputs are generated describing the original haplotypes network (RunID_Initial.nex), the unique haplotypes network (RunID_Unique.nex), raw topologies observed from each random order (RunID_RAW.csv), the frequency of each unique topology generated (RunID_topologies.csv), the frequency of occurrence for each haplotype found in any random order (RunID_summary.csv).

In terms of population-genetics analysis, haplotypes should be treated as independent alleles at a single locus.

Output Files – Formats:

<u>Nexus</u>

NEXUS is a format used to store information about genetic identities of individuals, either as measurements of morphological traits, or as genotyped identities. The NEXUS files produced by CallHap have the format:

```
##NEXUS
Begin Data;
          Dimensions ntax=<number_of_haplotypes>
ncahr=<number_of_snps>;
          Format datatype=DNA missing=N gap=-;
          Matrix
Hap1 Hap1Sequence
Hap2 Hap2Sequence
Hap3 Hap3Sequence
…
HapN HapNSequence
;
End;
```

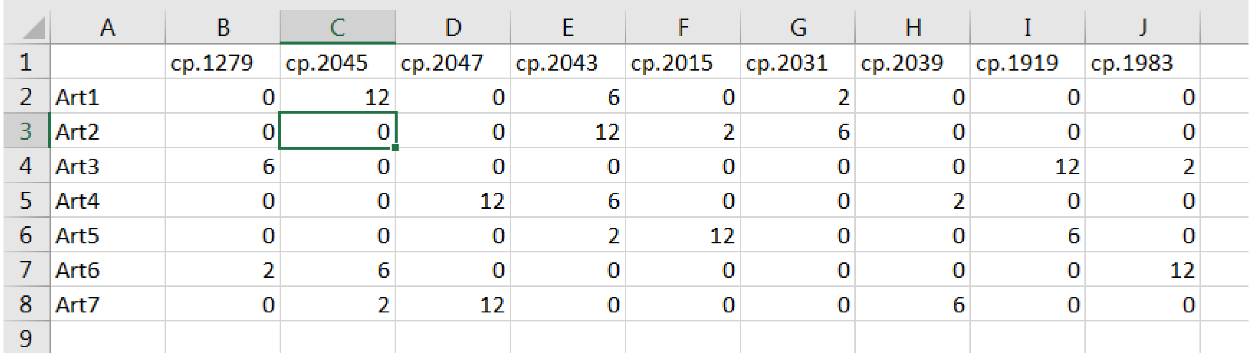
These files can be used in a program such as TCS (http://darwin.uvigo.es/software/tcs.html) or PopArt (http://popart.otago.ac.nz/index.shtml) to produce a network phylogeny.

The VCF filter produces a NEXUS containing all SSLs, and the haplotype caller produces a nexus file of the input SSLs, a nexus file of the unique haplotypes, and a nexus file for each output solution.

Genpop (.genpop)

The genpop file is a csv file designed to be imported into Adegenet. It describes the genetic structure of a population. In the genpop file, populations are rows, and haplotypes are columns. The labeling structure for a genpop table includes what locus the particular column is part of, in this case cp=chloroplast, and the number after the period is a base-ten representation of the binary number that defined the haplotype in question. This means that, as long as the same input SNPs in the same orders are used, with the same reference and alternate identities, different runs of CallHap should come up with the same end haplotypes. The values in the cells represent the number of individuals in that population with that haplotype.

Example:



	A	B	C	D	E	F	G	H	I	J
1		cp.1279	cp.2045	cp.2047	cp.2043	cp.2015	cp.2031	cp.2039	cp.1919	cp.1983
2	Art1	0	12	0	6	0	2	0	0	0
3	Art2	0	0	0	12	2	6	0	0	0
4	Art3	6	0	0	0	0	0	0	12	2
5	Art4	0	0	12	6	0	0	2	0	0
6	Art5	0	0	0	2	12	0	0	6	0
7	Art6	2	6	0	0	0	0	0	0	12
8	Art7	0	2	12	0	0	0	6	0	0
9										

This file contains a lot of the information from the frequencies file, but can be easily imported into R for further analysis. A suggested script for importing these files into R is shown below:

```
library("ape")
library("pegas")
library("seqinr")
library("ggplot2")
library("adegenet")

# Function to load genpop files
load.genpop = function(fileName) {
  testGenPop <- read.delim(fileName, header=TRUE,
  row.names = 1, sep=",")
  myGenpop <- as.genpop(tab=testGenPop, ploidy=1)
  return(myGenpop)
}

myGenpop <- load.genpop(inputFile)
```

If genpop output is turned on, the haplotype caller produces one genpop file for each output solution.

Structure (.str)

In this file, each row is a different individual. The first column gives an identifier for the individual, the second column identifies which population that individual belongs to, and the

third column gives the base-ten identifier for the haplotype that individual has.
If structure output is turned on, the haplotype caller produces one structure file for each output solution.

Regression file (_Regression.csv)
This file can be used to plot observed vs. predicted graphs to get an idea for how good of a fit the end haplotype model was. Each row represents a SNP in a particular pool. The first column gives the pool identifier, the second gives the SNP number, the third gives the observed SNP frequency, and the fourth gives the predicted SNP frequency based on a particular haplotype model.
The haplotype caller produces one regression file for each output solution.

Frequencies file (_freqs.csv)
Contains all the information that the genpop file contains, but also identifies which haplotypes are new and which were from single sample libraries, as well as providing per-population RSS values.
The haplotype caller produces one frequencies file for each output solution.

Predicted Frequencies file _PredFreqs.vcf)
VCF-formatted file showing the predicted SNP frequencies for each pool. The haplotype caller produces one predicted frequencies file for each output solution.

Topologies file (_topologies.csv)
A csv file that shows the haplotype identities and frequencies for all output topologies. This file can be used to help select which output topology to use.

Raw file (_RAW.csv)
A csv file that gives haplotype identities and occurrences for all produced models, not just those selected for output. Also provides information about which ordering the model came from, and what RSS value a model produced. Can be used to help select final models.

Detailed Instructions

Adapter and Quality Trimming:

Adapter and quality trimming should be performed before any other step in the pipeline. This ensures better read alignment and higher quality of the final data. The automated pipeline uses cutadapt for adapter trimming and sickle for quality trimming; however, you can use other trimming programs if so desired.

Cutadapt is available at http://cutadapt.readthedocs.io/en/stable/index.html under the MIT License and can be run using:

```
$ /path/to/cutadapt -a {inAdapter1} -A {inAdapter2} \
  -o {output_read_1}_at.fastq.gz \
  -p {output_read_2}_at.fastq.gz \
  {input_read_1}.fastq.gz {input_read_2}.fastq.gz
```

for paired-end reads or

```
$ /path/to/cutadapt -a {inAdapter1} \
  -o {output_read_1}_at.fastq.gz {input_read_1}.fastq.gz
```

for single-end reads.

If you aren't certain what adapter sequence you have, running FastQC (freely available at http://www.bioinformatics.babraham.ac.uk/projects/fastqc/ under GPLv3) may help determine what adapters are present. Otherwise, consult your library preparation protocol.

While cutadapt can also do quality trimming (using the -q option), or remove a fixed number of bases (using the -u option), the default pipeline uses a second program, (sickle) for quality trimming. Sickle is available at http://bioinformatics.ucdavis.edu/research-computing/software/ under the MIT License and can be run with

```
$ /path/to/sickle pe -f {output_read_1}_at.fastq.gz \
  -r {output_read_2}_at.fastq.gz -o \
  {output_read_1}ut_at_qt.fastq.gz -p \
  {output_read_2}_at_qt.fastq.gz -t sanger -s \
  {SampleName}_extras.fastq.gz -q {minBaseQuality} -g
```

for paired-end reads or

```
$ /path/to/sickle se -f {output_read_1}_at.fastq.gz \
  -o {output_read_1}_at_qt.fastq.gz -t sanger \
  -q {minBaseQuality} -g
```

for single-end reads.

For more details on these programs, consult their respective manuals.

Following trimming, it is recommended that at least 2-5 samples per flow cell be quality-checked using FastQC. For this pipeline, check that there are almost no remaining adapters of any type in the AdapterContent page of the report and that you are satisfied with the quality scores in the Per base sequence quality section and the base percentages in the Per base sequence content section.

Note: FastQC will generate output files in the same directory as the input files.

Read Alignment:
The automated pipeline uses BWA-mem to align reads with default options. BWA can be obtained from http://bio-bwa.sourceforge.net/ under GPLv3, and can be run using:

```
$ /path/to/bwa mem -M {reference}.fasta \
  {output_read_1}_at_qt.fastq.gz \
  {output_read_2}_at_qt.fastq.gz > \
  {SampleName}.sam
```

for paired-end reads or

```
$ /path/to/bwa mem -M {reference}.fasta \
  {output_read_1}_at_qt.fastq.gz > {SampleName}.sam
```

for single-end reads.
After alignment, the file is converted to a bam file:

```
$ /path/to/samtools view -Sbu -F 4 {SampleName}.sam | \
  /path/to/samtools sort - {SampleName}.sort
```

Index the bam file:

```
$ /path/to/samtools index {SampleName}.sort.bam
```

At this time, any unaligned reads are also removed.
Samtools can be obtained from http://www.htslib.org/.

Readgroup Creation:
PicardTools is used to add readgroups to the files. These are a requirement for local realignment with GATK, and for SNP calling with FreeBayes. For later analysis, it is useful if each sample have a different sample name (RGSM) and readgroup ID (RGID), since Freebayes (our SNP caller) uses the readgroup ID to differentiate samples. I used the library number as the readgroup ID.

PicardTools is available at https://broadinstitute.github.io/picard/, and can be run using

```
$ java -jar /path/to/picard AddOrReplaceReadGroups \
  INPUT={SampleName}.sort.bam \
  OUTPUT={SampleName}.sort.rg.bam \
  RGID={ReadGroupID} \
  RGLB={ReadGroupLibrary} \
  RBPL={ReadGroupSequencingPlatform} \
  RGPU={ReadGroupRunBarcode} \
  RGSM={ReadGroupSampleName} \
  CREATE_INDEX=true
```

RGLB, RBPL, RGPU, and RGSM are required for this tool to run.
RGID needs to be different for each library.

Local Realignment:
Local realignment is carried out using the Genome Analysis Toolkit (GATK, available at https://software.broadinstitute.org/gatk/). The first step in this process is to locate targets for local realignment:

```
$ java -jar /path/to/GATK -T RealignerTargetCreator \
  -R {reference}.fasta \
  -I {SampleName}.sort.rg.bam \
  -o {SampleName}.sort.rg.intervals
```

Following this, local realignment can be run using:

```
$ java -jar /path/to/GATK -T IndelRealigner \
  -R {reference}.fasta \
  -I {SampleName}.sort.rg.bam \
  -targetIntervals {SampleName}.sort.rg.intervals \
  -o {SampleName}.sort.rg.ra.bam \
  -dt NONE --maxReadsForRealignment 200000
```

SNP Calling:
Set up an input list of files:

```
$ ls -1 /path/to/files/*SSL*.rg.ra.bam > {RunID}.txt
$ ls -1 /path/to/files/*Pool*.rg.ra.bam >> {RunID}.txt
```

Or whatever identifier you used to differentiate PLs and SSLs. The important thing is that this file list all SSLs, followed by all PLs. Next, call FreeBayes:

```
$ /path/to/freebayes/freebayes -L {RunID}.txt \
  -p 1 -f /path/to/reference/{reference}.fasta \
  -v {RunID}_SNPs.vcf --use-best-n-alleles 2 \
  --min-repeat-entropy 1 --no-partial-observations \
  --min-alternate-fraction 0
```

This step may take a while, and while running, may look like it isn't doing anything.
FreeBayes can be found at https://github.com/ekg/freebayes.

Preparation for SNP Filtering and Haplotype Calling
Create a text file containing two tab-separated columns in the following format:

```
	PoolName	numIndividualsInPool
```

This will be used as the argument to -p in both the SNP filtering and Haplotype Calling steps.

SNP Filtering:
If you didn't call variants using FreeBayes, your VCF file must contain the "DP", "RO", and "AO" format fields for this program to work.

Before running SNP filtering, it may be necessary to trim off one or more columns from the VCF file if one sample was not called at a majority of positions; if a single sample is not called at a particular position, the variant at that position will be discarded, so a single sample uncalled (or at low depth) at a majority of positions can result in no data making it through the filtering step. To determine if a column needs to be removed, look at your VCF file in Excel, and see if there are any columns that are periods (".") for the majority of rows. Removing the column can also be done in Excel, but you need to be careful because Excel likes to add quotes when it saves files with commas in the cells, as do most spreadsheet editors I've found. Alternatively, set --dropLowDepth to a low value (say, about ½ the minimum depth you want to call at), and the filter will automatically filter out any samples with a mean depth less than that depth.

If desired, sample depth can be assessed using the GATK DepthOfCoverage tool (see https://software.broadinstitute.org/gatk/documentation/tooldocs/org_broadinstitute_gatk_tools_walkers_coverage_DepthOfCoverage.php for instructions). This tool takes a similar amount of time to SNP calling.

SNP filtering is accomplished by use of a custom python script, which can be run with the command:

```
$ python /path/to/CallHap/CallHap_VCF_Filt.py \
  -i {RunID}_SNPs.vcf -o {RunID}_d{600}q{20}_Haps.vcf \
  -O {RunID}_d{600}q{20}_Pools.vcf -n <number of SSLs> \
  -N <number of Pools> -d {600} -q {20} \
  -p {poolSizeFile.txt} --dropLowDepth {200}
```

`-i, --inVCF`	The input VCF file from FreeBayes
`-o, --outHaps`	The output haplotypes file, containing haplotypes found in the SSLs
`-O, --outPools`	The output Pool SNP frequencies file, containing frequency of the more common allele in each pool
`-n, --numSamps`	The number of SSLs in the input file
`-N, --numPools`	The number of Pools in the input file
`-d, --minDepth`	This option sets the minimum read depth that must be present at a position in ALL libraries in order for that position to be considered as a variant. It should be set based on the number

	of individuals in a PL. For haploid sequence, a depth of 15 per individual in the pool is recommended (Sims et al., 2014), so that for a pool of 20 individuals, a depth of 300 is required at a site to be able to call variants.
-p, --poolSizes	A file detailing the number of individuals in each pooled library
-q, --minQual	Controls the minimum PHRED-scaled variant quality needed to use a variant. Mostly useful for filtering out super-low quality variants, but can be set higher as necessary. -p is the number of individuals in each pool.
--minCallPrev	Controls the maximum allowable error in SSLs for a variant to be called. It can range from 1 (all reads in each SSL need to have the same identity) to 0.5 (Up to half the reads in a SSL can have a different identity). At a setting of 1, some real SNPs could be removed based on unavoidable errors in the SSLs, while at a setting of 0.5, confidence in the identity call for SSLs, and thus in the identity of haplotypes, will be significantly decreased. I set this parameter at a default of 0.9, to allow for some sequencing error in the SSLs while still maintaining a high accuracy of SSL identity calls.
--minSnpPrev	Coupled with poolSize, this option controls how much of a PL must be the alternate identity for a SNP to be at that position when there is no variation in the SSLs. The value is a positive floating-point decimal, which gets multiplied by 1/poolSize to yield the proportion of reads that must be of a different identity in a PL to yield a variant. At a value of zero, all positions would be called as variants if there was any variation in a PL. I set this at a default value of 0.75 in order to allow for some error in low-frequency haplotypes, while removing the majority of low-frequency sequencing errors from consideration.
--indelDist	How far away from indels a variant should be for use. IndelDist takes an integer value greater than 0; at a value of 0, distance from an indel will not be considered as a filter. I set this at a relatively conservative value of 100 (the length of my raw sequencing reads) as being the maximum distance at which the presence of an indel could have any effect on variant discovery.
--dropLowDepth	Automatically drop any samples with an average depth less than this depth. This should be no greater than the requested minimum depth. Failure to set this flag to something other than the default may result in loss of variants based on incomplete coverage. Default is to keep all columns.
--reportInterval	Report progress at this number of lines, default = 1000

It is recommended to run this program with different sets of parameters to determine what the optimum parameters will be for a particular run.

Haplotype Calling:

Before running this step, check how many cores are available on the system you're using with htop. Make sure you don't overload the system you're working on; don't set -t to higher than the number of available cores, and don't take up all the cores on the machine. Haplotype calling can be run using:

```
$ python /path/to/CallHap/CallHap_HapCallr.py \
  --inputHaps {RunID}_d{600}q{20}_Haps.vcf \
  --inputFreqs {RunID}_d{600}q{20}_Pools.vcf \
  -o {RunID} -p {poolSizeFile.txt} -t {5} -l {2} \
  --numRandom {100} --numTopRSS {3} --genpop --structure
```

`-i, --inputHaps`	The haplotypes file from SNP filtering
`-f, --inputFreqs`	The Pools file from SNP filtering
`-o, --outPrefix`	A unique output prefix for this run of haplotype caller
`-p, --poolSizes`	A file describing the number of individuals in each pool
`-t, --processes`	The number of threads to use during processing
`-l, --numIterations`	The number of times to iterate across the SNPs within each order
`-r, --highResidual`	How high a residual should be able to exist after adding a SNP, and is used to defer processing of a SNP where the residual doesn't reduce enough to another iteration.
`--dropFinal`	A flag which pairs with -r to remove SNPs with a high residual entirely at the end if they don't reduce the residual enough. May not work with current random ordering algorithm; don't use for now.
`--genpop`	A flag that instructs CallHap to generate genpop output
`--structure`	A flag that instructs CallHap to generate structure formatted output
`--numRandom`	Controls how many pseudo-random orderings of SNPs to use, and should be a value greater than zero. I set this value at 100 as a compromise between run time and increased chance of finding the correct solution; in practice, this value should be set based on the number of starting haplotypes relative to the number of SNPs present. If the number of starting haplotypes is close to the number of SNPs, this value can be low; the maximum number of haplotypes in the network is one more than the number of SNPs. However, if the number of SNPs is greater than the number of haplotypes, more attempts may be needed to

	help resolve the best network topology.
--numTopRSS	This option just influences how many RSS values down are processed for the final output solutions, and should be an integer greater than zero. I set it at a value of 3 so I could examine the higher RSS value solutions.
--saveFrequency	The frequency with which to save progress; this allows for a run to be restarted if it gets interrupted in the middle. The default is to not save progress (0).
--keepTemps	Keeps CallHap from deleting temporary files created when saving progress at the end of a run.
--noSearch	Generates haplotypes for the original input haplotype file; does not find novel haplotypes.
--restart	Restart the program from last saved points in temporary files.
-v, --version	Print version.
-d, --deterministic	SNPs are sorted from most frequent to least frequent before haplotypes are generated. Reduces runtime. Default program completely randomizes SNPs.
-a, --randHighFreq	Randomize a specified number of the most frequent SNPs before haplotypes are generated. Slightly reduces runtime. Default program completely randomizes SNPs. If flagged, the top 5 SNPs are randomized, unless otherwise specified.
-n, --numTop	Used in conjunction with –a flag. Specifies the number of most frequent SNPs to be randomized.

This program generates four to six output file per solution output (within the minimum number of RSS values):

- A NEXUS file (RunID_solNum_haps.nex) for network phylogeny creation; PopART (http://popart.otago.ac.nz/index.shtml) works fairly well. I've been using the TCS algorithm.
- A VCF file (RunID_solNum_PredFreqs.vcf) containing the estimated SNP frequencies based on the estimated haplotype frequencies, and the per-SNP average residuals in the INFO field
- A CSV file (RunID_solNum_freqs.csv) containing the per-pool haplotype frequencies and RSS values for each pool.
- A CSV file (RunID_solNum_Regression.csv) containing paired observed and predicted SNP frequencies from the Least Squared algorithm.
- (Optional): A Structure formatted file (RunID_solNum_iterNum.str) containing the expanded haplotype frequencies
- (Optional): A Genpop file containing the haplotype frequencies for use in Adegenet.

In addition, outputs are generated describing the original haplotypes network (RunID_Initial.nex), the unique haplotypes network (RunID_Unique.nex), raw topologies observed from each random order (RunID_RAW.csv), the frequency of each unique topology generated (RunID_topologies.csv), the frequency of occurrence for each haplotype found in any random order (RunID_summary.csv).

In terms of population-genetics analysis, haplotypes should be treated as independent alleles at a single locus.

Runtime Observations

There are several factors that influence runtime; these include number of SNPs, number of unique haplotypes, and number of pools. In general:

- More SNPs will lead to higher runtimes
    - SNPs external to the network (not between nodes on the initial network phylogeny) will increase runtime more than interior SNPs.
- More initial unique haplotypes will decrease the runtime
- More PLs will increase the runtime
- Sorting SNPs by frequency (-d flag) or randomizing only some of the SNPs (-a flag) may reduce overall runtime. However, note that complete randomization (default) with multiple orders and iterations hypothetically produces the most accurate hypotheses.

Minimum runtime occurs when all haplotypes present in the pools have been sampled in the SSLs.

Reruns with Consistently Added Haplotypes

In some cases, it may be useful to rerun the Haplotype calling steps with additional haplotypes. To determine if this step should be done, examine the _topologies file generated by CallHap_HapCallr.py in Excel. Sort it by RSS value (decreasing) and occurrences (increasing). If the best RSS value(s) only occur in a few random orders out of all random orders, check to see if there are any haplotypes consistently added in the best ~5% of topologies. If there are, try adding these haplotypes to the known haplotypes file and rerunning CallHap using the modified set of input haplotypes.

A script (AugmentHaps.py) has been provided to help with this process. To use it, first create a text file with the decimal form of the haplotypes you want to add, one per line. Then invoke AugmentHaplotypes.py with the following command:

```
$ /path/to/CallHap/AugmentHaps.py \
  --inputHaps {RunID}_d250q20_Haps.vcf \
  --newHaps {newHapsFile.txt} \
  -o {RunID}_d250q20_HapsExtended.vcf
```

Selecting the Best Model:

There are three methods of selecting the best model: select the model with lowest RSS value, select the most common model, or select the model with the lowest AIC (or AICc) value. The difference is between lower error values and preference for a smaller model. As a note, with AIC, smaller models are weighted more heavily than lower RSS values. To select a model, examine the _topologies file and sort it by RSS, then by Occurrences.

AIC can be calculated as:

$$AIC = 2 * numHaps + \left(numSNPs * \log\left(\frac{RSS}{numSNPs}\right)\right)$$

and AICc (corrected AIC) can be calculated as:

$$AICc = 2 * numHaps + \left(numSNPs * \log\left(\frac{RSS}{numSNPs}\right)\right) + \frac{2 * numHaps * (numHaps + 1)}{numSNPs - numHaps - 1}$$

Setup and Running on a SLURM Environment:

Although specifics of running CallHap on a given system will vary based on the parameters of the system in question, the following general guidelines may prove useful for users with a SLUM HPC environment.

While CallHap can be run parallelized, due to the Global Interpreter Lock in Python, only muti-processing is possible. Practically, what this means is that CallHap is limited to as many processes as there are cores on the processor, and cannot distribute processes across multiple processors. The best way to minimize run-time for a run where you want to do 100 random orders (-t 100) where a single node has less than 100 cores is to start enough parallel nodes that you have 100 total cores, and then merge the solutions from those nodes afterwards. For example, in a HPC environment where the general computing node has 20 cores, the following can be used to run 100 solutions at top speed:

```
#!/bin/bash
#SBATCH --job-name ExampleRun
#SBATCH --partition long
#SBATCH --array=1-5
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 1
#SBATCH --cpus-per-task 20
srun --unbuffered python
/path/to/CallHap/CallHap_HapCallr.py --inputHaps \
ExampleRun_Haps.vcf --inputFreqs ExampleRun_Pools.vcf \
--poolsize 20 --outPrefix \
ExampleRun_${SLURM_ARRAY_TASK_ID}out --processes 20 \
--numIterations 2 --highResidual 100 --genpop --structure \
--numRandom 20 --numTopRSS 3 --saveFrequency 10
```

Common Errors:

Problem	Solution
Quick-start pipeline produces empty files	Check that input files defined in the preconfig exist
Multiple best-RSS solutions	If one occurs more frequently than the other, use that one. If both occur equally, check to see if the network phylogenies for each solution look the same, and if the generated haplotype frequencies look the same. If the generated haplotype frequencies are identical, it doesn’t

	matter which haplotype is actually present. If generated haplotype frequencies differ, create non-biologically relevant pools containing the same DNA samples, but shuffled in new ways (perhaps by using individual 1 from each population as one pool, individual 2 from each population as a second, and so on).