# final_project_markdown

Eric Lyall and Jake Moersch

2022-03-27

# Introdution

The paper we are reviewing is called "Experimental evolution for niche breadth in bacteriophage T4 highlights the importance of structural genes". It was the doctoral thesis for Jenny Pham at the Department of Organismic and Evolutionary Biology at Harvard University.. This paper anayzed the evolution of T4 bacteriophage, which is a lytic virus that infects E.coli. The purpose of this paper was to gain a better understanding of "niche breadth evolution". Researchers wanted to understand how growing T4 bacteiophage in different niches (different E.coli strains) drives evolution. Specifically, this paper looked at what kind of genes were likely to mutate, and how the niche influence mutations.

Methods: The authors grew T4 bacteriophage (derived from a common ancestor) in 15 cultures of bacteria for approximately 50 generations. Each culture was passages daily. 5 of the bacteria cultures had E.coli C, 5 had E.coli K12, and 5 had alternating populations After 20 days, the author isolated phage population samples from each of the 15 populations and preformed whole genome sequencing on them with a coverage depth of 1000x. Then then compared the mutations from each of the samples.

The goals of our re-analysis are the following: 1) Preform mutation calling on all 15 samples 2) Breakdown the types of mutations that occurred in this dataset, and see if they are similar to the author 3) Look at the relative rates of mutations across 11 functional gene categories and niches (what strain the phage were evolved on). See if these results are similar to the author 4) Graphically compare mutations across different sample populations to look for similar mutation patterns We want to demonstrate that mutation in structural genes drive evolution of the bacteriophage.

We chose this analysis because it is very similar to what one team member (Eric Lyall) will have to for the Tropini Lab this summer. This analysis also makes use of a new mutation-calling tool called breseq, which neither of us have had previous exposure to. We are also completely unfamiliar working with microbial genomes, as we have only done human genomes in this course.

# Methods:

## Downloading the reference genome:

We need to download the reference genome to use as a comparison while searching for mutations. The reference genome can be downloaded from NCBI here: https://www.ncbi.nlm.nih.gov/nuccore/NC_000866 Following the instructions from the breseq website, we first need to make sure the "show sequence" box is selected, so that we download the sequence as well as the features:

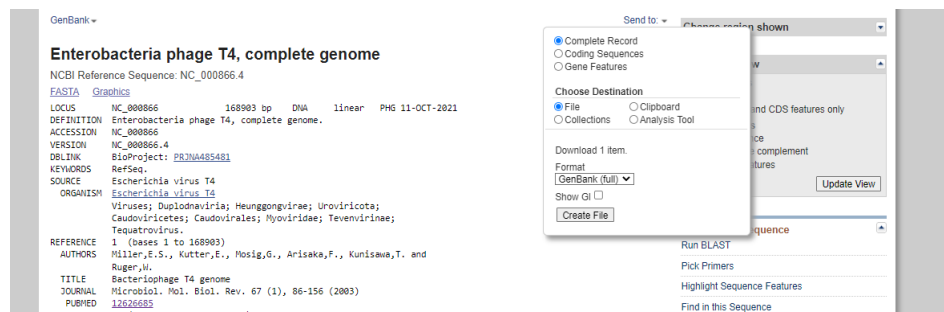At this point, we can click the "send to" button and download the file as a genebank file (.gb)



Figure 1: Accessing reference genome on GeneBank

Using scp, we can trasfer the file over to our cluster:

```
## Warning: package 'formatR' was built under R version 4.1.2
```

After transferring over the file, we need to check to make sure it contains the sequence as well as the features. This can be done by opening the file, and scrolling until you see a header called "Origin". Under this should be a sequence with nucleotides.

```
#cat sequence.gb

#We are able to see the nucleotide seqeunce here!
```

```
                /db_xref="GeneID:1258677"
                /translation="MLAYQARVKEEYDQLMLKINALSKFLESAKFLTVSAVEQELLLS
                QFISMKSYAECLEKRIAQFK"
     gene       complement(167965..168903)
                /gene="rIIB"
                /locus_tag="T4p278"
                /db_xref="GeneID:1258618"
     CDS        complement(167965..168903)
                /gene="rIIB"
                /locus_tag="T4p278"
                /note="membrane integrity protector; mutants give rapid
                lysis on various lysogenic strains due to effects of
                specific prophage products, mistakenly interpreted as
                related to lysis inhibition"
                /codon_start=1
                /transl_table=11
                /product="helix-turn-helix domain-containing protein"
                /protein_id="NP_049889.1"
                /db_xref="GeneID:1258618"
                /translation="MYNIKCLTKNEQAEIVKLYSSGNYTQQELADWQGVSVDTIRRVL
                KNAEEAKRPKVTISGDITVKVNSDAVIAPVAKSDIIWNASKKFISITVDGVTYNATPN
                THSNFQEILNLLVADKLEEAAQKINVRRAVEKYISGDVRIEGGSLFYQNIELRSGLVD
                RILDSMEKGENFEFYFPFLENLLENPSQKAVSRLFDFLVANDIEITEDGYFYAWKVVR
                SNYFDCHSNTFDNSPGKVVKMPRTRVNDDDTQTCSRGLHVCSKSYIRHFGSSTSRVVK
                VKVHPRDVVSIPIDYNDAKMRTCQYEVVEDVTEQFK"
ORIGIN
        1 aattttcctt attaggccgc aagggccttc atagtttag cgatttggga aacttcatca
       61 tcacttaaag agttgcgata accgatgaag tcggaaacaa tacggaattt cttggtaaac
      121 tcagcaacca tttttatcact gttttttgaa gcattatttg ataatacatc aaaaagatta
      181 gttactgtcc aaatgtcatg accgatggta tctttttccac cattaaaata tacaccctgt
```

We have sucessfully downloaded the reference genome!

# Mutation calling pipeline for a single sample:

We will now overview the data analysis pipeline for a single sample (SRR10323947). This consists of 1) downloading the fastq files from SRA

2) Running fastqc on the fastq files

3) Using trimmomatic to clean up are fastq files and remove the illumina adapters.

4) Running fastqc again on the trimmed samples to confirm trimmmatic was sucessful

5) Running breseq to find mutations relative to the T4 bacteriophage reference genome.

After this section, we will develop a series of pipelines that does all of this together.

## Step 1: Downloading the fastq file from SRA:

The first step is to download the sequences for each of the 15 samples from SRA.

```
#using conda to install the sra tools package
#conda install -c bioconda sra-tools

#It's important to add the split files command to get both forwards and reverse reads.
# fastq-dump SRR10323947 --split-files
```

## Step 2: Quality control- Running fastqc on the the fastq file:

We can use fastQC to check the quality of our files. This checks the quality scores of our sequences, and tells us if there is unexpected coverage, adapter content or unknown nucleotides. This needs to be run on both the forwards and reverse reads.

```
#fastqc SRR10323947_1.fastq
```

The read quality drops off significantly near the ends of reads. This means we will have to use trimmomatic to remove some of the poor quality bases near the end of every read. This allows downstream analysis with breseq to only work with high quality data.
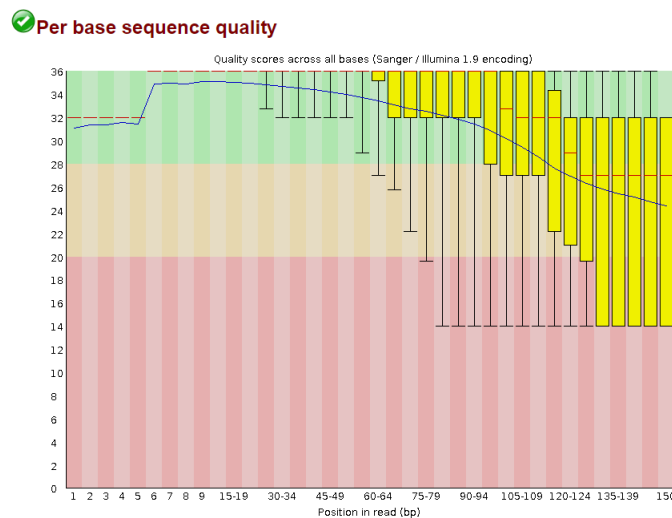


Figure 2: Base quality has significant drop off near the ends of reads

## Step 3: Quality control- Using trimmomatic to filter the fastq file and remove illumina adapters.

Next we can use trimmomatic to trim off the adapter sequences and poor reads near the end. The paper mentions: "Demultiplexed reads were trimmed for Nextera adapter sequences using Trimmomatic with default settings". We also have to trim off the illumina adapters from our reads, as shows in the photo below

```
#conda install -c bioconda trimmomatic

#First we have to make sure the fastqc files we have are in the zipped format-
#this makes it faster for trimmomatic.
#gzip *.fastq

#Next we need to copy the illumina adapters for nextera paired-end sequencing
#into our working directory. These are downloaded with trimmomatic.
#(bmeg400e_env) elyall_bmeg22@orca01:~/anaconda3/pkgs/trimmomatic-0.39-hdfd78af_2/share/trimmomatic/ada
#     cp NexteraPE-PE.fa /home/elyall_bmeg22/final_project/

#Now we can run the trimmomatic commands to remove the illumina adapters.
#The other parameters (sliding window, minlen, 2:40:15) are default parameters
```
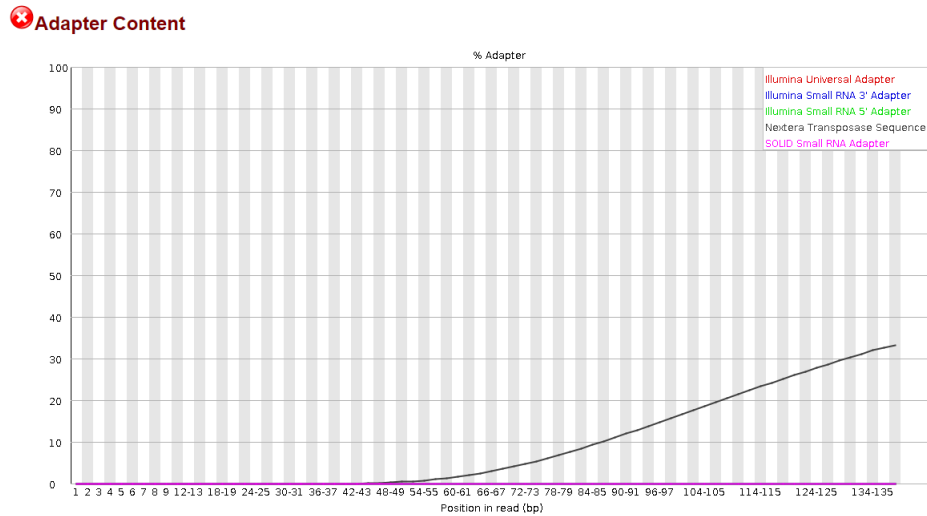
Figure 3: Adapters to get rid of

```
#taken off the trimmomatic documentation, as the study authors did not say what parameters they used.

#trimmomatic PE SRR10323947_1.fastq.gz SRR10323947_2.fastq.gz SRR10323947_1.trim.fastq.gz
#          SRR10323947_1un.trim.fastq.gz SRR10323947_2.trim.fastq.gz SRR10323947_2un.trim.fastq.gz
#          SLIDINGWINDOW:4:20 MINLEN:25 ILLUMINACLIP:NexteraPE-PE.fa:2:40:15

#Below is the output of the trimmomatic command:

#TrimmomaticPE: Started with arguments:
# SRR10323947_1.fastq.gz SRR10323947_2.fastq.gz SRR10323947_1.trim.fastq.gz
#          SRR10323947_1un.trim.fastq.gz SRR10323947_2.trim.fastq.gz
#          SRR10323947_2un.trim.fastq.gz SLIDINGWINDOW:4:20 MINLEN:25
#          ILLUMINACLIP:NexteraPE-PE.fa:2:40:15
#Using PrefixPair: 'AGATGTGTATAAGAGACAG' and 'AGATGTGTATAAGAGACAG'
#Using Long Clipping Sequence: 'GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG'
#Using Long Clipping Sequence: 'TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG'
#Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTCCGAGCCCACGAGAC'
#Using Long Clipping Sequence: 'CTGTCTCTTATACACATCTGACGCTGCCGACGA'
#ILLUMINACLIP: Using 1 prefix pairs, 4 forward/reverse sequences,
#          0 forward only sequences, 0 reverse only sequences
#Quality encoding detected as phred33
#Input Read Pairs: 1086157 Both Surviving: 670281 (61.71%)
#          Forward Only Surviving: 393675 (36.24%)
#          Reverse Only Surviving: 13161 (1.21%)
#          Dropped: 9040 (0.83%)
#TrimmomaticPE: Completed successfully
```

## Step 4: Quality control- running an additional fastqc to confirm the adapter sequences are removed:

We now need to run fastq again on our trimmed sample to confirmed if the adapter contents are actually removed.

5

```
#fastqc SRR10323947_1.trim.fastq.gz
```

You can see the the nextera adapter content is now removed. Since the authors did not make any additional adjustments, neither will we.
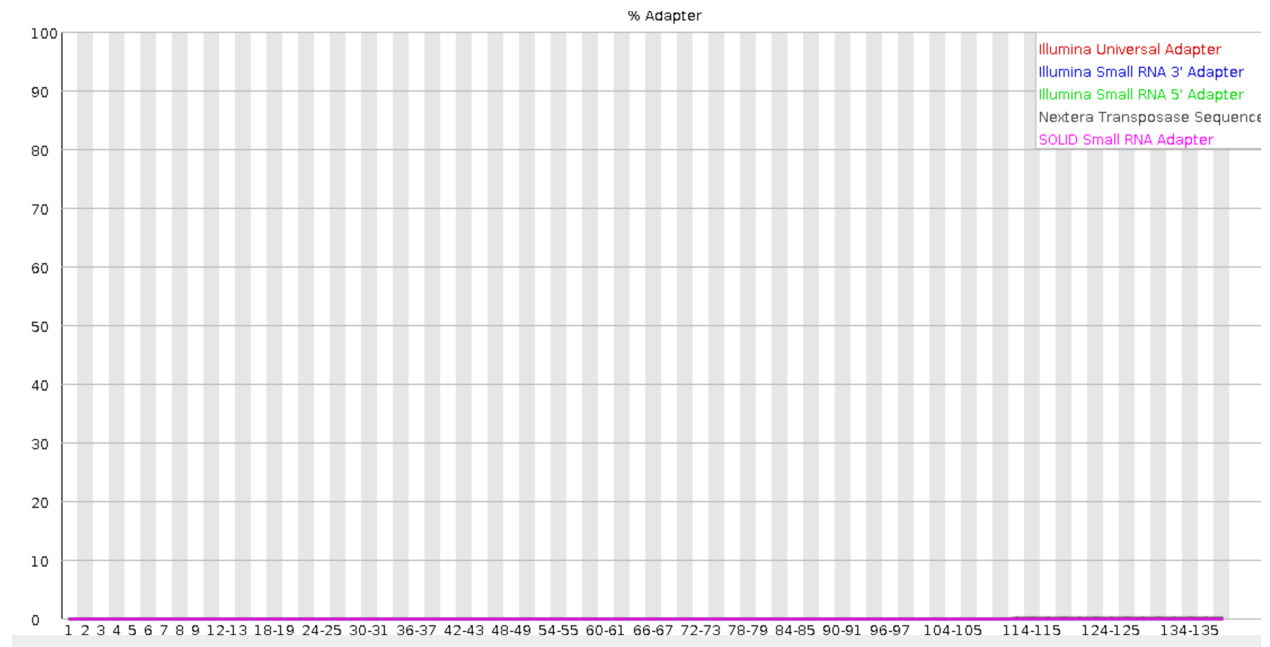


## Step 5: Running breseq to call mutations on our sample

The next step is to download and run breseq. breseq is a pipeline that is best for calling mutations on small microbial genomes. It starts by aligning the reads to a reference sequence, and then looks for consistent mismatches. Breseq can be run in "population mode" with the -p flag. This is used when a heterogeneous population is sequences, and one wants to find all of the possible mutations against a reference sequence. If greater than 5% of the reads at a given position mismatch, breseq reports a mutation at this position. breseq will report all of the mutations, their frequencies and positions. It also looks at the reading frame and determines if mutations are synonymous or non-synonymous, and identified deletions, insertions and SNP's. Using the genome annotation from the reference sequence, breseq will report what gene the mutation occurs in and the function of that gene if it's available. These features make breseq and incredibly useful tool for analyzing mutations in microbial sequences.

```
#Breseq requires bowtie and samtools to be installed first.

#conda -c bioconda install bowtie
#conda -c bioconda install samtools

#Now installing breseq
#conda -c bioconda install breseq

#ARunning breseq on a single sample,
#where seqeunce.gb is our reference T4 bacteriophage sequene.
# breseq -p -r sequence.gb -o /home/elyall_bmeg22/final_project/paired/trimmed_files/breseq_run/
#        SRR10323947_1.trim.fastq.gz SRR10323947_2.trim.fastq.gz
```

Let's look at some of the output files of our breseq data!

Breseq gives us summary information for each sample. This lets us know that most of our read are mapped. Breseq requires 90% of a reads length to map.

## Read File Information

| | read file | reads | bases | passed filters | average | longest | mapped |
|---|---|---|---|---|---|---|---|
| errors | SRR10323947_1.trim | 652,699 | 90,923,277 | 100.0% | 139.3 bases | 150 bases | 98.0% |
| errors | SRR10323947_2.trim | 652,003 | 88,032,389 | 99.9% | 135.0 bases | 150 bases | 97.5% |
| | **total** | **1,304,702** | **178,955,666** | 99.9% | 137.2 bases | **150** bases | 97.7% |

Figure 4: breseq summary information

Breseq also give a list of mutation predictions for each sample:



Figure 5: breseq mutation predictions

If you click on the evidence, it will show you all of the reads mapping to that position, and highlight the exptcted mutation.

We can see our coverage is close to 1000 base pairs on average (which is what the authors aimed to sequnce at).

# Creating pipeline to preform analysis on all 15 samples:

This section contains a combination of pipelines that will ultimately run mutation calling on all 15 of our samples

## Downloading all of the SRA names from NIH:

Making a textfile containing all of the SRA names

```
#First we have to download the acession list as a txt file form NCBI
#https://www.ncbi.nlm.nih.gov/sra?term=SRP226618

#creating a .txt file containing all of the SRA #'s for each sample:
#nano SRA_seqs.txt
#We then copy paste all of the SRA #'s names into this file
```

Figure 6: breseq alignment evidence

## Creating a pipeline that downloads fastq files and runs fastQC on them

Now, we are going to generate a pipeline that downloads the fastq files and runs fastQC on each of these. Normally we would use a job scheduler to do this, but apparently our server can't do this. So we will used Dr. De Boer's script which runs a script for each line in an input file The script is here: (https://raw.githubusercontent.com/BMEGGenoInfo/Assignments/main/Assignment_2/runTheseJobsSerially.sh

```
#Downloading Carl's script:

#wget
#https://raw.githubusercontent.com/BMEGGenoInfo/Assignments/main/Assignment_2/runTheseJobsSerially.sh

#Getting permissions to run it:
#chmod +x runTheseJobsSerially.sh

#Creating a pipeline called down_trim_pipeline.sh
#that downloads files from NCBI,
#runs fastQC and then runs trimmomatic on them.
#First we make the name of the pipeline:
#nano down_trim_pipeline.sh

#Then we get permission on it
#chmod +x down_trim_pipeline.sh

#Then we run it!
#./runTheseJobsSerially.sh ./pipelines/down_trim_pipeline.sh SRA_seqs.txt

#The pipeline is shown below:

#!/bin/bash
#set -e # this makes the whole script exit on any error.
#sample=$1
#mkdir -p fastq_samples # making a directory to add the fastq files
#mkdir -p fastqc_files #making a directory for fastqc_files
```

## Coverage Distribution at Unique−Only Positions
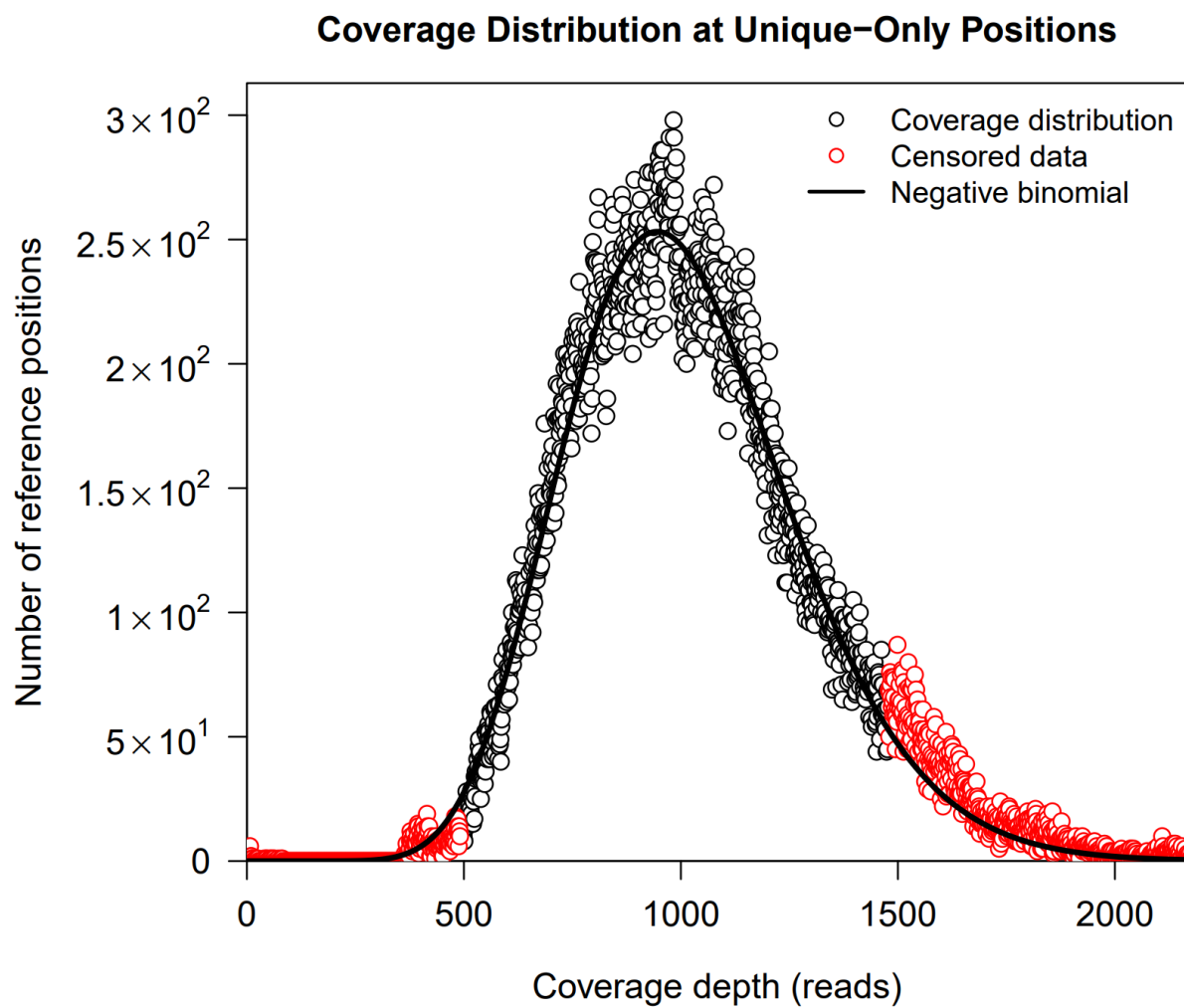


Figure 7: breseq coverage example

```
#mkdir -p downtrim_logfiles
##Now, going to run the pipeline:

#echo running pipeline for $sample
#if [ ! -e downtrim_logfiles/$sample.fastq.done ]
#then
#        echo Downloading $sample
#       #Command download sample:
#        fastq-dump $sample --split-files --outdir fastq_samples/
#        touch downtrim_logfiles/$sample.fastq.done #add a flag to say fastq downloaded.
#else
#        echo Already performed fastqc of $sample
#        fi
#if [ -e downtrim_logfiles/$sample.fastq.done ] #if the fastq is done,do fastqc
#then
#        echo Running fastqc with $sample
#        mkdir fastqc_files/$sample
#        fastqc $fastq_samples/{sample}_1.fastq --outdir fastqc_files/$sample/
#        fastqc $fastq_samples/{sample}_2.fastq --outdir fastqc_files/$sample/
#        touch downtrim_logfiles/$sample.fastqc.done
#else
#        echo: FastQC was not made sucessfully.
#        fi
```

## Making a trimmomatic pipeline for all samples

We found no additional concerns beyond the ones mention with our pilot sample in the fastQC. Before running a trimmomatic pipeline on all of the samples, we have to gzip the files.

```
#gzip fastq_samples/*.fastq

#Making the name of the trimmomatic pipeline:

#nano pipelines/trimmomatic.sh
#chmod +x trimmomatic.sh

#running the pipeline:
#./runTheseJobsSerially.sh ./pipelines/trimmomatic.sh SRA_seqs.txt
```

Below is the code for our trimmomatic pipeline

```
#!/bin/bash
#set -e # this makes the whole script exit on any error.
#sample=$1
#mkdir -p trimmomatic_files
#if [ ! -e downtrim_logfiles/$sample.trim.done ]
#then
#        mkdir trimmomatic_files/$sample
#         trimmomatic PE $fastq_samples/${sample}_1.fastq.gz fastq_samples/${sample}_2.fastq.gz
#         trimmomatic_files/$sample/${sample}_1.trim.fastq.gz
#         trimmomatic_files/$sample/${sample}_1un.trim.fastq.gz
```

```
#           trimmomatic_files/$sample/${sample}_2.trim.fastq.gz
#           trimmomatic_files/$sample/${sample}_2un.trim.fastq.gz
#           SLIDINGWINDOW:4:15 MINLEN:25 LEADING:3 TRAILING:3
#           ILLUMINACLIP:NexteraPE-PE.fa:2:40:15


#           touch downtrim_logfiles/$sample.trimmed.done
#else
#           echo:trimmomatic failed
#           fi
```

## Making a pipeline to run breseq

This pipeline could have been attached to the trimmomatic pipeline, but we prefered to run fastQC on some
of the trimmed samples to ensure the trimming is successful. breseq mutation calling for each sample (1000x
coverage, 167k bp) takes 40 minutes.

```
#Now we make a pipeline to run breseq:
# nano pipelines/breseq_pipe.sh
# chmod +x breseq_pipe.sh
# #Anddd running the pipeline
# ./runTheseJobsSerially.sh ./pipelines/breseq_pipe.sh SRA_seqs.txt
#
# #The pipeline starts below:
#
# #!/bin/bash
# set -e # this makes the whole script exit on any error.
# sample=$1
# mkdir -p breseq_files
# if [ -e downtrim_logfiles/$sample.trimmed.done ]
# then
#           #Going to run breseq on it
#           mkdir -p breseq_files/$sample
#           #Running the breseq command:
#           # breseq -p -r sequence.gb -o
#                   /home/elyall_bmeg22/final_project/breseq_files/$sample/
#               trimmomatic_files/$sample/${sample}_1.trim.fastq.gz
#               trimmomatic_files/$sample/${sample}_2.trim.fastq.gz
#           touch downtrim_logfiles/$sample.breseq.done
# else
#           echo:breseq failed
#           fi
```

# breseq mutation calling has finished!

Typically at this point we would run a single line of breseq code that would filter all the mutations we found
against the ancestral sequence.

This line would be: gdtools APPLY –f GENBANK –o updated.gbk –r reference.gbk ancesntral_input_sample.gd
This takes makes a new annotated genome (gbk file) contianing all the mutations that are found in the
reference sequence. Future samples would be compared against this updated gbk file using the gdtools
ANNOTATE or COMPARE functions.

However, the author's didn't actually share the data for the ancestral sample, which is supposed to be compared against the experimental sample, and have common mutations filtered out. We emailed the author for these files but nothing came. This is a bit of an issue for us- keeping all of these mutations in, most of which are from the ancestor, makes it difficult compare data against the paper, and focus on the mutations which are important. As a proxy for filtering against the ancestral sequence, we are going to filter against the final list of mutations the authors have provided in their supplementary table.

We discussed these changes with Carl and he said we would not lose difficultly score points here. In our conclusion, we will compare our results with those of the paper.

Another piece of information we will need is which E.Coli population each T4 sample evolved on (E.coli C, E.coli K12, or mixed.) This detail can be found by looking at the sample descriptions on the NCBI https://www.ncbi.nlm.nih.gov/sra/?term=PRJNA578899 The SRA # number is matched to the experimental name and population below:

SRR 61 = 55D18R1 # E coli K12 replicate 1

SRR 60 = 55D18R2 #E coli K12 replicate 2

SRR 59 = CD18R1 #E coli C replicate 1

SRR 58 = CD18R2 #E coli C replicate 2

SRR 57 = CD18R3 #E coli C replicate 3

SRR 56 = CD18R4 #E coli C replicate 4

SRR 55 = CD18R5 #E coli C replicate 5

SRR 54 = 55D18R3 # E coli K12 replicate 3

SRR 53 = 55D18R4 # # E coli K12 replicate 4

SRR 52 = 55D18R5 # E coli K12 replicate 5

SRR 51 = 55CD18R1 # alternating e.coli C and K12 replicate 1

SRR 50 = 55CD18R2 # alternating e.coli C and K12 replicate 2

SRR 49 =55CD18 R3 # alternating e.coli C and K12 replicate 3

SRR 48 = 55CD18R4 # alternating e.coli C and K12 replicate 4

SRR 47 = 55CD18R5 # alternating e.coli C and K12 replicate 5

# Annotating our mutations & filtering against the supplementary table:

To get more detail on our mutations, we can used the ANNOTATE function from breseq's gdtools. The ANNOTATE tool acts on .gd files. .gd files are tab-delineated files describing differences between a reference genome and the tested samples. Currently, our .gd files only contain detail on what gene was mutated, the base piar change and the positions. We need more detail for our analysis.
The ANNOTATE tool creates an annotated file that adds detail to each mutation found, including the type (e.g SNP), and whether is synonymous or non-synonymous. The ANNOTATE function used the reading frame from the reference equence and determines if the amino acid changes based on nucleotide changes. We can save this data in a tsv file, where each line contains a mutation from a sample.

```
#gdtools ANNOTATE -o annotated_all.tsv -f TSV -r reference_seq.gb 47_output.gd
#48_output.gd 49_output.gd 50_output.gd 51_output.gd 52_output.gd 53_output.gd
#54_output.gd 55_output.gd 56_output.gd 57_output.gd 58_output.gd
```

```r
#59_output.gd 60_output.gd 61_output.gd

#Uploading annotated mutations from a tsv file::
all_ann <- as.data.frame(read.table (file = 'annotated_all.tsv', sep = '\t', header = TRUE))

#loading up the supplementary table:
library(openxlsx)
```

```
## Warning: package 'openxlsx' was built under R version 4.1.3
```

```r
mutation_xl <- read.xlsx("Supplementary table 2a & 2b.xlsx",sheet = 1)
mutation_df <- as.data.frame(mutation_xl)

#Filtering out the mutation in all_ann that are not in the supplementary files.
#This is our stand-in for not having the ancestral sequence.
#This filtering is done by postion- if there'a mutation called by our team that does not share the posi
#of mutations shown in the paper, we deleted it.
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.3
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5      v purrr   0.3.4
## v tibble  3.1.6      v dplyr   1.0.8
## v tidyr   1.2.0      v stringr 1.4.0
## v readr   2.1.2      v forcats 0.5.1
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## Warning: package 'tidyr' was built under R version 4.1.3
```

```
## Warning: package 'readr' was built under R version 4.1.3
```

```
## Warning: package 'purrr' was built under R version 4.1.2
```

```
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
## Warning: package 'stringr' was built under R version 4.1.2
```

```
## Warning: package 'forcats' was built under R version 4.1.3
```

```
## -- Conflicts ------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
filtered_all_ann <- filter(all_ann, position %in% mutation_df$position )
nrow(filtered_all_ann)
```

## [1] 190

In total we have 192 surviving mutations, compared to the 174 reported in the paper. This means our analysis is largely successful! Many of the mutations we found were also found in the paper.

This is much better than what we had before filtering, which was 3197 mutations (most of which are probably from the ancestor)

This means many of the mutations were shared between our data and the paper's data. We may have more mutations because the there could be mutations that we have (ancestral or otherwise) which share the same position. As we have no way of distinguishing these, we will leave them in for our analysis

# Annotating our mutation list with functional gene categories

Now we are going to further annotate our mutation list with the gene functional categories. The authors of this paper took functional gene categories from a paper by Miller et al (the same paper that made the latest version of the T4 genome. We've downloaded these functional gene categories into an excel file, added it to the supplementary table and will use them for annotation. The purpose of annotating genes by functional category is to make data visualization eaiser. If we only have the names of specific genes, it's difficult to get a picture of what the broad themes were during the evolution of these phage. By annotating our mutations with functional categories, we are eventually able to present strong evidence that the mutations in structural viron proteins drive the niche evolution of bacteriophage.

```
#First we have to upload each functional category and their associated genes.
#These designations are taken from the a paper by Miller et al, which was used in this study.
#The "functional designations" are: Transcription, Translation,
#Nucleotide_metabolism, DNA_rep, Virion_prots, chaperonins, lysis,
#host_phage_int host_alt, homing_endonuclease, predicted_integral_membrane
#Any gene that doesn't fit in one of these categories is classified as unknown.

library(openxlsx)
func_des <- read.xlsx("Supplementary table 2a & 2b.xlsx",sheet = 'melt_cat_genes')
func_des_df <- as.data.frame(func_des)
#functional designation dataframe contains functional categories and their genes.

#Now, we can annotate all of our mutation with their matching functional designation.
#If no functional designation is found, we will label it as unknown

#We are going to iterate through each gene from our mutation data, and assign a gene designation to it:
all_genes <- filtered_all_ann$gene_name
#geting a list of all the genes in our annotated mutation calls from each sample
all_genes_func <- c() #Creating an empty vector where associated functional categories will be added
for(gene in all_genes)
{
        r <- which(func_des_df$gene == gene)
        #getting a list of rows where the mutated gene is associated iwth a functional category
        if(length(r)==0){ #if we can't find a funcional category associated with the gene
                all_genes_func <- c(all_genes_func,"unknown") #Say the gene function is unknown
        }
```

14

```
        else{
                all_genes_func <- c(all_genes_func,func_des_df$function_cat[r])
                #otherwise, add the functional category
        }
}

#Adding the vector containing all the gene functional designations to our datframe
filtered_all_ann$gene_function <- all_genes_func
```

# Calculating mutations per functional category:

Now we can start looking at mutations as they relate to functional categories. As with the paper, mutations are dominated by those in the virion protein category. This makes sense- modifying virion proteins is one of the easier ways for phage to increase their infection efficiency over a long period of evolution. Viron proteins (like tail fibers) facilitate entry into E.coli, which is required for phage replication and survival.

```
#Finding the number of mutations per functional category:
library(tibble)
library(ggplot2)
mut_per_func <- filtered_all_ann %>% group_by(gene_function) %>% summarise(n=n()) %>% arrange(desc(n))
ggplot(mut_per_func, aes(x="",y=n,fill = gene_function))+ geom_bar(width = 1, stat = "identity")+
  ylab("Number of mutations")+ xlab("All mutation frequencies")
```
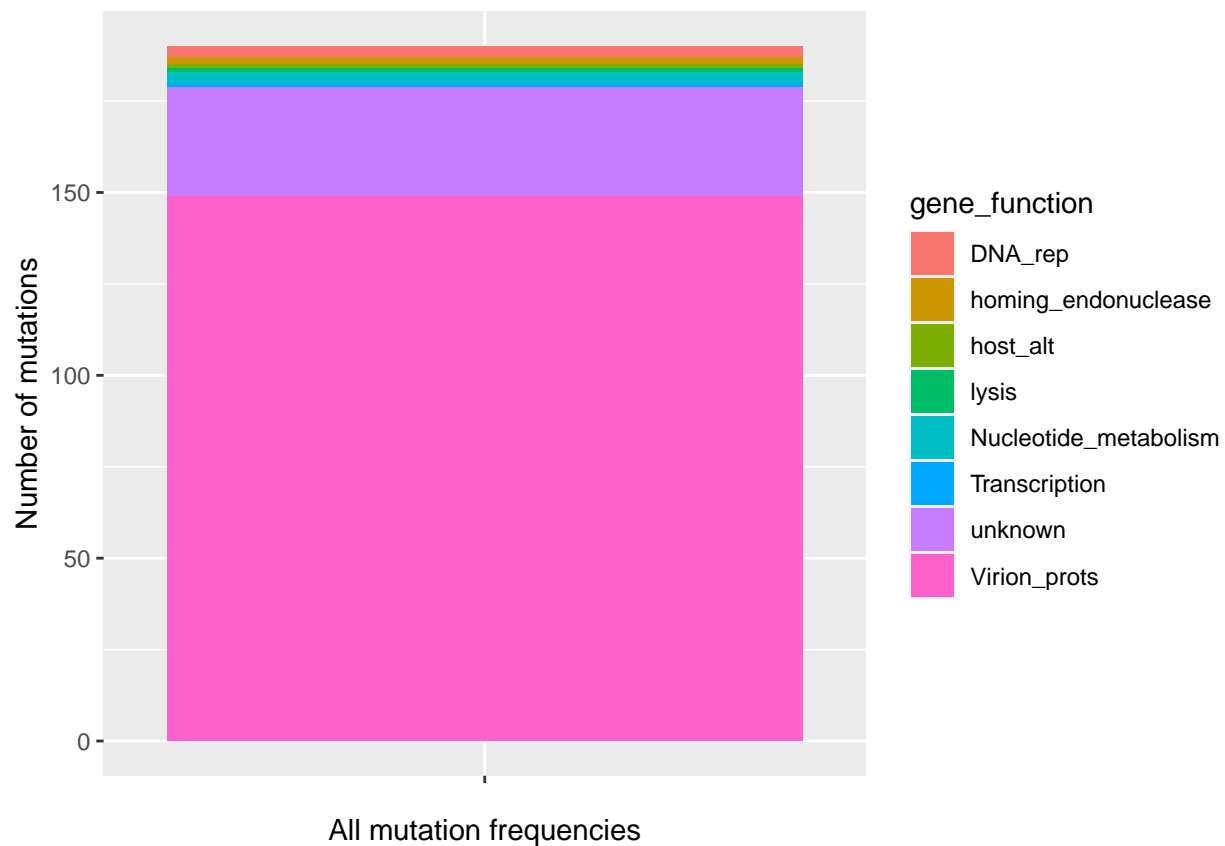
```
#Finding the number of higher-frequency mutations (greater than 50%) per functional category:
high_freq <- filtered_all_ann %>% filter(frequency > .5)
high_f_mut_func <- high_freq %>% group_by(gene_function) %>% summarise(n=n()) %>% arrange(desc(n))
ggplot(high_f_mut_func, aes(x="",y=n,fill = gene_function))+
  geom_bar(width = 1, stat = "identity") + ylab("Number of mutations") +
  xlab("Mutations greater than 50% frequency")
```
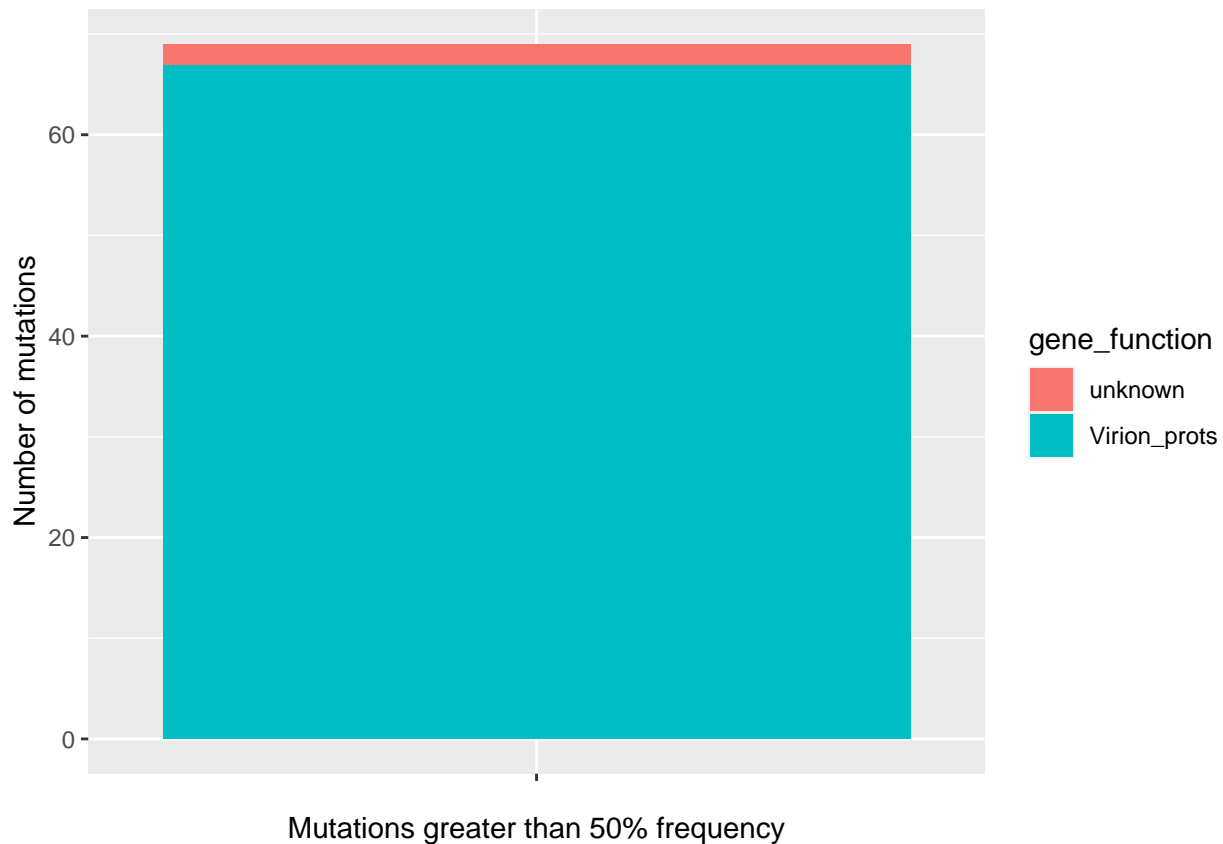


## Comparing mutations by type

Next we will get a breakdown of what kinds of mutations occured in our dataset. From the annotated mutation file, we get information on whether the mutation was an indel, SNP or substitution. We also get details on the impact of these mutations (whether they are synonymous, non-synonymous, non-sense or intergenic). The purpose of this is twofold. First, it acts as a sanity check- most of the mutations reported should be SNP's, as these are much less likely to result in a dysfuctional bacteriophage. Second, this breakdown will give us a grasp on what kind of mutations are in the dataset, aiding downstream analysis.

```
library(dplyr)
library(ggplot2)

#Getting the total number of mutations
nrow(filtered_all_ann)
```

```
## [1] 190
```

```
#There are 190 mutations in total

x = data.frame(table(filtered_all_ann$type))
names(x)[names(x) == "Var1"] <- "Mutation_Type"
names(x)[names(x) == "Freq"] <- "Frequency"
p = ggplot(data=x, aes(x=Mutation_Type, y=Frequency, fill = Mutation_Type)) +
  geom_bar(stat="identity")+
  geom_text(aes(label=Frequency), vjust=-0.3, size=3.5)+
  theme_minimal()

p
```
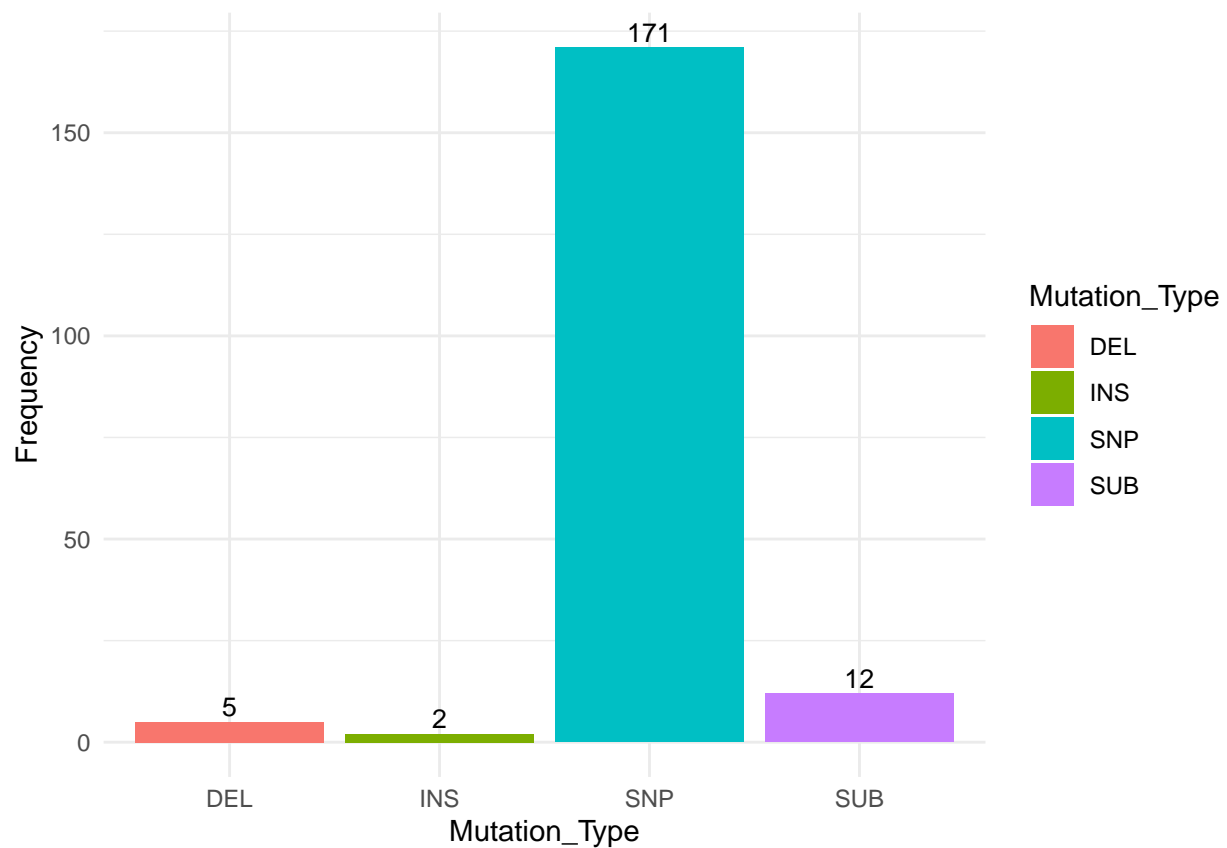


```
#Looking at specifically the SNP mutations to find out how many are synonymous,
#and how many are non-synonymous, and how many are intergenic
filtered_all_ann_noNonSNP <- filtered_all_ann[filtered_all_ann$type == 'SNP',]
y = data.frame(table(filtered_all_ann_noNonSNP$snp_type))
names(y)[names(y) == "Var1"] <- "SNP_Type"
names(y)[names(y) == "Freq"] <- "Frequency"
p = ggplot(data=y, aes(x=SNP_Type, y=Frequency, fill = SNP_Type)) +
  geom_bar(stat="identity")+
  geom_text(aes(label=Frequency), vjust=-0.3, size=3.5)+
  theme_minimal()

p
```
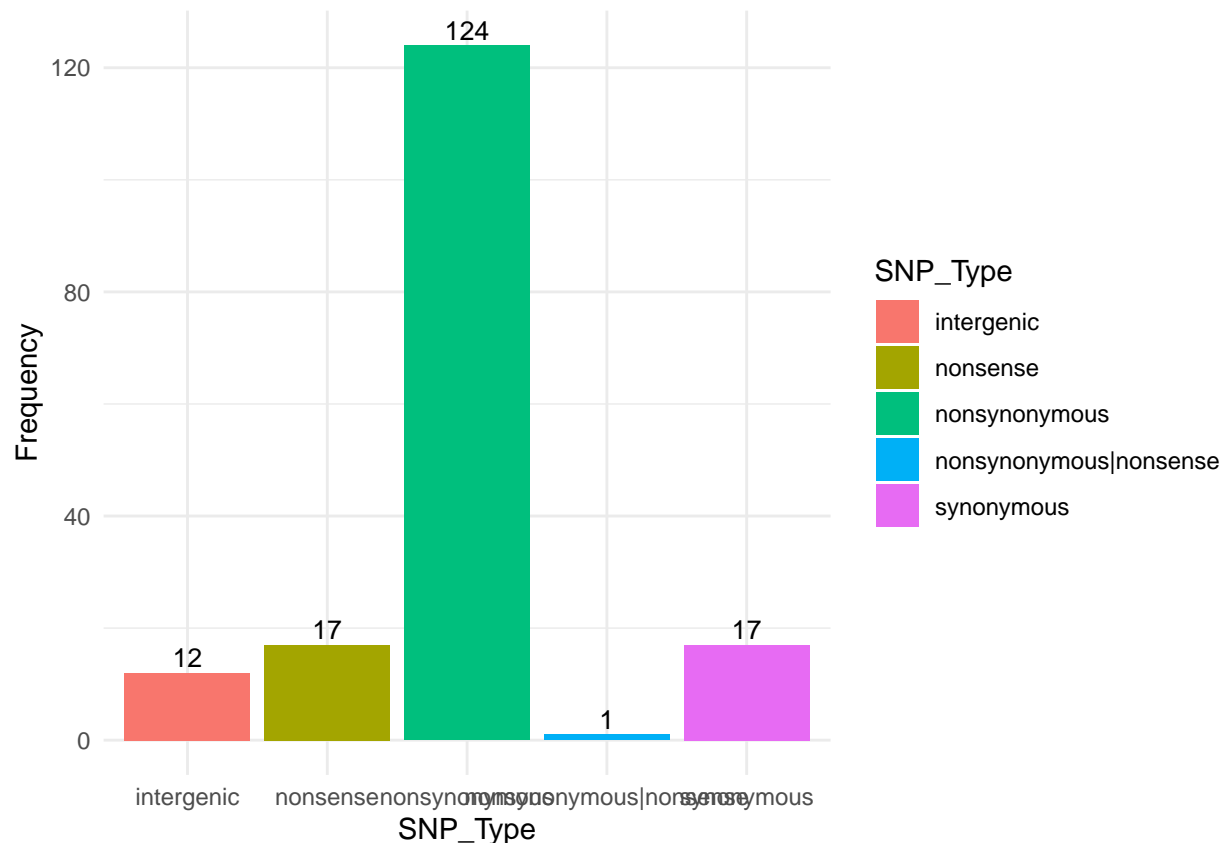
# Finding the number of mutations that are fixed and not fixed.

Next we will analyze the number of mutations that are fixed and not fixed. A fixed mutation will be defined at over 80% prevelance in the population - meaning these mutations were either essential, or associated with other essential mutations that gave the bacterophage a competitiva advantage while evolved. Non-fixed mtuations will be defined as mutations with a population frequency from 5-80%. These mutations occured, but their consequences are not strong enough to either a) vanish entirely by being out competed or b) become fully ingrained in the population (fixed). The pupose of doing this analysis is to see whether or not selection occured on the evolved populations. If selection occurs, we should see two things: 1) There should be a portion of mutations that become fixed in the population 2) The mutations that do become fixed should be mostly non-synonymous, resulting in an amino acid change. It would make little sense if the fixed mutations were all silent synonymous mutations, because these should not exert evolutionary pressure. As you can see in the graphs below, the majority of SNP mutations were non-synonymous, particulary in the fixed mutation category. This indicates selection occurred during the experiment. These results are in line with the reported results from the paper.

```
# trying to generate some graphs by fixed/non-fixed status, as in the paper
library(ggplot2)
library(dplyr)
# add bacterial population labels
b_pop_dict <- c("47_output" = 'Alternating C & K12', "48_output" = 'Alternating C & K12',
                "49_output" = 'Alternating C & K12', "50_output" ='Alternating C & K12',
                "51_output" = 'Alternating C & K12', "52_output"= 'E.coli K12',
                "53_output" = 'E.coli K12', "54_output" = 'E.coli K12',
```

```r
                "55_output" ='E.coli C' , "56_output" ='E.coli C' ,
                "57_output"= 'E.coli C', "58_output"= 'E.coli C' ,
                "59_output"= 'E.coli C' , "60_output" = 'E.coli K12', "61_output" ='E.coli K12' )
population_y_axis_vals <- c()
output_names <- filtered_all_ann$title
for(output_name in output_names){
  y_val <- b_pop_dict[output_name]
  population_y_axis_vals <- c(population_y_axis_vals,y_val)
}
filtered_all_ann$population_name <- population_y_axis_vals #Adding this to our mutation dataframe

summary_stat = filtered_all_ann%>%
  select(population_name,mutation_category,frequency)
#making summary tables
fixed <- summary_stat %>% filter(frequency > 0.8)
fixed_summary <- fixed %>% group_by(mutation_category,population_name) %>%
  summarise(n=n()) %>% arrange(desc(n))
```

```
## 'summarise()' has grouped output by 'mutation_category'. You can override using
## the '.groups' argument.
```

```r
observed <- summary_stat %>% filter(frequency < 0.8)
observed_summary <- observed %>% group_by(mutation_category,population_name) %>%
  summarise(n=n()) %>% arrange(desc(n))
```

```
## 'summarise()' has grouped output by 'mutation_category'. You can override using
## the '.groups' argument.
```

```r
#plotting
p = ggplot(fixed_summary, aes(x="",y = n, fill = mutation_category)) +
  geom_bar(width = 1, stat = "identity") + ylab("Number of mutations") +
  xlab("Mutations greater than 80% frequency")
p + facet_grid(cols = vars(population_name))
```
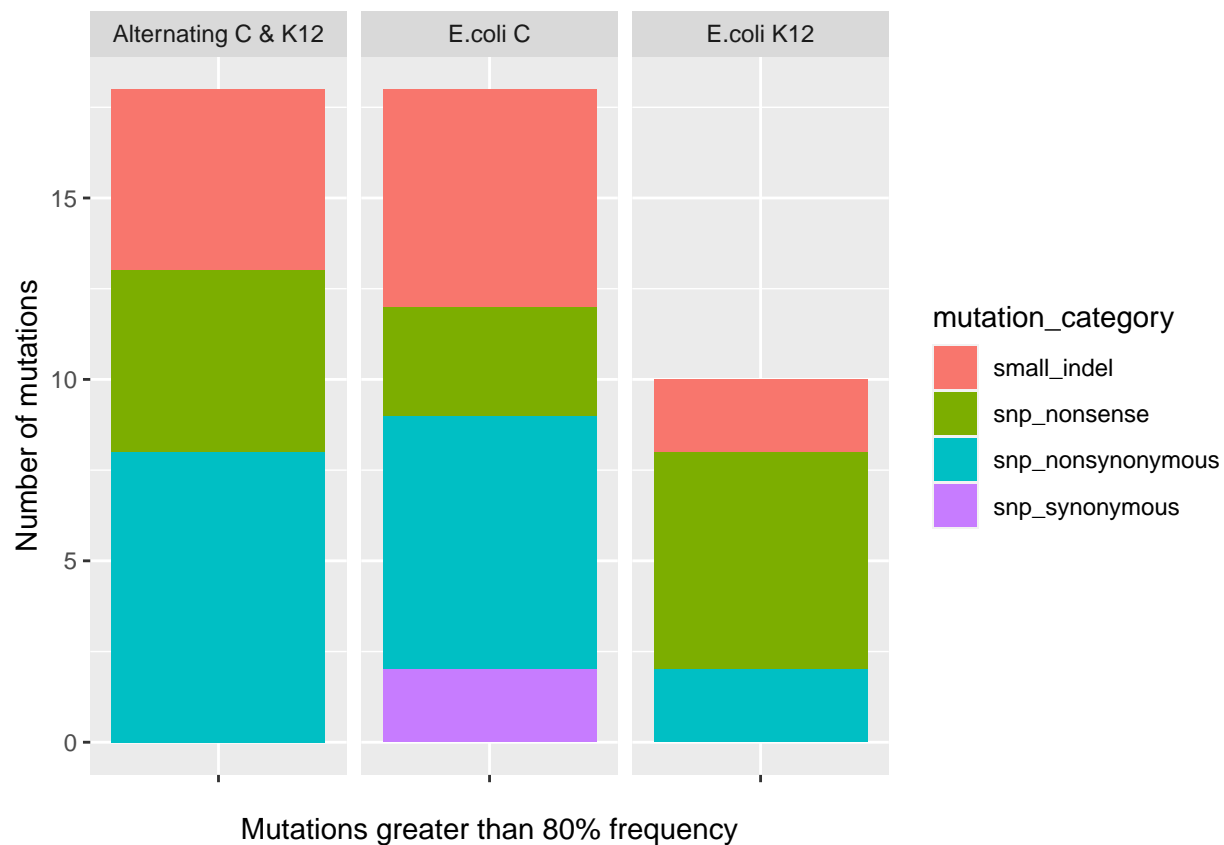
```
p = ggplot(observed_summary, aes(x="",y = n, fill = mutation_category)) +
  geom_bar(width = 1, stat = "identity") + ylab("Number of mutations") +
  xlab("Mutations between 5% and 80% frequency")
p + facet_grid(cols = vars(population_name))
```

Mutations between 5% and 80% frequency
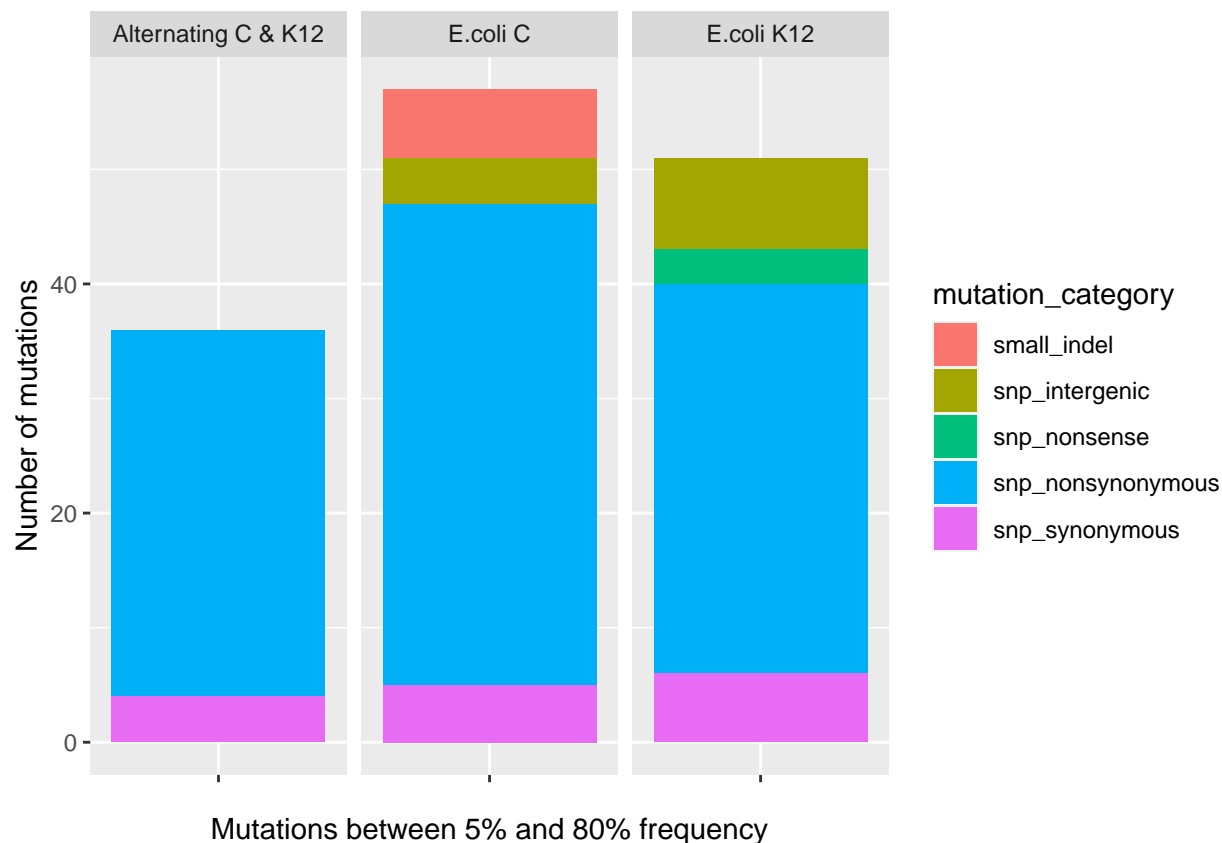
# Calculating relative mutation rates part #1:

The next task is to compare the relative mutation rates across each functional category. A mutation rate is defined as the # of mutations / size of functional category (in BP). The paper used functional categories taken from the 2003 Miller et al. paper. We downloaded a list of T4 genes and their lengths in BP, in addition to the list we already have of genes that make up the functional categories from Miller's paper. We first need to find the size of each functional category in base pairs. This size will allow us to calculate the mutation rate per functional category.

```
#Downloading the a list of T4 genes and their sizes from the Miller et al. paper into a dataframe
t4_genes <- read.xlsx("Supplementary table 2a & 2b.xlsx",sheet = 'T4_genes')

#we are going to add a new column to the dataframe func_des_df that contains the gene length.
#Some genes have multiple exons- in this case we will add them up
gene_length <- c()

#Iterating through all of the genes in our functional categories from Miller et al.
for(gene in func_des_df$gene){
        r <- which(t4_genes$gene == gene) #getting a list of rows with the gene name
        #If the gene isn't found, append a length of zero
        #and print the gene for a manual check to see if it truly does not exist
        if(length(r)==0){
                gene_length <- c(gene_length,0)
```

```
        }
        else{ #If the gene is found
                #There may be multiple gene exons with the same name, will add lengths
                len <- 0
                for(i in r){
                        len <- len + t4_genes$length[i]
                }
                gene_length <- c(gene_length,len) #Append the total gene length to vector
        }
}
#Adding this vector to the functional designation dataframe
func_des_df$gene_length <- gene_length
#Now, we have a dataframe with three colums: functional category, gene, and gene length
```

Now we can get the mutation rate for each sample across each functional category. This will allow us to preform an anova to find the effect of functional category and sample population (E.coli C, E.coli K12, alternating) of the mutation rate

# Creating a function to run a non-parametric ANOVA and Dunn's test

Here we made a function that runs an anova comparing mutation rates across functional gene categories and sample populations. This will tell us if certain factors a) the functional category, b) the bacteria strain phage evolved on or c) an interaction between these affects how likely a sample is to mutate. We used a non-parametric ANOVA because the data is not normally distributed, with some small sample sizes in specific functional categories. After running the ANOVA, we will set up a function to run Dunn's test. This will do a pairwise comparison of mutation rates for every combination of functional categories. As you'll see, only pairs containing virion proteins have a significant result. Ultimately, the ANOVA will show that only functional categories make a difference. The Dunn test will show that viron proteins are responsible for this difference in mutation rates.

```
library(rcompanion)
```

```
## Warning: package 'rcompanion' was built under R version 4.1.3
```

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.1.2
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(vctrs)
```

```
## Warning: package 'vctrs' was built under R version 4.1.2
```

```
##
## Attaching package: 'vctrs'


## The following object is masked from 'package:dplyr':
##
##      data_frame


## The following object is masked from 'package:tibble':
##
##      data_frame
```

```r
library(dplyr)

SRHanova <- function(filtered_all_ann, func_des_df) {

#We will start by making a dataframe where each column is a fucntional category,
  #each row is a sample and values are # of mutations:

#Creating an empty dataframe no # mutations per functional category.
  #There are 11 funcitonal categories + 1 category for unknown
  #+ 1 category for what population the sample is, and 15 E.coli samples.
n_mut_per_fcn_cat_df <- data.frame(matrix(ncol =length(unique(func_des_df$function_cat))+1,
                                          nrow = length(unique(filtered_all_ann$title))) )

#Assigning each functional category to the column names
colnames(n_mut_per_fcn_cat_df) <- c(unique(func_des_df$function_cat),"unknown")

#Assinging each sample to the rownames:
rownames(n_mut_per_fcn_cat_df) <- c(unique(filtered_all_ann$title))

#Adding a column for the sample populations
#Filling this dataframe with 0's as a default:
n_mut_per_fcn_cat_df[is.na(n_mut_per_fcn_cat_df)] <- 0

#We only need to work with sample names & gene functions here, so we can remove the other columns:
name_gene_fcn_df <- filtered_all_ann[c("title","gene_function")]

#Getting a summary of the # of mutations per each functional category in our experiment
fcn_breakdown <- name_gene_fcn_df  %>% group_by(title,gene_function) %>% summarise(n=n())

#Adding this data into the n_mut_per_fcn_cat_df
for(r in 1:nrow(fcn_breakdown)){
        n_mut_per_fcn_cat_df[fcn_breakdown$title[r],fcn_breakdown$gene_function[r]] <-
          fcn_breakdown$n[r]
}

#Then, we will normalize each column to the number of nucleotides in each functional category:

#Getting a list of the category names:
cat_names <- colnames(n_mut_per_fcn_cat_df)

#For every category name, find the length of the functional category in base pairs.
#Then will will divide the sample mutations in a category by this number to get the "mutation rate"
```

```r
#(# of mutation in a category / size of category in BP)

for(cat in cat_names){
        n_mut_per_fcn_cat_df[cat] <-
          n_mut_per_fcn_cat_df[cat]/ sum((func_des_df
                                    %>% filter(function_cat == cat))$gene_length)
}

#We are going to discard the unknown mutation columns from our analysis,
#because we can't normalize this data to find a mutation rate
#( the number of unknown genes is , unsurprisingly, unknown)
n_mut_per_fcn_cat_df <- select(n_mut_per_fcn_cat_df,-unknown)

#Melting the dataframe in preparation for an anova:
melted_mut_fcn_catdf <- melt(n_mut_per_fcn_cat_df)

#Renaming the columns to something we can understand:
melted_mut_fcn_catdf <- melted_mut_fcn_catdf %>% rename(functional_category = 1, mutation_rate = 2)

#Now we need to add a third column delineating what
#population of bacteria the mutation rate occcured in.
#We can do this by making a dictionary, an matchin the row names from
#or n_mut_per_fcn_cat_df
#and then creating a vector out of this.
#We can then mutiply the vector to reach the length of the melted dataframe.

#Setting up a dictionary to make sample rownames to their bacteria populations
b_pop_dict <- c("47_output" = 'Alternating C & K12', "48_output" = 'Alternating C & K12',
            "49_output" = 'Alternating C & K12', "50_output" ='Alternating C & K12',
            "51_output" = 'Alternating C & K12', "52_output"= 'E.coli K12',
            "53_output" = 'E.coli K12', "54_output" = 'E.coli K12',
            "55_output" ='E.coli C' , "56_output" ='E.coli C' ,
            "57_output"= 'E.coli C', "58_output"= 'E.coli C' ,
            "59_output"= 'E.coli C' , "60_output" = 'E.coli K12', "61_output" ='E.coli K12' )

#Creating a vector of bacteria populations
b_pop <- c()
output_names <- rownames(n_mut_per_fcn_cat_df)
for(output_name in output_names){
  b_pop <- c(b_pop,b_pop_dict[output_name])
}

#Now making a vector containing the population designations to add to the melted dataframe:
bact_pop <- vec_rep(c(b_pop),ncol(n_mut_per_fcn_cat_df))

melted_mut_fcn_catdf$bact_pop <- bact_pop #Adding the bacterial population to the dataframe

scheirerRayHare(mutation_rate ~ functional_category + bact_pop, data = melted_mut_fcn_catdf)


#We can also preform a Dunn's test,
#to determine if one functional category in particular has a significant effect mutation rate.
#This does pairwise comparisons of each category to see if the mutation rates are different.
```

```
#You'll see that only comparisons with viron proteins yields significant results.
#install.packages("dunn.test")
print("Running a Dunn Test now")
library(DescTools)
DunnTest(mutation_rate~functional_category,melted_mut_fcn_catdf)
  }
```

# Running a non-parametric ANOVA on all mutations, synonymous mutations and non-synonymous mutations.

Using the SRHanova function we created above, we can run an anova on three subsets of mutations: 1) all mutations, 2) synonymous mutations, 3) non-synonymous mutations. The paper did this same ANOVA to demonstrate that no matter the mutation type, the functional category was most important in determining mutation rates. Our anova results found significant p-values for functional categories (p ~ 0.00). The E.coli strain phage evolved in had no significant affect on the mutation rate. These results were consistent with all mutations, synonymous mutations, and non-synonymous mutations. The Dunn test only found significnat difference in mutation rates in functional category pairs that contains viron proteins. The shows that viron proteins is the only category that has a relatively higher mutation rate.

```
print("Running ANOVA on all mutations")
```

```
## [1] "Running ANOVA on all mutations"
```

```
SRHanova(filtered_all_ann,func_des_df) #no removals
```

```
## 'summarise()' has grouped output by 'title'. You can override using the
## '.groups' argument.
## No id variables; using all as measure variables


##
## DV:  mutation_rate
## Observations:  165
## D:  0.4021345
## MS total:  2282.5
##
## [1] "Running a Dunn Test now"


## Warning: package 'DescTools' was built under R version 4.1.3


##
##  Dunn's test of multiple comparisons using rank sums : holm
##
##                                        mean.rank.diff    pval
## Translation-Transcription                  -5.6000000  1.0000
## Nucleotide_metabolism-Transcription         9.2000000  1.0000
## DNA_rep-Transcription                       8.6000000  1.0000
## Virion_prots-Transcription                 79.6000000 3.2e-11 ***
## chaperonins-Transcription                  -5.6000000  1.0000
## lysis-Transcription                         0.7333333  1.0000
```

```
## host_phage_int-Transcription                              -5.6000000  1.0000
## host_alt-Transcription                                     0.5333333  1.0000
## homing_endonuclease-Transcription                          5.1333333  1.0000
## predicted_integral_membrane-Transcription                 -5.6000000  1.0000
## Nucleotide_metabolism-Translation                         14.8000000  1.0000
## DNA_rep-Translation                                       14.2000000  1.0000
## Virion_prots-Translation                                  85.2000000  7.4e-13 ***
## chaperonins-Translation                                    0.0000000  1.0000
## lysis-Translation                                          6.3333333  1.0000
## host_phage_int-Translation                                 0.0000000  1.0000
## host_alt-Translation                                       6.1333333  1.0000
## homing_endonuclease-Translation                           10.7333333  1.0000
## predicted_integral_membrane-Translation                    0.0000000  1.0000
## DNA_rep-Nucleotide_metabolism                             -0.6000000  1.0000
## Virion_prots-Nucleotide_metabolism                        70.4000000  9.1e-09 ***
## chaperonins-Nucleotide_metabolism                        -14.8000000  1.0000
## lysis-Nucleotide_metabolism                               -8.4666667  1.0000
## host_phage_int-Nucleotide_metabolism                     -14.8000000  1.0000
## host_alt-Nucleotide_metabolism                            -8.6666667  1.0000
## homing_endonuclease-Nucleotide_metabolism                 -4.0666667  1.0000
## predicted_integral_membrane-Nucleotide_metabolism        -14.8000000  1.0000
## Virion_prots-DNA_rep                                      71.0000000  6.5e-09 ***
## chaperonins-DNA_rep                                      -14.2000000  1.0000
## lysis-DNA_rep                                             -7.8666667  1.0000
## host_phage_int-DNA_rep                                   -14.2000000  1.0000
## host_alt-DNA_rep                                          -8.0666667  1.0000
## homing_endonuclease-DNA_rep                               -3.4666667  1.0000
## predicted_integral_membrane-DNA_rep                      -14.2000000  1.0000
## chaperonins-Virion_prots                                 -85.2000000  7.4e-13 ***
## lysis-Virion_prots                                       -78.8666667  5.0e-11 ***
## host_phage_int-Virion_prots                              -85.2000000  7.4e-13 ***
## host_alt-Virion_prots                                    -79.0666667  4.4e-11 ***
## homing_endonuclease-Virion_prots                         -74.4666667  8.1e-10 ***
## predicted_integral_membrane-Virion_prots                 -85.2000000  7.4e-13 ***
## lysis-chaperonins                                          6.3333333  1.0000
## host_phage_int-chaperonins                                 0.0000000  1.0000
## host_alt-chaperonins                                       6.1333333  1.0000
## homing_endonuclease-chaperonins                           10.7333333  1.0000
## predicted_integral_membrane-chaperonins                   0.0000000  1.0000
## host_phage_int-lysis                                      -6.3333333  1.0000
## host_alt-lysis                                            -0.2000000  1.0000
## homing_endonuclease-lysis                                  4.4000000  1.0000
## predicted_integral_membrane-lysis                         -6.3333333  1.0000
## host_alt-host_phage_int                                    6.1333333  1.0000
## homing_endonuclease-host_phage_int                        10.7333333  1.0000
## predicted_integral_membrane-host_phage_int                 0.0000000  1.0000
## homing_endonuclease-host_alt                               4.6000000  1.0000
## predicted_integral_membrane-host_alt                      -6.1333333  1.0000
## predicted_integral_membrane-homing_endonuclease          -10.7333333  1.0000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
filtered_all_ann_noNonSNP <- filtered_all_ann[filtered_all_ann$type == 'SNP',]
filtered_all_ann_synonymous <-
```

```
   filtered_all_ann_noNonSNP[filtered_all_ann_noNonSNP$snp_type == 'synonymous',]
filtered_all_ann_nonsynonymous <-
   filtered_all_ann_noNonSNP[filtered_all_ann_noNonSNP$snp_type == 'nonsynonymous',]
print("synonymous only")
```

## [1] "synonymous only"

```
SRHanova(filtered_all_ann_synonymous,func_des_df)
```

## 'summarise()' has grouped output by 'title'. You can override using the
## '.groups' argument.
## No id variables; using all as measure variables

##
## DV:  mutation_rate
## Observations:  132
## D:  0.2670973
## MS total:  1463
##
## [1] "Running a Dunn Test now"

##
##  Dunn's test of multiple comparisons using rank sums : holm
##
##                                                     mean.rank.diff    pval
## Translation-Transcription                                 0.000000  1.0000
## Nucleotide_metabolism-Transcription                      11.916667  1.0000
## DNA_rep-Transcription                                     5.666667  1.0000
## Virion_prots-Transcription                               53.916667 1.3e-09 ***
## chaperonins-Transcription                                 0.000000  1.0000
## lysis-Transcription                                       0.000000  1.0000
## host_phage_int-Transcription                              0.000000  1.0000
## host_alt-Transcription                                    0.000000  1.0000
## homing_endonuclease-Transcription                         0.000000  1.0000
## predicted_integral_membrane-Transcription                 0.000000  1.0000
## Nucleotide_metabolism-Translation                        11.916667  1.0000
## DNA_rep-Translation                                       5.666667  1.0000
## Virion_prots-Translation                                 53.916667 1.3e-09 ***
## chaperonins-Translation                                   0.000000  1.0000
## lysis-Translation                                         0.000000  1.0000
## host_phage_int-Translation                                0.000000  1.0000
## host_alt-Translation                                      0.000000  1.0000
## homing_endonuclease-Translation                           0.000000  1.0000
## predicted_integral_membrane-Translation                   0.000000  1.0000
## DNA_rep-Nucleotide_metabolism                            -6.250000  1.0000
## Virion_prots-Nucleotide_metabolism                       42.000000 9.0e-06 ***
## chaperonins-Nucleotide_metabolism                       -11.916667  1.0000
## lysis-Nucleotide_metabolism                             -11.916667  1.0000
## host_phage_int-Nucleotide_metabolism                    -11.916667  1.0000
## host_alt-Nucleotide_metabolism                          -11.916667  1.0000
## homing_endonuclease-Nucleotide_metabolism               -11.916667  1.0000
## predicted_integral_membrane-Nucleotide_metabolism       -11.916667  1.0000
```

```
## Virion_prots-DNA_rep                               48.250000 1.1e-07 ***
## chaperonins-DNA_rep                                 -5.666667  1.0000
## lysis-DNA_rep                                       -5.666667  1.0000
## host_phage_int-DNA_rep                              -5.666667  1.0000
## host_alt-DNA_rep                                    -5.666667  1.0000
## homing_endonuclease-DNA_rep                         -5.666667  1.0000
## predicted_integral_membrane-DNA_rep                 -5.666667  1.0000
## chaperonins-Virion_prots                           -53.916667 1.3e-09 ***
## lysis-Virion_prots                                 -53.916667 1.3e-09 ***
## host_phage_int-Virion_prots                        -53.916667 1.3e-09 ***
## host_alt-Virion_prots                              -53.916667 1.3e-09 ***
## homing_endonuclease-Virion_prots                   -53.916667 1.3e-09 ***
## predicted_integral_membrane-Virion_prots           -53.916667 1.3e-09 ***
## lysis-chaperonins                                    0.000000  1.0000
## host_phage_int-chaperonins                           0.000000  1.0000
## host_alt-chaperonins                                 0.000000  1.0000
## homing_endonuclease-chaperonins                      0.000000  1.0000
## predicted_integral_membrane-chaperonins              0.000000  1.0000
## host_phage_int-lysis                                 0.000000  1.0000
## host_alt-lysis                                       0.000000  1.0000
## homing_endonuclease-lysis                            0.000000  1.0000
## predicted_integral_membrane-lysis                    0.000000  1.0000
## host_alt-host_phage_int                              0.000000  1.0000
## homing_endonuclease-host_phage_int                   0.000000  1.0000
## predicted_integral_membrane-host_phage_int           0.000000  1.0000
## homing_endonuclease-host_alt                         0.000000  1.0000
## predicted_integral_membrane-host_alt                 0.000000  1.0000
## predicted_integral_membrane-homing_endonuclease      0.000000  1.0000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
print("nonsynonymous only")
```

```
## [1] "nonsynonymous only"
```

```r
SRHanova(filtered_all_ann_nonsynonymous,func_des_df)
```

```
## `summarise()` has grouped output by 'title'. You can override using the
## `.groups` argument.
## No id variables; using all as measure variables
```

```
##
## DV:  mutation_rate
## Observations:  165
## D:  0.3625865
## MS total:  2282.5
##
## [1] "Running a Dunn Test now"
```

```
##
##  Dunn's test of multiple comparisons using rank sums : holm
##
```

```
##                                                  mean.rank.diff    pval
## Translation-Transcription                           -5.76666667   1.0000
## Nucleotide_metabolism-Transcription                 -0.60000000   1.0000
## DNA_rep-Transcription                                3.83333333   1.0000
## Virion_prots-Transcription                          76.73333333   1.4e-11 ***
## chaperonins-Transcription                           -5.76666667   1.0000
## lysis-Transcription                                  0.46666667   1.0000
## host_phage_int-Transcription                        -5.76666667   1.0000
## host_alt-Transcription                               0.40000000   1.0000
## homing_endonuclease-Transcription                    5.30000000   1.0000
## predicted_integral_membrane-Transcription           -5.76666667   1.0000
## Nucleotide_metabolism-Translation                    5.16666667   1.0000
## DNA_rep-Translation                                  9.60000000   1.0000
## Virion_prots-Translation                            82.50000000   2.2e-13 ***
## chaperonins-Translation                              0.00000000   1.0000
## lysis-Translation                                    6.23333333   1.0000
## host_phage_int-Translation                           0.00000000   1.0000
## host_alt-Translation                                 6.16666667   1.0000
## homing_endonuclease-Translation                     11.06666667   1.0000
## predicted_integral_membrane-Translation              0.00000000   1.0000
## DNA_rep-Nucleotide_metabolism                        4.43333333   1.0000
## Virion_prots-Nucleotide_metabolism                  77.33333333   9.3e-12 ***
## chaperonins-Nucleotide_metabolism                   -5.16666667   1.0000
## lysis-Nucleotide_metabolism                          1.06666667   1.0000
## host_phage_int-Nucleotide_metabolism                -5.16666667   1.0000
## host_alt-Nucleotide_metabolism                       1.00000000   1.0000
## homing_endonuclease-Nucleotide_metabolism            5.90000000   1.0000
## predicted_integral_membrane-Nucleotide_metabolism   -5.16666667   1.0000
## Virion_prots-DNA_rep                                72.90000000   1.8e-10 ***
## chaperonins-DNA_rep                                 -9.60000000   1.0000
## lysis-DNA_rep                                       -3.36666667   1.0000
## host_phage_int-DNA_rep                              -9.60000000   1.0000
## host_alt-DNA_rep                                    -3.43333333   1.0000
## homing_endonuclease-DNA_rep                          1.46666667   1.0000
## predicted_integral_membrane-DNA_rep                 -9.60000000   1.0000
## chaperonins-Virion_prots                           -82.50000000   2.2e-13 ***
## lysis-Virion_prots                                 -76.26666667   1.9e-11 ***
## host_phage_int-Virion_prots                        -82.50000000   2.2e-13 ***
## host_alt-Virion_prots                              -76.33333333   1.8e-11 ***
## homing_endonuclease-Virion_prots                   -71.43333333   4.8e-10 ***
## predicted_integral_membrane-Virion_prots           -82.50000000   2.2e-13 ***
## lysis-chaperonins                                    6.23333333   1.0000
## host_phage_int-chaperonins                           0.00000000   1.0000
## host_alt-chaperonins                                 6.16666667   1.0000
## homing_endonuclease-chaperonins                     11.06666667   1.0000
## predicted_integral_membrane-chaperonins              0.00000000   1.0000
## host_phage_int-lysis                                -6.23333333   1.0000
## host_alt-lysis                                      -0.06666667   1.0000
## homing_endonuclease-lysis                            4.83333333   1.0000
## predicted_integral_membrane-lysis                   -6.23333333   1.0000
## host_alt-host_phage_int                              6.16666667   1.0000
## homing_endonuclease-host_phage_int                  11.06666667   1.0000
## predicted_integral_membrane-host_phage_int           0.00000000   1.0000
## homing_endonuclease-host_alt                         4.90000000   1.0000
```

```
## predicted_integral_membrane-host_alt                -6.16666667  1.0000
## predicted_integral_membrane-homing_endonuclease    -11.06666667  1.0000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## Plotting mutations across the genome for each sample:

Our last task will be to graph the mutations for each sample across then entire genome length. This will tell us if there's any similarity between samples, and we can also compare against the original figure here.

```
#We are going to try and do this with a scatter plot, using the x- axis as position of the
#mutation in the genome. We will make the y-axis a different fixed number for each sample.

#install.packages("ggpubr")
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 4.1.3
```

```
#First we will make a dictionary with y-axis values for each of the sample names:
y_axis_dict <- c("47_output" = 5, "48_output" = 4, "49_output" = 3, "50_output" =2,
                 "51_output" = 1, "52_output"= 5, "53_output" = 4, "54_output" = 3,
                 "55_output" =5 , "56_output" =4 , "57_output"= 3, "58_output"= 2,
                 "59_output"= 1, "60_output" = 2, "61_output" =1 )

#Now we are going to add a new column to the "filtered_all_ann" dataframe which contains
#the y-axis value corresponding to each sample name.
mutation_y_axis_vals <- c()
output_names <- filtered_all_ann$title
for(output_name in output_names){
  y_val <- y_axis_dict[output_name]
  mutation_y_axis_vals <- c(mutation_y_axis_vals,y_val)
}
filtered_all_ann$y_ax_val <- mutation_y_axis_vals #Adding this to our mutation dataframe

# doing the same thing for bacterial population names
b_pop_dict <- c("47_output" = 'Alternating C & K12', "48_output" = 'Alternating C & K12',
    "49_output" = 'Alternating C & K12', "50_output" ='Alternating C & K12', "51_output"
    = 'Alternating C & K12', "52_output"= 'E.coli K12', "53_output" = 'E.coli K12',
    "54_output" = 'E.coli K12', "55_output" ='E.coli C' , "56_output" ='E.coli C' ,
    "57_output"= 'E.coli C', "58_output"= 'E.coli C' , "59_output"= 'E.coli C' ,
    "60_output" = 'E.coli K12', "61_output" ='E.coli K12' )

population_y_axis_vals <- c()
output_names <- filtered_all_ann$title
for(output_name in output_names){
  y_val <- b_pop_dict[output_name]
  population_y_axis_vals <- c(population_y_axis_vals,y_val)
}
filtered_all_ann$population_name <- population_y_axis_vals #Adding this to our mutation dataframe

#Now we will attempt to maake a scatterplot with all the mutations:
```

```
library(ggplot2)
p = ggplot(filtered_all_ann, aes(x=position,y =
reorder(y_ax_val, desc(y_ax_val)), shape = mutation_category, color = gene_function)) +
  geom_point(aes(fill = gene_function)) + scale_shape_manual(values = c(21,5,24,22,2,4)) + ylab("sample
p + facet_grid(factor(population_name, level = c('E.coli C','E.coli K12','Alternating C & K12'))~.)
```
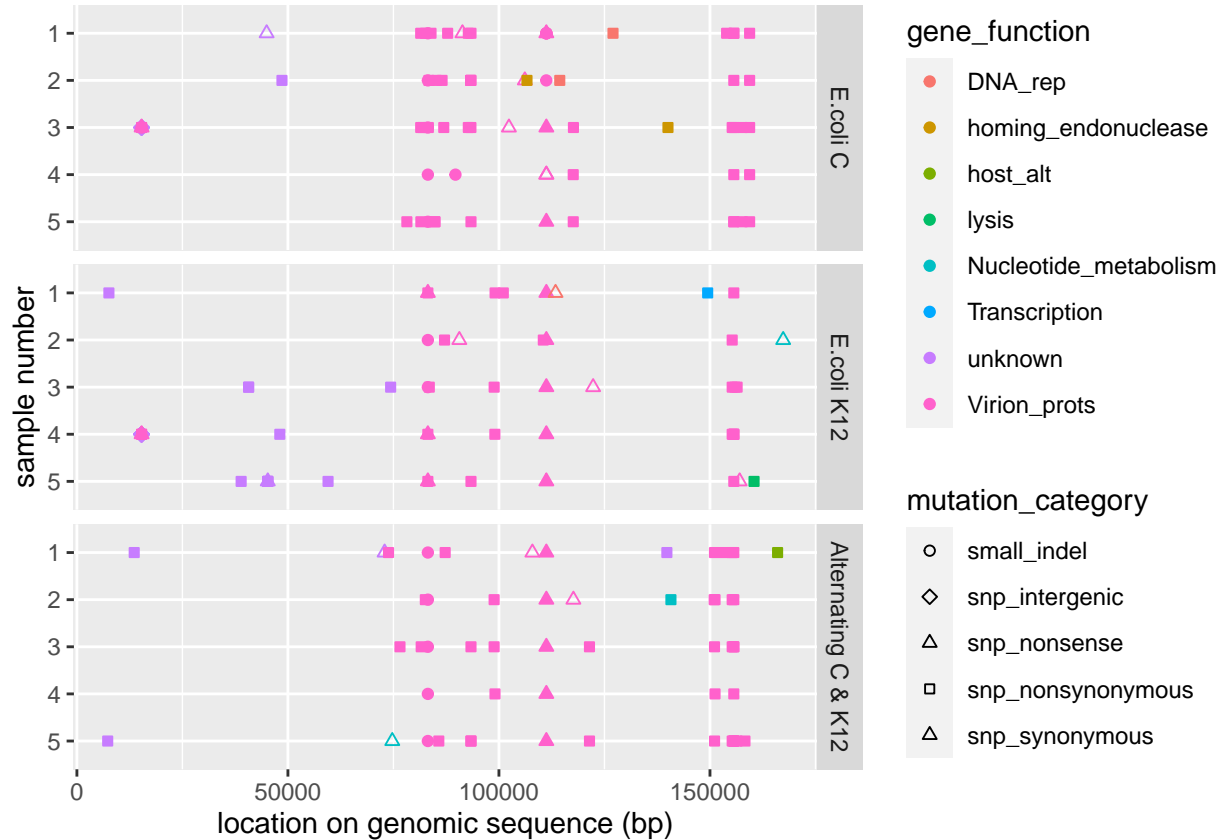


#Conclusion:

Overall we reached similar findings to the Pham et al. paper. First, we demonstrated that selection did occur, and found strong evidence that multiple non-synonymous mutations occurred and persisted withint the population. We found that most of the mutations were dominated by structural genes. We also concluded that the mutation rate in structural genes was the only category with significantly higher mutation rates. This result was the same as the study. Through our non-parametric ANOVA, we concluded that the bacteria phage evolved with (E.coliC, K12 or alternative populations) made no difference on the relative mutation rates in any functional category. This result also lined up with the paper. Our final genome mutation plot does not look identical to the one in Pham et al's paper. This is likely because we did not have access to the ancestral sample. When we initially filtered our mutations against those reported in their supplementary table, it's possible we lost some mutations. I noticed they changed some of the mutation positions by hand during there analysis, and this may have messed up our filtering. This isn't a huge worry, as with the ancestral sequence and proper filtering we would probably get the same results.

Below is a plot of the mutations found by the paper for comparison. One can see we have somewhat similar results, but there's a few points missing & a few extra ones. The extra ones may come from mutations that were actually in the ancestral sequence, but had a matching position to the mutations reported in the paper's supplementary table, so never got filtered out.

References: The paper we followed: Pham, Jenny Y., C. Brandon Ogbunugafor, Alex N. Nguyen Ba, and Daniel L. Hartl. "Experimental Evolution for Niche Breadth in Bacteriophage T4 Highlights the Importance
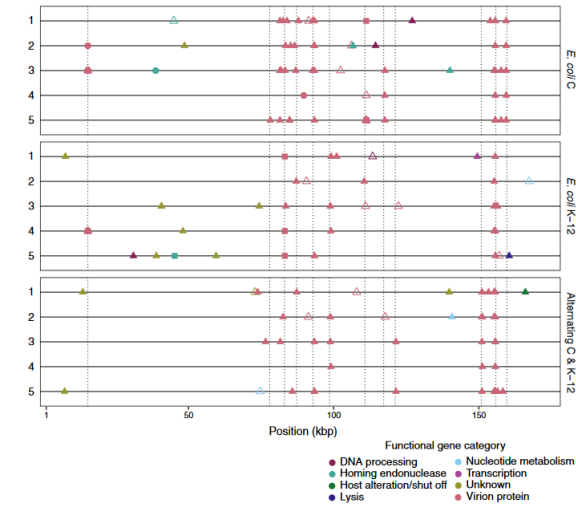
FIGURE 4 *The distribution of new mutations detected in the evolved phage populations.* Panels indicate one of three evolutionary histories, and lines within panels represent particular populations within an evolutionary history; numbers for each horizontal line correspond to the numbered populations that underwent passaging. Each point represents a new mutation (relative to the ancestor) detected at ≥5% frequency. Point shape indicates type of mutation (filled triangles: nonsynonymous; empty triangles: synonymous; circles: indel; squares: xNPs (multiple adjacent nucleotide polymorphisms); and diamonds: intergenic). Colors indicate functional gene category; labels above the top panel and dashed lines denote relative positions of virion structural genes

Figure 8: Adapters to get rid of

of Structural Genes." MicrobiologyOpen 9, no. 2 (2020): e968. https://doi.org/10.1002/mbo3.968.

The paper that give the T4 genome, genes and functional designations: Miller, Eric S., Elizabeth Kutter, Gisela Mosig, Fumio Arisaka, Takashi Kunisawa, and Wolfgang Rüger. "Bacteriophage T4 Genome." Microbiology and Molecular Biology Reviews 67, no. 1 (March 2003): 86–156. https://doi.org/10.1128/MMBR.67.1.86-156.2003.

Next time I would do the analysis very similarly, but with a functioning ancestral sequence so that ancestral mutations can be filtered out using breseq's tools. I would also try to visualize some of the mutations on the protein PBD structures themselves.