BINF 6110 – Project 3

Name: Abelhard Jauwena

Student ID: 1040413

Introduction

The goal of variant calling is to identify variations in a DNA sequence sample using high-throughput sequencing technology (*Variant Calling*, n.d.). Variant calling also seeks to differentiate between true variants and experimental errors (Bohannan & Mitrofanova, 2019). Clinical genomic studies usually focus on identifying three different types of variants, namely 1) single nucleotide variants (SNVs, which are comprised of single-base substitutions at certain parts of the genome), 2) small insertions and deletions (indels), and 3) structural variants (SVs) (Bohannan & Mitrofanova, 2019; *Summoning insights*, 2023; Zverinova & Guryev, 2021). Among these three variant types, SNVs are the easiest to identify (*Summoning insights*, 2023). Variant calling supports genomics-related research in many ways. For example, researchers can use data obtained from variant calling to conduct fine mapping, which is a method of identifying genetic variants associated with complex traits (Schaid et al., 2018; *Variant Calling*, n.d.). Additionally, variant calling data can also give insights into the genomic bases of many diseases (e.g., cancer), as SNVs, small indels, and SVs often serve as important disease markers (*Summoning insights*, 2023). In turn, these insights can pave the way for advancements in the field of precision medicine (*Summoning insights*, 2023).

The availability of many kinds of variant calling software has allowed researchers to conduct variant calling much more quickly and conveniently. Among them, BCFtools (formerly SAMtools) and Freebayes remain the two most used variant callers (Bohannan & Mitrofanova, 2019). BCFtools is a Hidden Markov model-based variant caller that identifies variants through two main commands (Liu et al., 2022). The first command, `bcftools mpileup`, reads many different alignments and then creates a vertical slice for all the reads that cover each position in a genome (Danecek et al., 2021). BCFtools then calculates genotype likelihoods for all the reads, which determines how consistent the observed data are with the possible genotypes in each position (Danecek et al., 2021). The second command, `bcftools call`, uses the Hardy-Weinberg equilibrium to evaluate the most likely genotype for each position and, subsequently, call variants (Danecek et al., 2021; freebayes, *a*, n.d.). The allele frequencies used in the Hardy-

Weinberg equilibrium can be derived from the data or given by the user (Danecek et al., 2021). BCFtools has several advantages over other variant callers, namely that it consumes less memory, has higher speed, can convert between VCF and BCF formats, can process third-party formats using the "convert" command, and can manipulate variant calls in both compressed and uncompressed VCF and BCF files (Danecek et al., 2021; Liu et al., 2022). Nevertheless, BCFtools presents several major downfalls. First, it still encounters problems processing extremely large files, which typically necessitates multiple computers to run tasks in parallel (Danecek et al., 2021). For instance, BCFtools may fail to represent all the highly polymorphic sites in a very large BCF file containing tens of thousands of samples (Danecek et al., 2021). Moreover, some BCFtools commands are only equipped to handle haploid and diploid organisms (Danecek et al., 2021). Lastly, BCFtools currently only provide limited support when it comes to processing large "64-bit" genomes (Danecek et al., 2021).

In contrast, Freebayes is a Bayesian variant caller designed to find small variations (Bian et al., 2018). These variations include single nucleotide polymorphisms (SNPs, which are SNVs that occur in germline DNA), indels, multinucleotide polymorphisms (MNPs), and complex events whose lengths are smaller than the length of a short-read sequencing alignment (Illumina, n.d.; freebayes, *a*, n.d.). Unlike BCFtools, Freebayes is haplotype-based, and so it identifies variants by literally aligning read sequences to a target instead of via precise alignments (freebayes, *a*, n.d). Being a haplotype-based variant caller, Freebayes avoids one major issue with alignment-based variant calling, which is that identical sequences can have multiple possible alignments (freebayes, *a*, n.d.). As such, Freebayes mostly shows satisfactory performances when performing variant calling on various genome sequences, such as those obtained from sheep or wheat (Stegemiller et al., 2023; Yao et al., 2020). However, Freebayes usually requires more processing and filtering steps to extract clean variants from data sets (Stegemiller et al., 2023). When these steps are neglected, Freebayes often presents several issues, such as showing low specificity and sensitivity or failing to detect more SNPs with increasing input read depths (Liu et al., 2022; Yao et al., 2020). Considering these tradeoffs between the two variant callers, this report aims to further investigate the performances of BCFtools and Freebayes by evaluating how accurately they can identify variants from 10 individual burbot DNA sequences.

Methods

Below is a stepwise list of the methods I used to conduct my analyses. I have also indicated the programming language used at the top of each block.

Preparation

Make all the required directories in Graham.

```
Unix Shell
mkdir /scratch/ajauwena/binf_6110/p3 # This will be my working directory.
cd /scratch/ajauwena/binf_6110/p3
mkdir burbot_genome burbot_raw_data bwa_assem sorted_bam_files
sorted_bam_files_read_groups results_bcftools results_freebayes analyses
```

Change to the directories containing the files for analysis. Then, copy all the files to my corresponding directories.

```
Unix Shell
# The directory containing the reference genome.
cd /scratch/emandevi/genomic_methods_w23/Project3/burbot_genome
scp * ${wd}/burbot_genome

# The directory containing the burbot raw data.
cd /scratch/emandevi/genomic_methods_w23/Project3/burbot_raw_data
scp * ${wd}/burbot_raw_data

# The directory containing the burbot .sam files obtained using the "Burrows-Wheeler-
Aligner" ("BWA") (Burrows-Wheeler, n.d.).
cd /scratch/emandevi/genomic_methods_w23/Project3/bwa_assem
scp * ${wd}/bwa_assem
```

Change to my working directory. Then, request additional memory through an interactive run.

```
Unix Shell
cd /scratch/ajauwena/binf_6110/p3
srun --pty --account="def-lukens" -t 0-05:00:00 --mem=30000 /bin/bash
```

Create all the required scripts. I have provided these scripts in the **Appendix** section at the end of this report.

```
Unix Shell
nano sam_to_sorted_bam_converter.sh
nano variant_caller_and_vcf_file_combiner_bcftools.sh
nano read_group_adder.sh
```

```
nano read_group_reindexer.sh
nano variant_caller_and_vcf_file_combiner_freebayes.sh
nano filterer.sh
nano snp_number_finder.sh
nano bgzipper_and_indexer.sh
nano snp_overlap_finder.sh
nano allele_frequency_and_site_depth_calculator.sh
```

Make all the scripts executable.

```
Unix Shell
chmod +x sam_to_sorted_bam_converter.sh
variant_caller_and_vcf_file_combiner_bcftools.sh read_group_adder.sh
read_group_reindexer.sh variant_caller_and_vcf_file_combiner_freebayes.sh filterer.sh
snp_number_finder.sh bgzipper_and_indexer.sh snp_overlap_finder.sh
allele_frequency_and_site_depth_calculator.sh
```

Load the appropriate packages in R.

```
R
library(dplyr)
library(tidyverse)
```

Variant Calling

Execute the script “sam_to_bam_converter.sh.”

```
Unix Shell
./sam_to_sorted_bam_converter.sh /scratch/ajauwena/binf_6110/p3/bwa_assem
/scratch/ajauwena/binf_6110/p3/sorted_bam_files
```

Change to the directory “bwa_assem.” Then, move all the sorted .bam files to the directory “sorted_bam_files.”

```
Unix Shell
cd bwa_assem
mv *.bam /scratch/ajauwena/binf_6110/p3/sorted_bam_files
```

Load the required module and execute the script “variant_caller_and_vcf_file_combiner_bcftools.sh.”

```
Unix Shell
module load bcftools
./variant_caller_and_vcf_file_combiner_bcftools.sh
/scratch/ajauwena/binf_6110/p3/burbot_genome
```

```
/scratch/ajauwena/binf_6110/p3/sorted_bam_files
/scratch/ajauwena/binf_6110/p3/results_bcftools
```

Load the required module and execute the script “read_group_adder.sh.”

```
Unix Shell
module load picard
./read_group_adder.sh /scratch/ajauwena/binf_6110/p3/sorted_bam_files
/scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups
```

Load the required module and execute the script “read_group_reindexer.sh.”

```
Unix Shell
module load samtools
./read_group_reindexer.sh /scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups
```

Load the required modules and execute the script

“variant_caller_and_vcf_file_combiner_freebayes.sh.”

```
Unix Shell
module load nixpkgs/16.09 gcc/7.3.0 freebayes/1.2.0 java/1.8.0_192
./variant_caller_and_vcf_file_combiner_freebayes.sh
/scratch/ajauwena/binf_6110/p3/burbot_genome
/scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups
/scratch/ajauwena/binf_6110/p3/results_freebayes
```

Filtering VCF Files

Load the required modules and execute the script “filterer.sh” on the directories

“results_bcftools” and “results_freebayes.”

```
Unix Shell
module load StdEnv/2020
module load nixpkgs/16.09 intel/2018.3
module load vcftools/0.1.16
./filterer.sh /scratch/ajauwena/binf_6110/p3/results_bcftools
./filterer.sh /scratch/ajauwena/binf_6110/p3/results_freebayes
```

Extracting SNP-Related Information

Execute the script “snp_number_finder.sh.”

```
Unix Shell
./snp_number_finder.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
```

```
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f /scratch/ajauwena/binf_6110/p3/analyses
```

Execute the script “bgzipper_and_indexer.sh.”

```
                                        Unix Shell
./bgzipper_and_indexer.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f /scratch/ajauwena/binf_6110/p3/results_bcftools
/scratch/ajauwena/binf_6110/p3/results_freebayes
```

Execute the script “snp_overlap_finder.sh.”

```
                                        Unix Shell
./snp_overlap_finder.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered_for_bgzippi
ng.recode.vcf.gz
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered_for_bgzip
ping.recode.vcf.gz /scratch/ajauwena/binf_6110/p3/analyses
```

Execute the script “allele_frequency_and_site_depth_calculator.sh.”

```
                                        Unix Shell
./allele_frequency_and_site_depth_calculator.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f /scratch/ajauwena/binf_6110/p3/analyses
```

Transfer the files for analysis in the directory “analyses” to my local computer.

```
# I used the program “WinSCP” for this step.
```

Obtaining the Number of SNPs

Read in the file corresponding to the number of SNPs identified using both software as a data frame.

```
                                              R
# Read in the appropriate file.
snp_number_df <- read.table('snp_number.txt',
                            sep = '\t',
                            header = FALSE)

# Set the values in the first row as column names.
names(snp_number_df) <- snp_number_df[1, ]
snp_number_df <- snp_number_df[-1, ]
```

```
# View the number of SNPs.
View(snp_number_df)
```

Obtaining the Percentage of SNPs That Overlap

Read in the file corresponding to the overlap of SNPs as a data frame.

```
                                                R
# Read in the appropriate file.
snp_overlap_df <- read.table('snp_overlap.txt',
                             sep = '\t',
                             header = FALSE)

# Set the values in the first row as column names.
names(snp_overlap_df) <- snp_overlap_df[1, ]
snp_overlap_df <- snp_overlap_df[-1, ]

# View the overlap of SNPs.
View(snp_overlap_df)
```

Obtaining Minor Allele Frequencies

Read in the file corresponding to the allele frequencies of all SNPs identified using BCFtools as a data frame. Then, plot a histogram showing the distribution of these minor allele frequencies.

```
                                                R
# Read in the appropriate file.
allele_freq_df_bcftools <-
read.table('results_bcftools_filtered_minor_allele_frequency.frq',
                                      sep = '\t',
                                      header = FALSE,
                                      row.names = NULL,
                                      fill = TRUE)
allele_freq_df_bcftools <- allele_freq_df_bcftools[-1, ]
colnames(allele_freq_df_bcftools) <- c('CHROM', 'POS', 'N_ALLELES', 'N_CHR',
'FREQ_1', 'FREQ_2')
# Note: Reading .frq files into R is rather difficult because there is no header for
the sixth column. As such, I proceeded to set "header" as "FALSE," "row.names" as
"NULL," and "fill" as "TRUE," as you can see above. Afterward, I removed the first
row and wrote the column names manually. I also used this approach for the .frq file
obtained from using Freebayes.

# Convert the allele frequencies into numeric values.
allele_freq_df_bcftools$FREQ_1 <- as.numeric(allele_freq_df_bcftools$FREQ_1)
allele_freq_df_bcftools$FREQ_2 <- as.numeric(allele_freq_df_bcftools$FREQ_2)
```

```
# Subset the columns corresponding to the allele into a separate data frame for
looping.
allele_freq_subset_df_bcftools <- allele_freq_df_bcftools %>%
  select(FREQ_1, FREQ_2)

# Create an empty vector that will contain all the allele frequencies (i.e., both
major and minor).
maf_vec_bcftools <- c()

# Loop through each row and column in the data frame subset and append all the values
to the empty vector above.
for (i in 1:nrow(allele_freq_subset_df_bcftools)) {
  for (j in 1:ncol(allele_freq_subset_df_bcftools)) {
    maf_vec_bcftools <- append(x = maf_vec_bcftools,
                               values = allele_freq_subset_df_bcftools[i, j])
  }
}

# Create another data frame containing all the allele frequencies from the vector
above.
maf_df_bcftools <- data.frame(maf_vec_bcftools)

# Plot the histogram.
maf_df_bcftools %>%
  drop_na(maf_vec_bcftools) %>%
  ggplot(aes(x = maf_vec_bcftools)) +
  geom_histogram(color = 4,
                 fill = 'black',
                 binwidth = 0.05) +
  ggtitle('Fig. 1. A histogram of the distribution of minor allele frequencies across
all SNPs identified using BCFtools') +
  xlab('Minor Allele Frequency') +
  ylab('SNP Count') +
  xlim(0, 0.5) +
  ylim(0, 10000)
```

Read in the file corresponding to the allele frequencies of all SNPs identified using Freebayes as a data frame. Then, plot a histogram showing the distribution of these minor allele frequencies.

```
                                                R
# Read in the appropriate file.
allele_freq_df_freebayes <-
read.table('results_freebayes_filtered_minor_allele_frequency.frq',
                                        sep = '\t',
                                        header = FALSE,
                                        row.names = NULL,
                                        fill = TRUE)
```

```
allele_freq_df_freebayes <- allele_freq_df_freebayes[-1, ]
colnames(allele_freq_df_freebayes) <- c('CHROM', 'POS', 'N_ALLELES', 'N_CHR',
'FREQ_1', 'FREQ_2')

# Convert the allele frequencies into numeric values.
allele_freq_df_freebayes$FREQ_1 <- as.numeric(allele_freq_df_freebayes$FREQ_1)
allele_freq_df_freebayes$FREQ_2 <- as.numeric(allele_freq_df_freebayes$FREQ_2)

# Subset the columns corresponding to the allele into a separate data frame for
looping.
allele_freq_subset_df_freebayes <- allele_freq_df_freebayes %>%
  select(FREQ_1, FREQ_2)

# Create an empty vector that will contain all the allele frequencies (i.e., both
major and minor).
maf_vec_freebayes <- c()

# Loop through each row and column in the data frame subset and append all the values
to the empty vector above.
for (i in 1:nrow(allele_freq_subset_df_freebayes)) {
  for (j in 1:ncol(allele_freq_subset_df_freebayes)) {
    maf_vec_freebayes <- append(x = maf_vec_freebayes,
                                values = allele_freq_subset_df_freebayes[i, j])
  }
}

# Create another data frame containing all the allele frequencies from the vector
above.
maf_df_freebayes <- data.frame(maf_vec_freebayes)

# Plot the histogram.
maf_df_freebayes %>%
  drop_na(maf_vec_freebayes) %>%
  ggplot(aes(x = maf_vec_freebayes)) +
  geom_histogram(color = 2,
                 fill = 'black',
                 binwidth = 0.05) +
  ggtitle('Fig. 2. A histogram of the distribution of minor allele frequencies across
all SNPs identified using Freebayes') +
  xlab('Minor Allele Frequency') +
  ylab('SNP Count') +
  xlim(0, 0.5) +
  ylim(0, 10000)
```

Obtaining Depth Across All SNPs

Read in the file corresponding to the depths of all SNPs (summed across all 10 individuals) identified using BCFtools as a data frame. Then, plot a histogram showing the distribution of the depths of these SNPs.

```R
# Read in the appropriate file.
summed_depth_df_bcftools <-
read.table('results_bcftools_filtered_summed_depth.ldepth',
                                        sep = '\t',
                                        header = FALSE)

# Set the values in the first row as column names.
names(summed_depth_df_bcftools) <- summed_depth_df_bcftools[1, ]
summed_depth_df_bcftools <- summed_depth_df_bcftools[-1, ]

# Convert the summed depths into numeric values.
summed_depth_df_bcftools$SUM_DEPTH <- as.numeric(summed_depth_df_bcftools$SUM_DEPTH)

# Plot the histogram.
summed_depth_df_bcftools %>%
  drop_na(SUM_DEPTH) %>%
  ggplot(aes(x = SUM_DEPTH)) +
  geom_histogram(color = 4,
                 fill = 'black') +
  ggtitle('Fig. 3. A histogram of the distribution of depths across all SNPs (summed
across all 10 individuals) identified using BCFtools') +
  xlab('Depth') +
  ylab('SNP Count') +
  xlim(0, 6000) +
  ylim(0, 25000)
```

Read in the file corresponding to the depths of all SNPs (summed across all 10 individuals) identified using Freebayes as a data frame. Then, plot a histogram showing the distribution of the depths of these SNPs.

```R
# Read in the appropriate file.
summed_depth_df_freebayes <-
read.table('results_freebayes_filtered_summed_depth.ldepth',
                                         sep = '\t',
                                         header = FALSE)

# Set the values in the first row as column names.
names(summed_depth_df_freebayes) <- summed_depth_df_freebayes[1, ]
```

```
summed_depth_df_freebayes <- summed_depth_df_freebayes[-1, ]

# Convert the summed depths into numeric values.
summed_depth_df_freebayes$SUM_DEPTH <-
as.numeric(summed_depth_df_freebayes$SUM_DEPTH)

# Plot the histogram.
summed_depth_df_freebayes %>%
  drop_na(SUM_DEPTH) %>%
  ggplot(aes(x = SUM_DEPTH)) +
  geom_histogram(color = 2,
                 fill = 'black') +
  ggtitle('Fig. 4. A histogram of the distribution of depths across all SNPs (summed
across all 10 individuals) identified using Freebayes') +
  xlab('Depth') +
  ylab('SNP Count') +
  xlim(0, 6000) +
  ylim(0, 25000)
```

Obtaining Mean Depth*

**Measured in reads per locus per individual*

Read in the file corresponding to the mean depths of all SNPs identified using BCFtools as a data frame. Then, plot a histogram showing the distribution of the mean depths of these SNPs.

```
                                                R
# Read in the appropriate file.
mean_depth_df_bcftools <-
read.table('results_bcftools_filtered_mean_depth.ldepth.mean',
                                      sep = '\t',
                                      header = FALSE)

# Set the values in the first row as column names.
names(mean_depth_df_bcftools) <- mean_depth_df_bcftools[1, ]
mean_depth_df_bcftools <- mean_depth_df_bcftools[-1, ]

# Convert the mean depths into numeric values.
mean_depth_df_bcftools$MEAN_DEPTH <- as.numeric(mean_depth_df_bcftools$MEAN_DEPTH)

# Plot the histogram.
mean_depth_df_bcftools %>%
  drop_na(MEAN_DEPTH) %>%
  ggplot(aes(x = MEAN_DEPTH)) +
  geom_histogram(color = 4,
                 fill = 'black') +
  ggtitle('Fig. 5. A histogram of the distribution of mean depths across all SNPs
(measured in reads per locus per individual) identified using BCFtools') +
```

```
  xlab('Mean Depth') +
  ylab('SNP Count') +
  xlim(0, 600) +
  ylim(0, 25000)
```

Read in the file corresponding to the mean depths of all SNPs identified using Freebayes as a data frame. Then, plot a histogram showing the distribution of the mean depths of these SNPs.

```
                                                R
# Read in the appropriate file.
mean_depth_df_freebayes <-
read.table('results_freebayes_filtered_mean_depth.ldepth.mean',
                                        sep = '\t',
                                        header = FALSE)

# Set the values in the first row as column names.
names(mean_depth_df_freebayes) <- mean_depth_df_freebayes[1, ]
mean_depth_df_freebayes <- mean_depth_df_freebayes[-1, ]

# Convert the mean depths into numeric values.
mean_depth_df_freebayes$MEAN_DEPTH <- as.numeric(mean_depth_df_freebayes$MEAN_DEPTH)

# Plot the histogram.
mean_depth_df_freebayes %>%
  drop_na(MEAN_DEPTH) %>%
  ggplot(aes(x = MEAN_DEPTH)) +
  geom_histogram(color = 2,
                 fill = 'black') +
  ggtitle('Fig. 6. A histogram of the distribution of mean depths across all SNPs
(measured in reads per locus per individual) identified using Freebayes') +
  xlab('Mean Depth') +
  ylab('SNP Count') +
  xlim(0, 600) +
  ylim(0, 25000)
```

Results

Below are the tables and figures showing the results of my analyses.

Number of SNPs

bcftools	freebayes
9183	38634

Tab. 1. The number of SNPs identified using BCFtools and Freebayes.

Tab. 1 shows that Freebayes was able to identify approximately four times more SNPs compared to BCFtools.

The Percentage of SNPs That Overlap

percentage_snp_bcftools	percentage_snp_overlap_over_snp_bcftools	percentage_snp_overlap_over_snp_freebayes	percentage_snp_freebayes
21.0	79.0	18.8	81.2

Tab. 2. The percentage of SNPs identified using BCFtools and Freebayes in the 10 burbot sequences, as well as the percentage of overlapping SNPs that were also identified using BCFtools and Freebayes.

The results shown in *Tab. 2* are consistent with the results shown in *Tab. 1*; that is, the percentage of SNPs identified using Freebayes in the 10 burbot sequences was approximately four times greater than the percentage of SNPs identified using BCFtools. Consequently, the number of overlapping SNPs that were also detected using Freebayes is significantly smaller compared to the ones that were also detected using BCFtools, owing to the greater number of SNPs Freebayes was able to identify.

Moreover, we can deduce the percentage of overlapping SNPs from the second and third columns of *Tab. 2*. Firstly, we can obtain the percentage of overlapping SNPs with respect to the percentage of SNPs identified using BCFtools.

$$\frac{\%\ of\ overlapping\ SNPs}{\%\ of\ SNPs\ identified\ using\ BCFtools\ mpileup} = \%\ of\ overlapping\ SNPs\ also\ identified\ using\ BCFtools\ mpileup$$

$$\frac{\%\ of\ overlapping\ SNPs}{21.0\ \%} = 79.0\%$$

$$\%\ of\ overlapping\ SNPs = \mathbf{0.1659}\%$$

Secondly, we can obtain the percentage of overlapping SNPs with respect to the percentage of SNPs identified using Freebayes.

$$\frac{\%\ of\ overlapping\ SNPs}{\%\ of\ SNPs\ identified\ using\ Freebayes} = \%\ of\ overlapping\ SNPs\ also\ identified\ using\ Freebayes$$

$$\frac{\%\ of\ overlapping\ SNPs}{81.2\ \%} = 18.8\%$$

$$\%\ of\ overlapping\ SNPs = \mathbf{0.152656}\%$$

Lastly, from the values above, we can obtain the average percentage of SNPs that overlap across the two variant callers, which is approximately 0.16%.

$$Average\ \%\ of\ overlapping\ SNPs = \frac{0.1659\% + \ 0.152656\%}{2}$$

$$Average\ \%\ of\ overlapping\ SNPs = 0.159278\%$$

$$Average\ \%\ of\ overlapping\ SNPs \approx \mathbf{0.16}\%$$

Minor Allele Frequencies

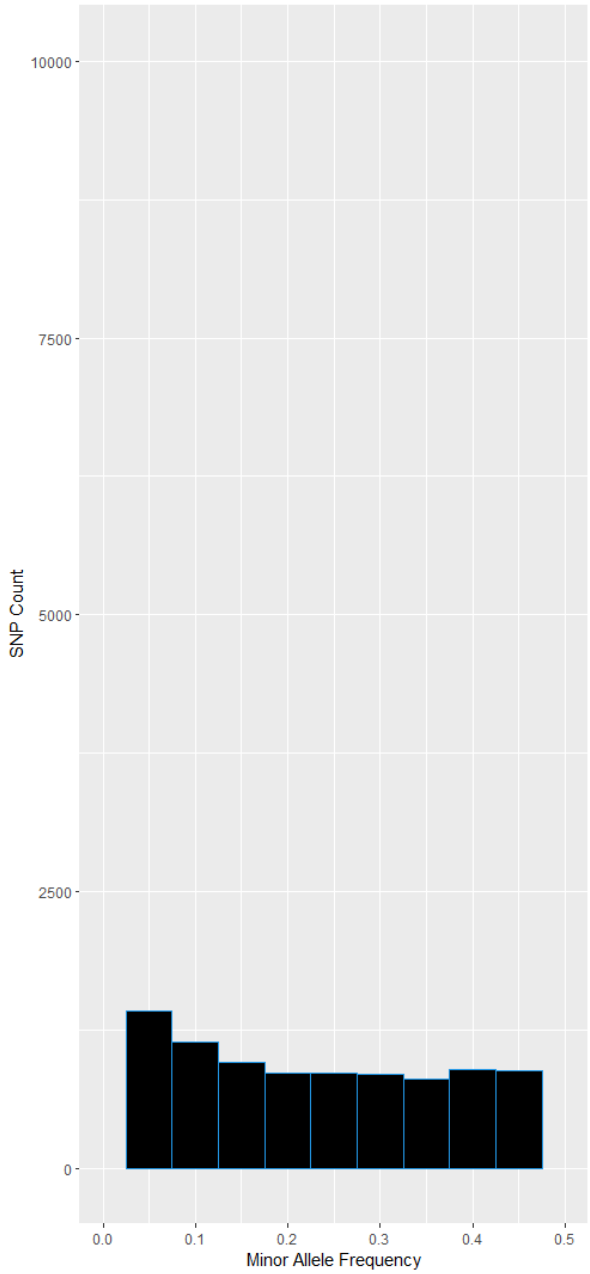

Fig. 1. A histogram of the distribution of minor allele frequencies across all SNPs identified using BCFtools.

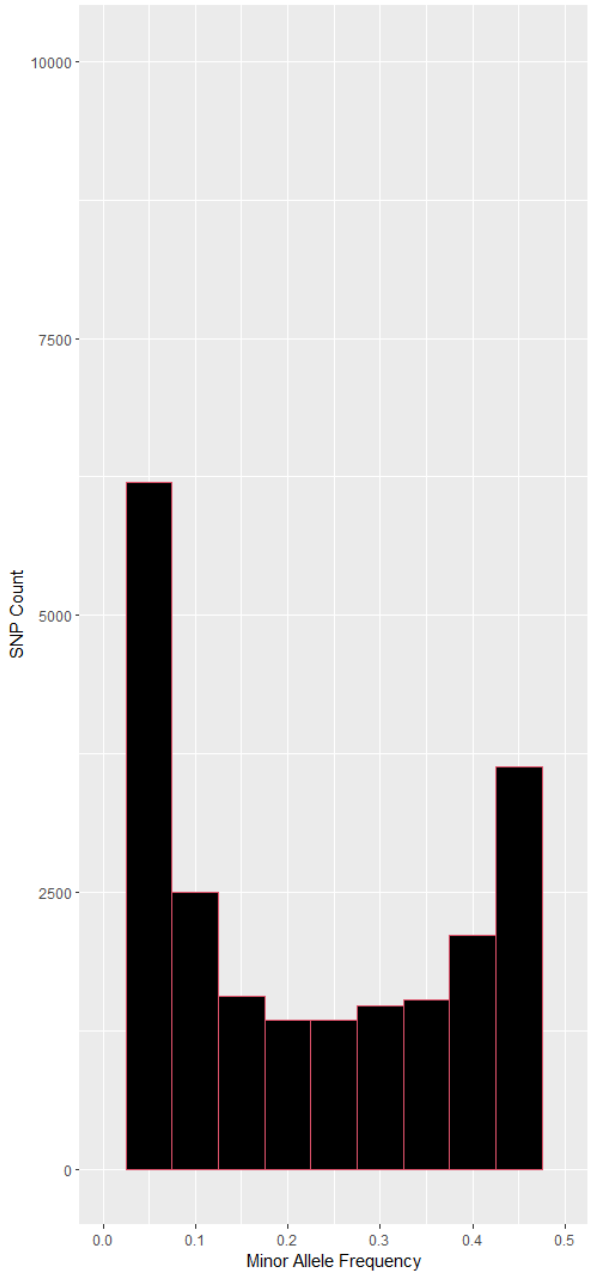

Fig. 2. A histogram of the distribution of minor allele frequencies across all SNPs identified using Freebayes.

Again, comparing *Fig. 1* against *Fig. 2* shows that Freebayes identified significantly more SNPs than BCFtools. Furthermore, both *Fig. 1* and *Fig. 2* show that most minor alleles occur rarely in the 10 burbot sequences, although Freebayes identified several minor alleles that occur quite frequently as well.

Depth Across All SNPs

Fig. 3. A histogram of the distribution of depths across all SNPs (summed across all 10 individuals) identified using BCFtools.

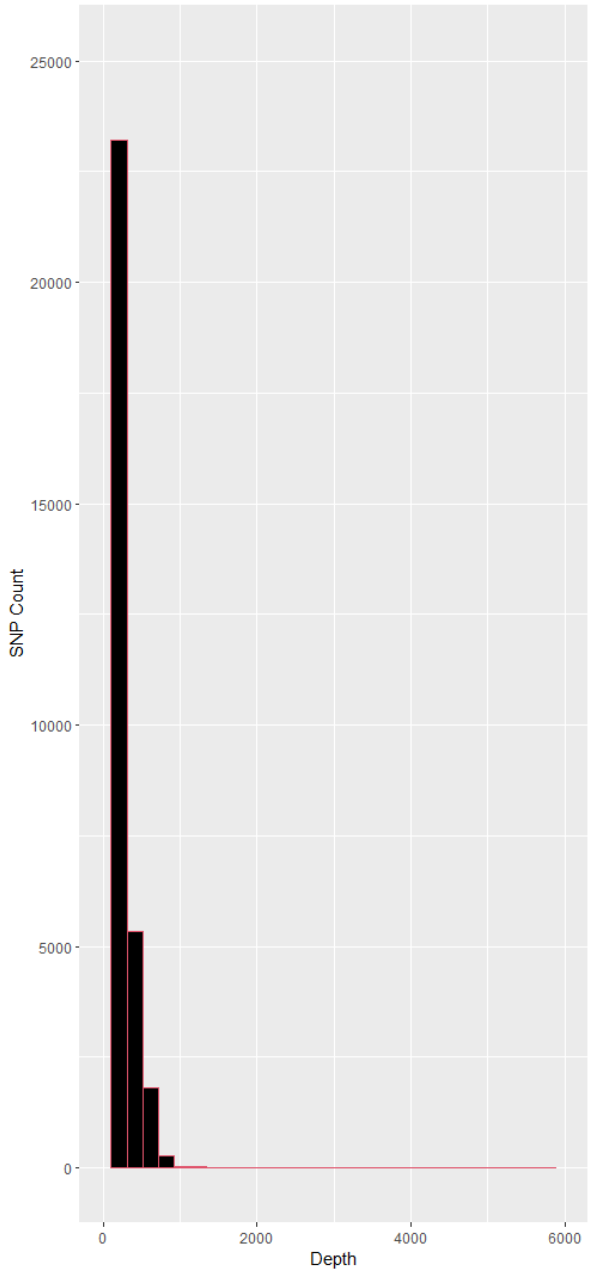

Fig. 4. A histogram of the distribution of depths across all SNPs (summed across all 10 individuals) identified using Freebayes.

Fig. 3 and *Fig. 4* show that most SNPs were read with low depth. However, Freebayes was able to read sequences with significantly higher depth than BCFtools.

Mean Depth*

**Measured in reads per locus per individual*

Fig. 5. A histogram of the distribution of mean depths across all SNPs (measured in reads per locus per individual) identified using BCFtools.

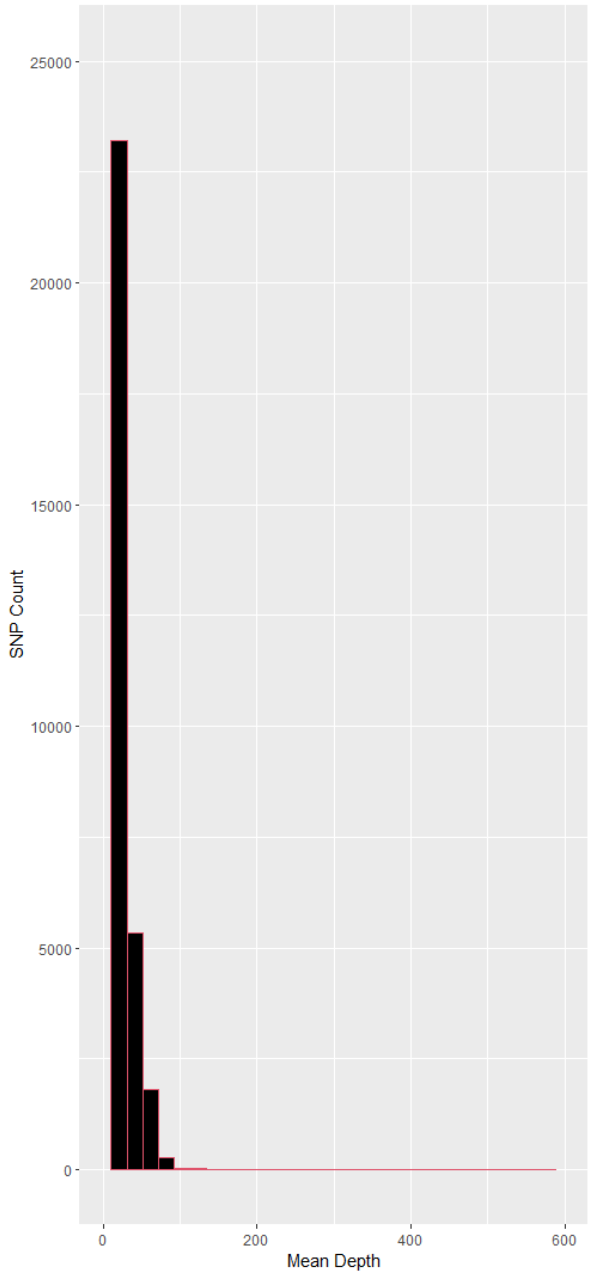

Fig. 6. A histogram of the distribution of mean depths across all SNPs (measured in reads per locus per individual) identified using Freebayes.

Both *Fig. 5* and *Fig. 6* show that most SNPs were read with low depth throughout the 10 burbot sequences. Again, Freebayes was able to read SNPs with much greater depth than BCFtools.

Discussion

Freebayes consistently showed higher performances than BCFtools in detecting variants in the 10 burbot sequences. Specifically, Freebayes detected significantly more SNPs and read them with much greater depth compared to BCFtools. Surprisingly, many papers report conflicting results when comparing the performances of the two variant callers. Yao et al. (2020), for example, reported that following the mapping of raw sequence reads using the BWA-mem aligner, BCFtools showed the highest sensitivity and specificity compared to Freebayes and five other variant callers. Moreover, Liu et al. (2022) reported that at high sequencing depths, BCFtools-single called more SNPs than Freebayes, which identified the fewest SNPs among five other SNP calling pipelines (Liu et al., 2022). Both BCFtools-single and BCFtools-multiple also consistently exhibited greater sensitivity and specificity at various input read depths (i.e., from 5x to 50x) compared to Freebayes (Liu et al., 2022). However, the results from this report do support the fact that Freebayes was able to detect the highest number of SNPs at around 20x sequencing depth (see *Fig. 4* and *Fig. 6*) (Liu et al., 2022).

There are several reasons as to why the results from this report do not align with the results found in the literature. Firstly, as mentioned previously, BCFtools is limited in its ability to represent polymorphic sites in very large files (Danecek et al., 2021). As such, BCFtools may struggle to identify SNPs and read them with adequate depth in the 10 burbot sequences, which range from 33-71 MB in size. Secondly, the choice of alignment software may affect how well the two variant callers perform. The literature suggests that some aligner-variant caller pairs may show higher performances than others. For example, the BWA-mem aligner and BCFtools variant caller pairing accurately identified the most variants in wheat whole genome capture (WEC) re-sequencing data (Yao et al., 2020). Similarly, the BBMap aligner and Freebayes variant caller pairing showed the best performance when detecting variants in data sets derived from *C. elegans* (Smith & Yun, 2017). Unfortunately, no information was given pertaining to the alignment software that was used to produce the 10 burbot sequences analyzed in this report.

Thirdly, the filtering steps that were taken (see "Appendix 6: 'filterer.sh'") may have allowed Freebayes to extract clean variants from the 10 burbot sequences. To be exact, these steps filtered for sites with a minimum quality score of 30, a minor allele count greater than or equal to one, a minor allele frequency less than or equal to 0.5, no missing data, no indels, and no more than two alleles (*VCFtools*, n.d.). As a result, Freebayes was able to detect significantly

more SNPs and read them with much greater depth compared to BCFtools. Lastly, the characteristics of the 10 burbot sequences may bias Freebayes over BCFtools; specifically, these sequences were obtained from a non-model organism, are of low quality (e.g., they contain many ambiguous nucleotides), and have low coverage (i.e., they have only a few nucleotides aligned to their corresponding loci in the reference genome) (National Library, n.d.; The Sequencing, n.d.). Because Freebayes is haplotype-based and operates by literally aligning sequences to the reference genome, the missing and/or unaligned nucleotides present in the data set may have a less severe impact on its performance (freebayes, *a*, n.d.).

The results from this report open many possible avenues for exploration. For example, future studies can investigate how the choice of alignment software impacts the performances of different variant callers. Alternatively, future studies can also consider applying additional filtering steps to include more and/or higher-quality sites. Example filtering steps can include explicitly specifying read depth, genotype likelihoods (i.e., which genotype is most likely to occur at a given site), and genotype quality (i.e., the probability that the called genotype is wrong given that the site is polymorphic) (Danecek et al., 2011).

References

Bian, X., Zhu, B., Wang, M., Hu, Y., Chen, Q., Nguyen, C., Hicks, B., & Meerzaman, D. (2018). Comparing the performance of selected variant callers using synthetic data and genome segmentation. *BMC Bioinformatics*, *19*(1). **https://doi.org/10.1186/s12859-018-2440-7**

Bohannan, Z. S., & Mitrofanova, A. (2019). Calling Variants in the Clinic: Informed Variant Calling Decisions Based on Biological, Clinical, and Laboratory Variables. *Computational and Structural Biotechnology Journal*, *17*, 561–569. **https://doi.org/10.1016/j.csbj.2019.04.002**

Burrows-Wheeler Aligner. (n.d.). Burrows-Wheeler Aligner. Retrieved March 27, 2023, from **https://bio-bwa.sourceforge.net/**

Danecek, P., Auton, A., Abecasis, G. R., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., & Durbin, R. (2011). The variant call format and VCFtools. *Bioinformatics*, *27*(15), 2156–2158. **https://doi.org/10.1093/bioinformatics/btr330**

Danecek, P., Bonfield, J. K., Liddle, J., Marshall, J., Ohan, V., Pollard, M. O., Whitwham, A., Keane, T. E., McCarthy, S. A., Davies, R. J. O., & Li, H. (2021). Twelve years of SAMtools and BCFtools. *GigaScience*, *10*(2). **https://doi.org/10.1093/gigascience/giab008**

freebayes*, a haplotype-based variant detector*. (n.d.). GitHub. Retrieved March 21, 2023, from **https://github.com/freebayes/freebayes**

Illumina. (n.d.). *Efficiently detect single nucleotide polymorphisms and variants*. **https://www.illumina.com/techniques/popular-applications/genotyping/snp-snv-genotyping.html**

Liu, J., Shen, Q., & Bao, H. (2022). Comparison of seven SNP calling pipelines for the next-generation sequencing data of chickens. *PLOS ONE*, *17*(1), e0262574. **https://doi.org/10.1371/journal.pone.0262574**

National Library of Medicine. (n.d.). *Assembly Anomalies and Other Reasons a Genome Assembly may be Excluded from RefSeq*. **https://www.ncbi.nlm.nih.gov/assembly/help/anomnotrefseq/**

Schaid, D. J., Chen, W., & Larson, N. B. (2018). From genome-wide associations to candidate causal variants by statistical fine-mapping. *Nature Reviews Genetics, 19*(8), 491–504. **https://doi.org/10.1038/s41576-018-0016-z**

Smith, H. C., & Yun, S. (2017). Evaluating alignment and variant-calling software for mutation identification in C. elegans by whole-genome sequencing. *PLOS ONE, 12*(3), e0174446. **https://doi.org/10.1371/journal.pone.0174446**

Stegemiller, M. R., Redden, R. R., Notter, D. R., Taylor, T., Taylor, J. B., Cockett, N. E., Heaton, M. P., Kalbfleisch, T. S., & Murdoch, B. M. (2023). Using whole genome sequence to compare variant callers and breed differences of US sheep. *Frontiers in Genetics, 13*. **https://doi.org/10.3389/fgene.2022.1060882**

Summoning insights: NGS variant calling best practices. (2023, January 11). OGT. Retrieved March 16, 2023, from **https://www.ogt.com/ca/about-us/ogt-blog/summoning-insights-ngs-variant-calling-best-practices/**

The Sequencing Center. (n.d.). *What is sequencing coverage?* **https://thesequencingcenter.com/knowledge-base/coverage/**

Variant Calling. (n.d.). CD Genomics. Retrieved March 15, 2023, from **https://www.cd-genomics.com/variant-calling.html**

VCFtools. (n.d.). VCFtools. Retrieved March 24, 2023, from **https://vcftools.github.io/man_latest.html**

Yao, Z., You, F. M., N'Diaye, A. T., Majidi, M. M., McCartney, C. A., Hiebert, C. W., Pozniak, C. J., & Xu, W. (2020). Evaluation of variant calling tools for large plant genome re-sequencing. *BMC Bioinformatics, 21*(1). **https://doi.org/10.1186/s12859-020-03704-1**

Zverinova, S., & Guryev, V. (2021). Variant calling: Considerations, practices, and developments. *Human Mutation, 43*(8), 976–985. **https://doi.org/10.1002/humu.24311**

Appendix

Appendix 1: "sam_to_sorted_bam_converter.sh"

```
                                        Unix Shell
# To execute this script, run:
./sam_to_sorted_bam_converter.sh /scratch/ajauwena/binf_6110/p3/bwa_assem
/scratch/ajauwena/binf_6110/p3/sorted_bam_files

# Command line arguments:
# $0: ./sam_to_sorted_bam_converter.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/bwa_assem (a directory containing the .sam files
for the 10 individual fish).
# $2: /scratch/ajauwena/binf_6110/p3/sorted_bam_files (a directory that will contain
the sorted .bam files for the 10 individual fish).

# Loop through each .sam file in the appropriate directory.
for file in $1/*
do

      # Extract the file's base name.
      base_name=$(echo ${file} | rev | cut -d '.' -f3 | cut -d '/' -f1 | rev)

      # Convert the .sam file to a sorted .bam file.
      samtools sort -O bam -o $2/${base_name}_sorted.bam ${file}

done
```

Appendix 2: "variant_caller_and_vcf_file_combiner_bcftools.sh"

```
                                        Unix Shell
# To execute this script, run:
./variant_caller_and_vcf_file_combiner_bcftools.sh
/scratch/ajauwena/binf_6110/p3/burbot_genome
/scratch/ajauwena/binf_6110/p3/sorted_bam_files
/scratch/ajauwena/binf_6110/p3/results_bcftools

# Command line arguments:
# $0: ./variant_caller_and_vcf_file_combiner_bcftools.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/burbot_genome (a directory containing the burbot
reference genome).
# $2: /scratch/ajauwena/binf_6110/p3/sorted_bam_files (a directory containing the
sorted .bam files for the 10 individual fish).
# $3: /scratch/ajauwena/binf_6110/p3/results_bcftools (a directory containing the
results obtained from calling variants using BCFtools).

# Loop through each sorted .bam file in the appropriate directory.
```

```
for file in $2/*
do

	# Append the file to a list.
	echo ${file} >> $3/bam_file_list_bcftools.txt

done

# Call variants in the files in the list using BCFtools.
bcftools mpileup -a DP,AD -f $1/burbot_2021.fasta -b $3/bam_file_list_bcftools.txt |
bcftools call -m --variants-only > $3/results_bcftools.vcf
```

Appendix 3: "read_group_adder.sh"

```
                                        Unix Shell
# To execute this script, run:
./read_group_adder.sh /scratch/ajauwena/binf_6110/p3/sorted_bam_files
/scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups

# Command line arguments:
# $0: ./read_group_adder.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/sorted_bam_files (a directory containing the
sorted .bam files for the 10 individual fish).
# $2: /scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups (a directory
containing the sorted .bam files for the 10 individual fish with read groups).

# Set a counter.
c=1

# Loop through each sorted .bam file in the appropriate directory.
for file in $1/*
do

	# Extract the file's base name.
	base_name=$(echo ${file} | rev | cut -d '.' -f2 | cut -d '/' -f1 | rev)

	# Inform the user of the file that is being processed.
	echo "Adding a read group to" ${base_name} "..."

	# Replace the read groups in the file.
	java -jar $EBROOTPICARD/picard.jar AddOrReplaceReadGroups I=${file}
O=$2/${base_name}_rg.bam RGID=${c} RGLB=lib1 RGPL=illumina RGPU=unit1
RGSM=${base_name}

	# Increment the counter by 1.
	c=$((c + 1))
```

```
done
```

Appendix 4: “read_group_reindexer.sh”

```
                                        Unix Shell
# To execute this script, run:
./read_group_reindexer.sh /scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups

# Command line arguments:
# $0: ./read_group_reindexer.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups (a directory
containing the sorted .bam files for the 10 individual fish with read groups).

# Loop through each sorted .bam file with a read group in the appropriate directory.
for file in $1/*
do

      # Extract the file's base name.
      base_name=$(echo ${file} | rev | cut -d '.' -f2 | cut -d '/' -f1 | rev)

      # Append the file to a list.
      echo "Reindexing" ${base_name} "..."

      # Append the file to a list.
      samtools index ${file}

done
```

Appendix 5: “variant_caller_and_vcf_file_combiner_freebayes.sh”

```
                                        Unix Shell
# To execute this script, run:
./variant_caller_and_vcf_file_combiner_freebayes.sh
/scratch/ajauwena/binf_6110/p3/burbot_genome
/scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups
/scratch/ajauwena/binf_6110/p3/results_freebayes

# Command line arguments:
# $0: ./variant_caller_and_vcf_file_combiner_freebayes.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/burbot_genome (a directory containing the burbot
reference genome).
# $2: /scratch/ajauwena/binf_6110/p3/sorted_bam_files_read_groups (a directory
containing the sorted and reindexed .bam files for the 10 individual fish with read
groups).
# $3: /scratch/ajauwena/binf_6110/p3/results_freebayes (a directory containing the
results obtained from calling variants using Freebayes).
```

```
# Loop through each sorted .bam file in the appropriate directory.
for file in $2/*
do

	# Extract the file's extension name.
	extension_name=$(echo ${file} | rev | cut -d '.' -f1 | rev)

	# If the file's extension name is "bam"...
	if [ ${extension_name} == "bam" ]
	then

		# Append the file to a list.
		echo ${file} >> $3/bam_file_list_freebayes.txt

	fi

done

# Call variants in the files in the list using Freebayes.
freebayes -f $1/burbot_2021.fasta -L $3/bam_file_list_freebayes.txt >
$3/results_freebayes.vcf
```

Appendix 6: "filterer.sh"

```
                                        Unix Shell
# To execute this script on the directory "results_bcftools," run:
./filterer.sh /scratch/ajauwena/binf_6110/p3/results_bcftools

# Command line arguments (when executed on the directory "results_bcftools"):
# $0: ./filterer.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/results_bcftools (a directory containing the
results obtained from calling variants using BCFtools).

# To execute this script on the directory "results_freebayes," run:
./filterer.sh /scratch/ajauwena/binf_6110/p3/results_freebayes

# Command line arguments (when executed on the directory "results_freebayes"):
# $0: ./filterer.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/results_freebayes (a directory containing the
results obtained from calling variants using Freebayes).

# Loop through each file in the appropriate directory.
for file in $1/*
do

	# Extract the file's extension name.
	extension_name=$(echo ${file} | rev | cut -d '.' -f1 | rev)
```

```
	# If the file's extension name is "vcf"...
	if [ ${extension_name} == "vcf" ]
	then

		# Extract the file's base name.
		base_name=$(echo ${file} | rev | cut -d '.' -f2 | cut -d '/' -f1 | rev)

		# Filter the file using reasonable thresholds.
		vcftools --vcf ${file} --minQ 30 --mac 1 --max-maf 0.5 --max-missing 1 --remove-indels --max-alleles 2 --recode --out $1/${base_name}_filtered # (VCFtools, n.d.).

	fi

done
```

Appendix 7: "snp_number_finder.sh"

```
Unix Shell
# To execute this script, run:
./snp_number_finder.sh /scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf /scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vcf /scratch/ajauwena/binf_6110/p3/analyses

# Command line arguments:
# $0: ./snp_number_finder.sh (this script).
# $1: /scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf (a .vcf file containing variants identified using BCFtools, filtered for quality greater than 30).
# $2: /scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vcf (a .vcf file containing variants identified using Freebayes, filtered for quality greater than 30).
# $3: /scratch/ajauwena/binf_6110/p3/analyses (a directory containing files used for analyses).

# Print column headers and write them to a .txt file.
printf "bcftools\tfreebayes\n" > $3/snp_number.txt

# Extract the number of SNPs obtained from calling variants using BCFtools.
snps_bcftools=$(echo ${1} | bcftools stats ${1} | grep "number of SNPs:" | cut -d$'\t' -f4)

# Extract the number of SNPs obtained from calling variants using Freebayes.
```

```
snps_freebayes=$(echo ${2} | bcftools stats ${2} | grep "number of SNPs:" | cut -
d$'\t' -f4)

# Print the numbers of SNPs obtained above and append them to the .txt file.
echo -e "${snps_bcftools}\t${snps_freebayes}" >> $3/snp_number.txt
```

Appendix 8: "bgzipper_and_indexer.sh"

```
                                        Unix Shell
# To execute this script, run:
./bgzipper_and_indexer.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f /scratch/ajauwena/binf_6110/p3/results_bcftools
/scratch/ajauwena/binf_6110/p3/results_freebayes

# Command line arguments:
# $0: ./bgzipper_and_indexer.sh (this script).
# $1:
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
(a .vcf file containing variants identified using BCFtools, filtered for quality
greater than 30).
# $2:
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f (a .vcf file containing variants identified using Freebayes, filtered for quality
greater than 30).
# $3: /scratch/ajauwena/binf_6110/p3/results_bcftools (a directory containing the
results obtained from calling variants using BCFtools).
# $4: /scratch/ajauwena/binf_6110/p3/results_freebayes (a directory containing the
results obtained from calling variants using Freebayes).

# Make a copy of the .vcf file obtained from calling variants using BCFtools for
bgzipping.
cp ${1} $3/results_bcftools_filtered_for_bgzipping.recode.vcf

# bgzip the .vcf file copy.
bgzip $3/results_bcftools_filtered_for_bgzipping.recode.vcf

# Index the bgzipped file.
bcftools index $3/*.gz

# Make a copy of the .vcf file obtained from calling variants using Freebayes for
bgzipping.
cp ${2} $4/results_freebayes_filtered_for_bgzipping.recode.vcf

# bgzip the .vcf file copy.
bgzip $4/results_freebayes_filtered_for_bgzipping.recode.vcf
```

```
# Index the bgzipped file.
bcftools index $4/*.gz
```

Appendix 9: "snp_overlap_finder.sh"

```
                                        Unix Shell
# To execute this script, run:
./snp_overlap_finder.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered_for_bgzippi
ng.recode.vcf.gz
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered_for_bgzip
ping.recode.vcf.gz /scratch/ajauwena/binf_6110/p3/analyses

# Command line arguments:
# $0: ./snp_overlap_finder.sh (this script).
# $1:
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered_for_bgzippi
ng.recode.vcf.gz (a bgzipped .vcf file containing variants identified using BCFtools,
filtered for quality greater than 30).
# $2:
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered_for_bgzip
ping.recode.vcf.gz (a bgzipped .vcf file containing variants identified using
Freebayes, filtered for quality greater than 30).
# $3: /scratch/ajauwena/binf_6110/p3/analyses (a directory containing files used for
analyses).

# Print column headers and write them to a .txt file.
printf
"percentage_snp_bcftools\tpercentage_snp_overlap_over_snp_bcftools\tpercentage_snp_ov
erlap_over_snp_freebayes\tpercentage_snp_freebayes\n" > $3/snp_overlap.txt

# Obtain the SNPs that overlap between the two .vcf files.
snp_overlap=$(vcf-compare ${1} ${2} | grep ^VN | cut -f 2-)

# Obtain the percentage of SNPs identified using BCFtools.
snp_bcftools=$(echo ${snp_overlap} | cut -d '%' -f1 | cut -d '(' -f2)
echo "The percentage of SNPs identified using BCFtools:" ${snp_bcftools}

# Obtain the percentage of SNPs that overlap that were also identified using
BCFtools.
snp_overlap_over_snp_bcftools=$(echo ${snp_overlap} | cut -d '%' -f2 | cut -d '(' -
f2)
echo "The percentage of SNPs that overlap that were also identified using BCFtools:"
${snp_overlap_over_snp_bcftools}
```

```
# Obtain the percentage of SNPs that overlap that were also identified using
Freebayes.
snp_overlap_over_snp_freebayes=$(echo ${snp_overlap} | cut -d '%' -f3 | cut -d '(' -
f2)
echo "The percentage of SNPs that overlap that were also identified using freebayes:"
${snp_overlap_over_snp_freebayes}

# Obtain the percentage of SNPs identified using Freebayes.
snp_freebayes=$(echo ${snp_overlap} | cut -d '%' -f4 | cut -d '(' -f2)
echo "The percentage of SNPs identified using freebayes:" ${snp_freebayes}

# Print the values obtained above and append them to the .txt file.
printf
"${snp_bcftools}\t${snp_overlap_over_snp_bcftools}\t${snp_overlap_over_snp_freebayes}
\t${snp_freebayes}\n" >> $3/snp_overlap.txt
```

Appendix 10: "allele_frequency_and_site_depth_calculator.sh"

```
                                        Unix Shell
# To execute this script, run:
./allele_frequency_and_site_depth_calculator.sh
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f /scratch/ajauwena/binf_6110/p3/analyses

# Command line arguments:
# $0: ./allele_frequency_and_site_depth_calculator.sh (this script).
# $1:
/scratch/ajauwena/binf_6110/p3/results_bcftools/results_bcftools_filtered.recode.vcf
(a .vcf file containing variants identified using BCFtools, filtered for quality
greater than 30).
# $2:
/scratch/ajauwena/binf_6110/p3/results_freebayes/results_freebayes_filtered.recode.vc
f (a .vcf file containing variants identified using Freebayes, filtered for quality
greater than 30).
# $3: /scratch/ajauwena/binf_6110/p3/analyses (a directory containing files used for
analyses).

# Extract the base names of the two inputted .vcf files.
base_name_bcftools=$(echo ${1} | rev | cut -d '.' -f3 | cut -d '/' -f1 | rev)
base_name_freebayes=$(echo ${2} | rev | cut -d '.' -f3 | cut -d '/' -f1 | rev)

# Obtain the minor allele frequency for each .vcf file.
vcftools --vcf ${1} --freq2 --out $3/${base_name_bcftools}_minor_allele_frequency
vcftools --vcf ${2} --freq2 --out $3/${base_name_freebayes}_minor_allele_frequency
# The output files will have the suffix ".frq" (VCFtools, n.d.).
```

```
# Obtain the depth of all snps, summed across individuals, for each .vcf file.
vcftools --vcf ${1} --site-depth --out $3/${base_name_bcftools}_summed_depth
vcftools --vcf ${2} --site-depth --out $3/${base_name_freebayes}_summed_depth
# The output files will have the suffix ".ldepth" (VCFtools, n.d.).

# Obtain the mean depth for each .vcf file, measured in reads per locus per
individual.
vcftools --vcf ${1} --site-mean-depth --out $3/${base_name_bcftools}_mean_depth
vcftools --vcf ${2} --site-mean-depth --out $3/${base_name_freebayes}_mean_depth
# The output files will have the suffix ".ldepth.mean" (VCFtools, n.d.).
```