groot Documentation

Release 1.1.2

Will Rowe

May 11, 2020

Contents

GROOT is a tool to profile **Antibiotic Resistance Genes** (**ARGs**) in metagenomic samples.

The main advantages of *GROOT* over existing tools are:

- quick classification of reads to candidate ARGs
- accurate annotation of full-length ARGs
- can run on a laptop in minutes

GROOT aligns reads to ARG variation graphs, producing an alignment file that contains the graph traversals possible for each query read. The alignment file is then used to generate a simple resistome profile report.

CHAPTER 1

Overview

Antimicrobial resistance remains a major threat to global health. Profiling the collective antimicrobial resistance genes within a metagenome (the "resistome") facilitates greater understanding of antimicrobial resistance gene diversity and dynamics. In turn, this can allow for gene surveillance, individualised treatment of bacterial infections and more sustainable use of antimicrobials. However, resistome profiling can be complicated by high similarity between reference genes, as well as the sheer volume of sequencing data and the complexity of analysis workflows. We have developed an efficient and accurate method for resistome profiling that addresses these complications and improves upon currently available tools.

Our method combines variation graph representation of gene sets with an LSH Forest indexing scheme to allow for fast classification of metagenomic reads using similarity-search queries. Subsequent hierarchical local alignment of classified reads against graph traversals facilitates accurate reconstruction of full-length gene sequences using a scoring scheme. GROOT runs on a laptop and can process a typical 2 gigabyte metagenome in 2 minutes using a single CPU.

CHAPTER 2

Contents

2.1 Using GROOT

GROOT uses a subcommand syntax; call the main program with `groot` and follow it by the subcommand to indicate what action to take.

This page will cover a worked example, details on the available GROOT commands and some tips for using the program.

For more information on the graphs that **GROOT** uses, please read the groot-graphs page.

2.1.1 An example

This example will take us from raw reads to resistome profile for a single metagenome

Get some sequence data:

```
fastq-dump SRR4454613
```

Get a pre-clustered ARG database:

```
groot get -d resfinder
```

Create variation graphs from the ARG-database and index:

```
groot index -m resfinder.90 -i grootIndex -w 100 -p 8
```

Align the reads and report ARGs

```
groot align -i grootIndex -f SRR4454613.fastq -p 8 | groot report -c 0.95
```

2.1.2 GROOT subcommands

get

The `get` subcommand is used to download a pre-clustered ARG database that is ready to be indexed. Here is an example:

```
groot get -d resfinder -o . --identity 90
```

The above command will download the ResFinder database, which has been clustered at 90% identity, check its integrity and unpacks it to the current directory. The database will be named `resfinder.90` and contain several `*.msa` files.

Flags explained:

- `-d`: which database to get
- `-o`: directory to save the database to
- `--identity`: the identity threshold at which the database was clustered (note: only 90 currently available)

The following databases are available:

- arg-annot (default)
- resfinder
- card
- groot-db
- groot-core-db

These databases were clustered by sequence identity and stored as **Multiple Sequence Alignments** (MSAs). See groot-databases for more info.

index

The `index` subcommand is used to create variation graphs from a pre-clustered ARG database and then index them. Here is an example:

```
groot index -m resfinder.90 -i grootIndex -w 100 -p 8
```

The above command will create a variation graph for each cluster in the `resfinder.90` database and initialise an LSH forest index. It will then move through each graph traversal using a 100 node window, creating a MinHash sketch for each window. Finally, each sketch is added to the LSH Forest index. The index will be named `grootIndex` and contain several files.

Flags explained:

- `-m`: a directory of MSA files (the database from `groot get`)
- `-i`: where to save the index
- `-w`: the window size to use (**should be similar to the length of query reads**)
- `-p`: how many processors to use

The same index can be used on multiple samples, however, it should be re-indexed if you wish to change the seeding parameters.

Some more flags that can be used:

- `-k`: size of k-mer to use for MinHashing
- `-s`: size of MinHash sketch
- `-x`: number of partitions in the LSH Ensemble index
- `-y`: maxK in the LSH Ensemble index
- `--maxSketchSpan`: max number of identical neighbouring sketches permitted in any graph traversal

  Important: GROOT can only accept MSAs as input. You can cluster your own database or use `groot get` to obtain a pre-clustered one.

align

The `align` subcommand is used to align reads against the indexed variation graphs. Here is an example:

```
groot align -i grootIndex -f file.fastq -t 0.97 -p 8 > ARG-reads.bam
```

The above command will seed the fastq reads against the indexed variation graphs. It will then perform a hierarchical local alignment of each seed against the variation graph traversals. The output alignment is essentially the ARG classified reads (which may be useful) and can then be used to report full-length ARGs (using the `report` subcommand).

Flags explained:

- `-i`: which index to use
- `-f`: what FASTQ reads to align
- `-t`: the containment threshold for seeding reads
- `-p`: how many processors to use

Data can streamed in and out of the align subcommand. For example:

```
gunzip -c file.gz | ./groot align -i grootIndex -p 8 | ./groot report
```

Multiple FASTQ files can be specified as input, however all are treated as the same sample and paired-end info isn't used. To specify multiple files, make sure they are comma separated (`-f fileA.fq,fileB.fq`) or use gunzip/cat with a wildcard (gunzip -c *.fq.gz | groot...).

Some more flags that can be used:

- `--noAlign`: if set, no exact alignment will be performed (graphs will still be weighted using approximate read mappings)

report

The `report` subcommand is used to processes graph traversals and generate a resistome profile for a sample. Here is an example:

```
groot report --bamFile ARG-reads.bam -c 1
```

Flags explained:

- `--bamFile`: the input BAM file (output from `groot align` subcommand)
- `-c`: the coverage needed to report an ARG (e.g. 0.95 = 95% ARG bases covered by reads)

Some more flags that can be used:

- `--lowCov`: overrides `c` option and will report ARGs which may not be covered at the 5'/3' ends

2.2 GROOT Graphs

This is a brief overview of the graphs used by **GROOT** and what they can be used for.

2.2.1 Overview

If you are reading this, you are probably familiar with some of the many Antibiotic Resistance Gene (**ARG**) databases that exist (e.g. ResFinder, CARD, ARG-annot). Each of these databases are basically multifasta files that contain the sequences of **ARGs** that confer resistance to antibiotics.

The **ARG** sequences in these databases can be grouped by type (based on nucleotide identity). Within each of these groups there are 1 or more **ARG** sequences which can vary from each other by 1 or more nucleotides. For example, the group of `bla-OXA` type genes contains many closely related sequences of genes encoding carbapenemases (`bla-OXA-1`, `bla-OXA-2`, `bla-OXA-3` etc.) . Traditionally, querying the **ARG** databases involves looking at each of these closely related sequences within each group in turn, even if they only vary by 1 nucleotide.

Rather than using all of the linear reference sequences in the **ARG** databases, **GROOT** collapses closely related genes into variation graphs. For example, it can take all `bla-OXA` type genes in the ResFinder database and represent these as a single variation graph. To get the groups of closely related genes within a database, the database is clustered and each cluster is stored as a Multiple Sequence Alignment (**MSAs**). Please read the groot-databases page for more information on this.

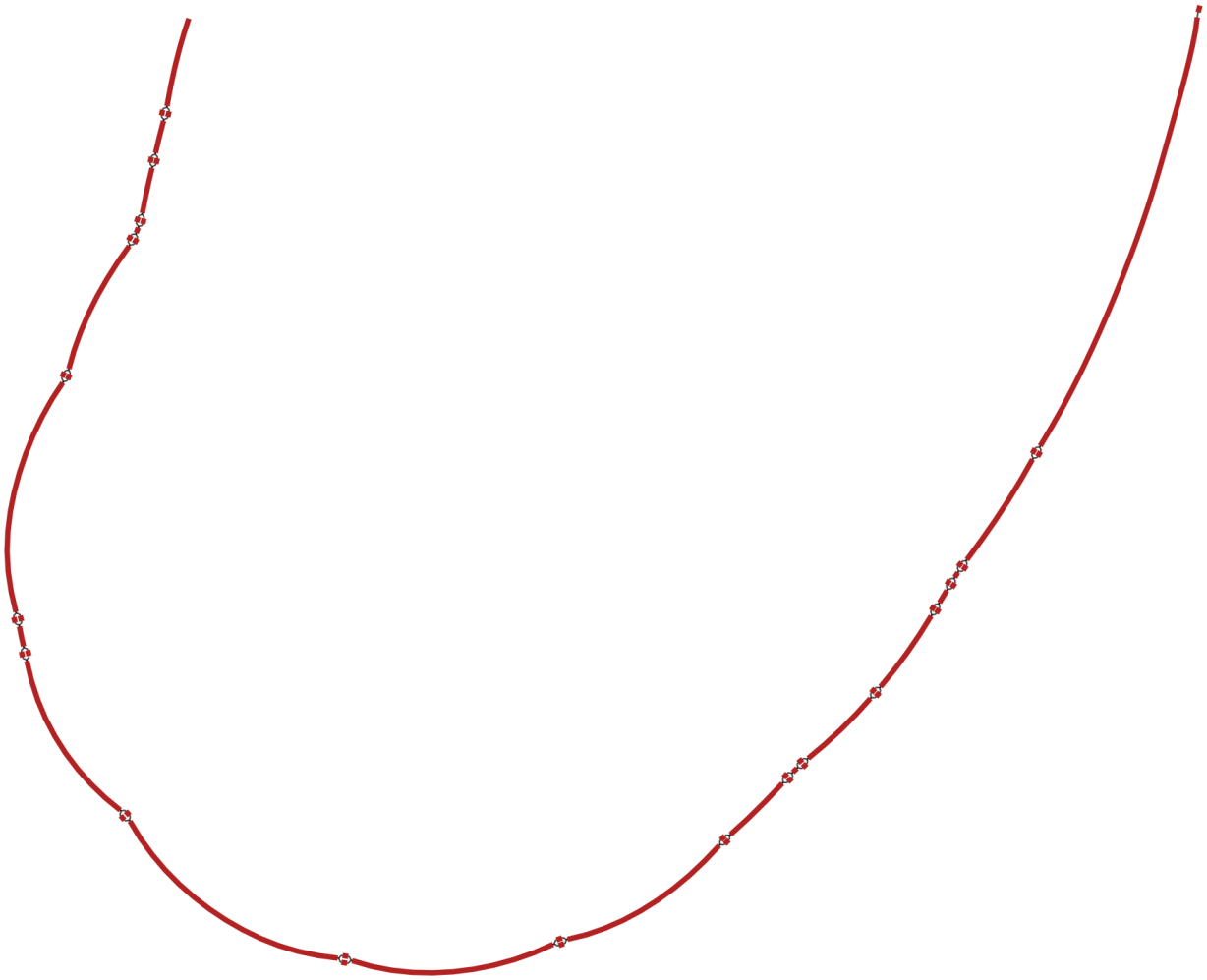
GROOT creates variation graphs from **MSAs** and stores them as special structures called `groot graphs`. The `groot graphs` essentially follow the format conventions of the Graphical Fragment Assembly (GFA) format that is used by some genome assemblers and consists of:

- `Segments`: a continuous sequence or subsequence.
- `Links`: an overlap between two segments. Each link is from the end of one segment to the beginning of another segment. The link stores the orientation of each segment and the amount of basepairs overlapping.
- `Paths`: an ordered list of oriented segments, where each consecutive pair of oriented segments are supported by a link record.

So, a `groot graph` is:

> a series of connected nodes; each node containing a segment, the links for that segment and the paths (genes) which utilise that segment.

For example, here is a graph for a set of closely related `Bla-CTX-M` type genes:



ctxm-example

This graph is derived from 3 genes: `(Bla)CTX-M-40`, `(Bla)CTX-M-63`, `(Bla)CTX-M-8`. Each of these genes has a path through the graph, involving different combinations of segments. **GROOT** will index each of the paths so that it can seed query reads to graph regions.

2.2.2 Using the graphs

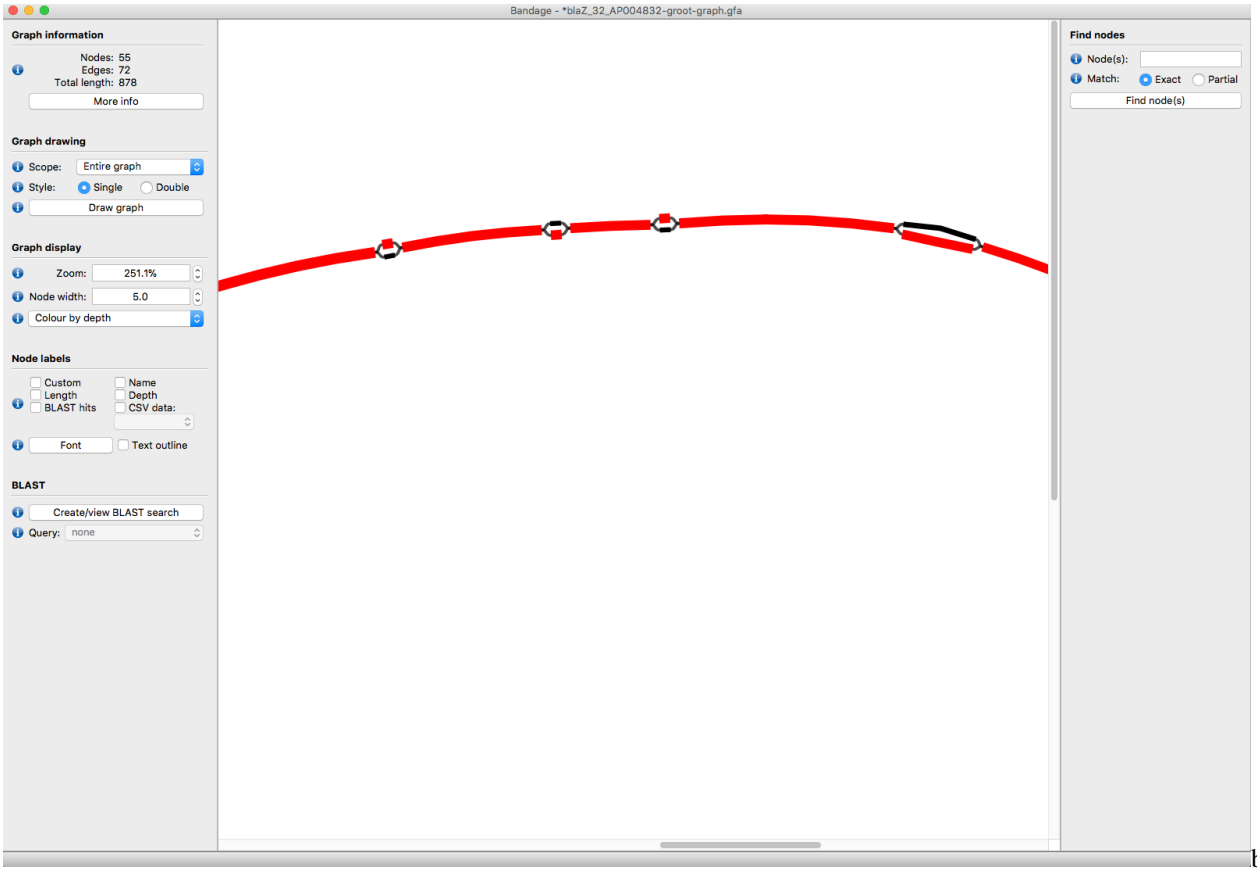
Since version 0.4, **GROOT** outputs any `groot graphs` that have had reads align to them. These graphs are in **GFA** format (`.gfa`) and can be viewed in Bandage.

In each GFA file, **GROOT** also records the number of reads that aligned to each segment of the variation graph. This allows us to look at which variants in an ARG gene cluster are dominant in a sample. For example, let's look at a graph for `BlaZ` type genes found in a gut microbiome sample:

```
groot get -d resfinder
groot index -m resfinder.90 -i index -p 8
groot align -i index -f reads-1.fq,reads-2.fq -g groot-graphs -p 8 | groot report --
↪lowCov

Bandage load groot-graphs/blaZ-groot-graph.gfa --draw
```

In the `Graph display` options, select `Colour by depth`. You will get something like this:



blaz-example

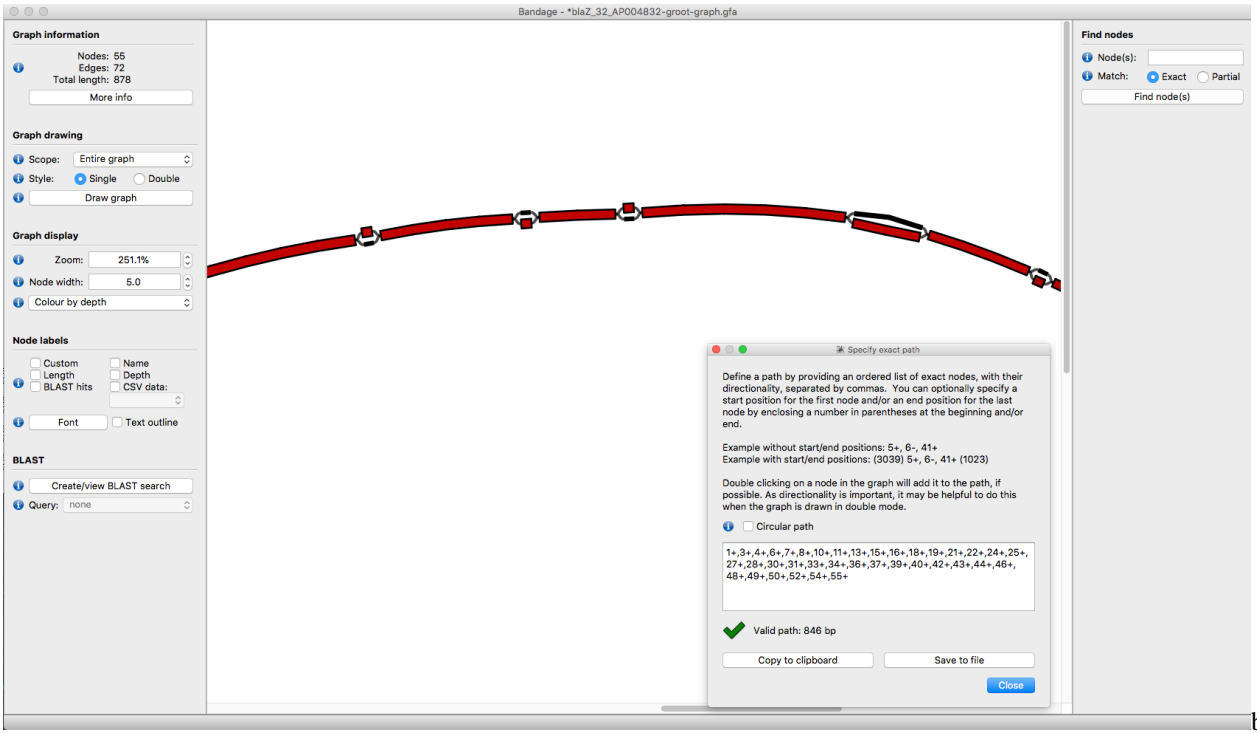
In this example, **GROOT** found two blaZ variants in a microbiome (`blaZ_35` and `blaZ_36`), but one has 50X more coverage than the other over the variant sites (the red segments).

To tell which segments come from which genes, we can overlay path information. One way to do this is to blast each reference sequence against the graph segments and then colour the segments. Another way is to specify a path:

- open the `blaZ-groot-graph.gfa` file and get the path you want to highlight

```
e.g. blaZ_35:
1+,3+,4+,6+,7+,8+,10+,11+,13+,15+,16+,18+,19+,21+,22+,24+,25+,27+,28+,30+,31+,33+,34+,
↪36+,37+,39+,40+,42+,43+,44+,46+,48+,49+,50+,52+,54+,55+
```

- in Bandage, select `Output > Specify Exact Path For Copy` and enter the path
- Bandage will then highlight the segments in the path:



blaz-example2

To finish, an important point about the RC and FC tags in the `groot-graph.gfa` files:

> RC tags record the number of reads aligned to a segment, multipled by the length of the segment. This is so that bandage displays the depth nicely. FC tags record just the raw number of reads aligned to a segment.

2.3 GROOT Databases

This is a brief overview of the databases used by **GROOT**.

2.3.1 Overview

As mentioned on the groot-graphs page, **GROOT** creates variation graphs from Multiple Sequence Alignments (**MSAs**) and stores them as `groot graphs`.

The **MSAs** represent a clustered database, each cluster is a collection of sequences which share high nucleotide identity. **GROOT** reads in each **MSA** file and converts it to a graph.

A set of clustered databases is provided (use the `groot get` subcommand) or you can generate your own clustered database. To cluster a database yourself, you can use the following commands on any multifasta file containing the sequences you want to use with **GROOT**:

```
mkdir CLUSTERED-DB && cd $_

vsearch --cluster_size /path/to/ARGs.fna --id 0.90 --msaout MSA.tmp

awk '!a[$0]++ {of="./cluster-" ++fc ".msa"; print $0 >> of ; close(of)}' RS= ORS="\n\n
↪" MSA.tmp && rm MSA.tmp

cd ..
```

- the above snippet will create a clustered database in the directory CLUSTERED-DB - now you can run `groot index`

```
groot index -m ./CLUSTERED-DB
```

2.3.2 groot-db and groot-core-db

As mentioned earlier, the `groot get` subcommand can download a pre-clustered database for you to use with **GROOT**. The following databases are available:

- arg-annot (default)
- resfinder
- card
- groot-db
- groot-core-db

The `groot-db` and `groot-core-db` are both databases that are derived from ResFinder, ARG-annot and CARD. They have been included after requests by several users for a combination of available **ARG** databases. They were made as follows:

`groot-db` is made by combining all sequences in ResFinder, ARG-annot and CARD. Duplicates are removed and the sequences are then clustered at 90% identity.

`groot-core-db` is made by combining sequences that are present in each of the ResFinder, ARG-annot and CARD databases. One copy of each sequence is kept and then this collection is clustered at 90% identity.

Both `groot-db` and `groot-core-db` prepend a tag to each reference sequence so that the origin can be determined. For example:

```
>*groot-db_ARGANNOT__(AGly)APH-Stph:HE579073:1778413-1779213:801
```

In the directory downloaded by `groot get`, there will also be a timestamp that tells you when the database was created. The **GROOT** database will have used the most recently available versions of ResFinder/CARD/ARG-annot. The script used to do this is available in the **GROOT** repo:

```
db/groot-database/make-groot-dbs.sh
```

2.4 Tutorial

This tutorial is intended to show you how to use **GROOT** to identify Antimicrobial Resistance Genes (**ARGs**) and generate resistome profiles as part of a metagenome analysis workflow.

We will use some metagenome data from a recent paper by Winglee et al., where they identify differences in the microbiota in rural versus recently urban subjects from the Hunan province of China.

In particular, we will explore this finding:

> "Urban subjects have increased gene diversity, including increased antibiotic resistance"

The aims are:

- download and check the metagenome data (x40 samples)
- classify ARG-derived reads
- generate resistome profiles for each sample
- visualise the data
- look at ARG context

2.4.1 1. Setup an environment

Use conda to set up an environment with all the software we need for this tutorial:

```
conda create -n grootTutorial -c bioconda parallel sra-tools==2.8 fastqc==0.11.7␣
↪bbmap==37.90 multiqc groot==0.8.5 seqkit==0.7.2 samtools==1.4 metacherchant==0.1.0
source activate grootTutorial
```

2.4.2 2. Get the data

We downloaded `additional file 1, table s1` from the Winglee paper and saved the SRA accession number and the rural/urban status for each metagenome used in the study.

To begin, save the accession table to file:

Next, use `fastq-dump` to download the sequence data from the accession table:

```
cut -f 1 samples.txt | parallel --gnu "fastq-dump {}"
```

2.4.3 3. QC

Run FastQC and check the quality of the data:

```
ls *.fastq | parallel --gnu "fastqc {}"
multiqc .
```

- open up the MultiQC report in a browser and check the data
- samples should all be clear of adapters
- 3 samples failed on sequence quality

Try cleaning up the 3 failed samples using BBduk:

```
ls SRR4454598.fastq SRR4454599.fastq SRR4454610.fastq | parallel --gnu "bbduk.sh in={}
↪ out={/.}.trimmed.fastq qtrim=rl trimq=20 minlength=97"
```

- check the data is now passing on sequence quality
- replace the original failed files with the trimmed ones

2.4.4 4. Run GROOT

Download the pre-clustered ARG-ANNOT database:

```
groot get -d arg-annot
```

Index the database:

```
groot index -m arg-annot.90 -i groot-index -w 100 -p 8
```

- the metagenomes reads we downloaded earlier are 100 bases long
- we need to set the indexing window size (l) to 100 to match the query read length
- the default MinHash settings should be fine but you can play with them (`groot index --help`)

Align the reads against the index:

```
ls *.fastq | parallel --gnu "groot align -i groot-index -f {} -p 8 -g {/.}-groot-
↪graphs > {/.}.bam"
```

- for each sample, the align subcommand produces a BAM file containing all graph traversals for each read
- each BAM file essentially contains the ARG-derived reads in each sample
- the graphs (`.gfa`) which had reads align are stored in a separate directory for each sample (`-g {/.}-groot-graphs`)

4.a. Compare ARG diversity

Now that we have classified ARG-derived reads from all of the samples, we can compare the proportion of ARG-derived reads in urban vs. rural samples. We are aiming to replicate the analysis as presented in figure 5d of the Winglee paper:

fig5

Firstly, calculate the proportion of ARG-derived reads in each sample and save to a csv file:

```
while read -r line
do
    ID=$(echo $line | cut -f1 -d ' ')
    status=$(echo $line | cut -f2 -d ' ')
    bamFile=${ID}.bam
    mapped=$(samtools view -c -F 4 $bamFile)
    proportion=$(echo "scale=10 ; $mapped / 10000000" | bc)
    printf "$status,$proportion\n" >> proportion-ARG-derived.csv
done < samples.txt
```

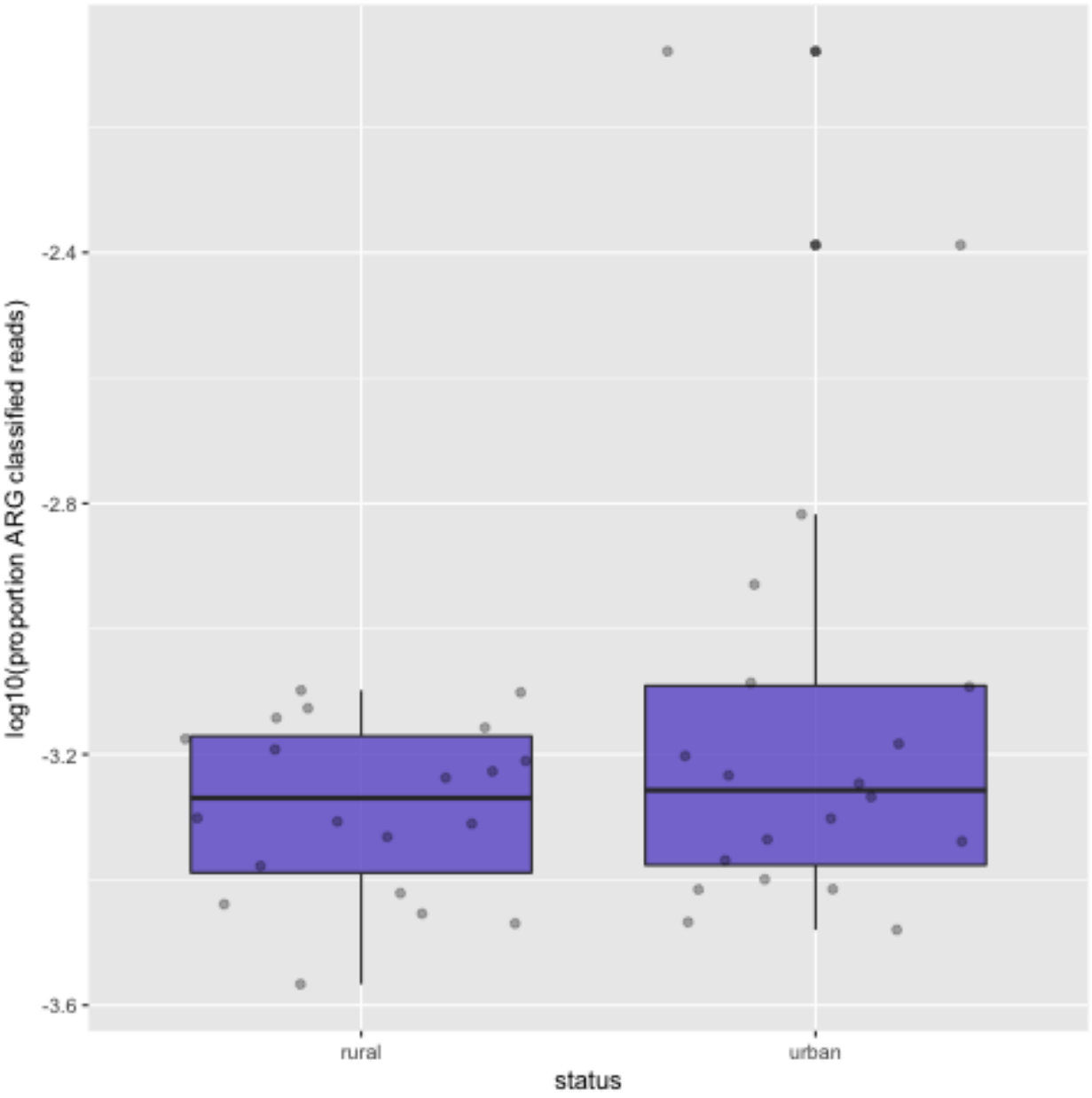
Now create a simple box plot using R:

```
library(ggplot2)

# read in the data and log transform the proportion of ARG classified reads
data <- read.csv(file='proportion-ARG-derived.csv', header=FALSE)
colnames(data) <- c("status", "proportion")
data[2] <- log(data[2], 10)

# plot the data and save to file
png(file = "boxplot.png")
ggplot(data, aes(x=status, y=proportion)) + geom_boxplot(fill="slateblue", alpha=0.8)␣
↪+ geom_point(alpha = 0.3, position = "jitter") + xlab("status") + ylab(
↪"log10(proportion ARG classified reads)")
dev.off()
```

This should produce a box plot like this:



boxplot

We have now compared the proportion of ARG-derived reads from urban and rural samples. This has just used the ARG-derived reads, as was done in the original paper. Although the mean proportion of ARG-derived reads is greater in the urban set, our figure still looks a little different to the published figure. More to come on this…

4.b. Generate resistome profiles

The main advantage of GROOT is that it will generate a resistome profile by reporting full-length ARGs. To do this, we need to use the `groot report` command. The report command will output a tab separated file that looks like this:

Let's report the resistome profiles for each of our samples:

```
ls *.bam | parallel --gnu "groot report --bamFile {} -c 1 --plotCov > {/.}.report"
```

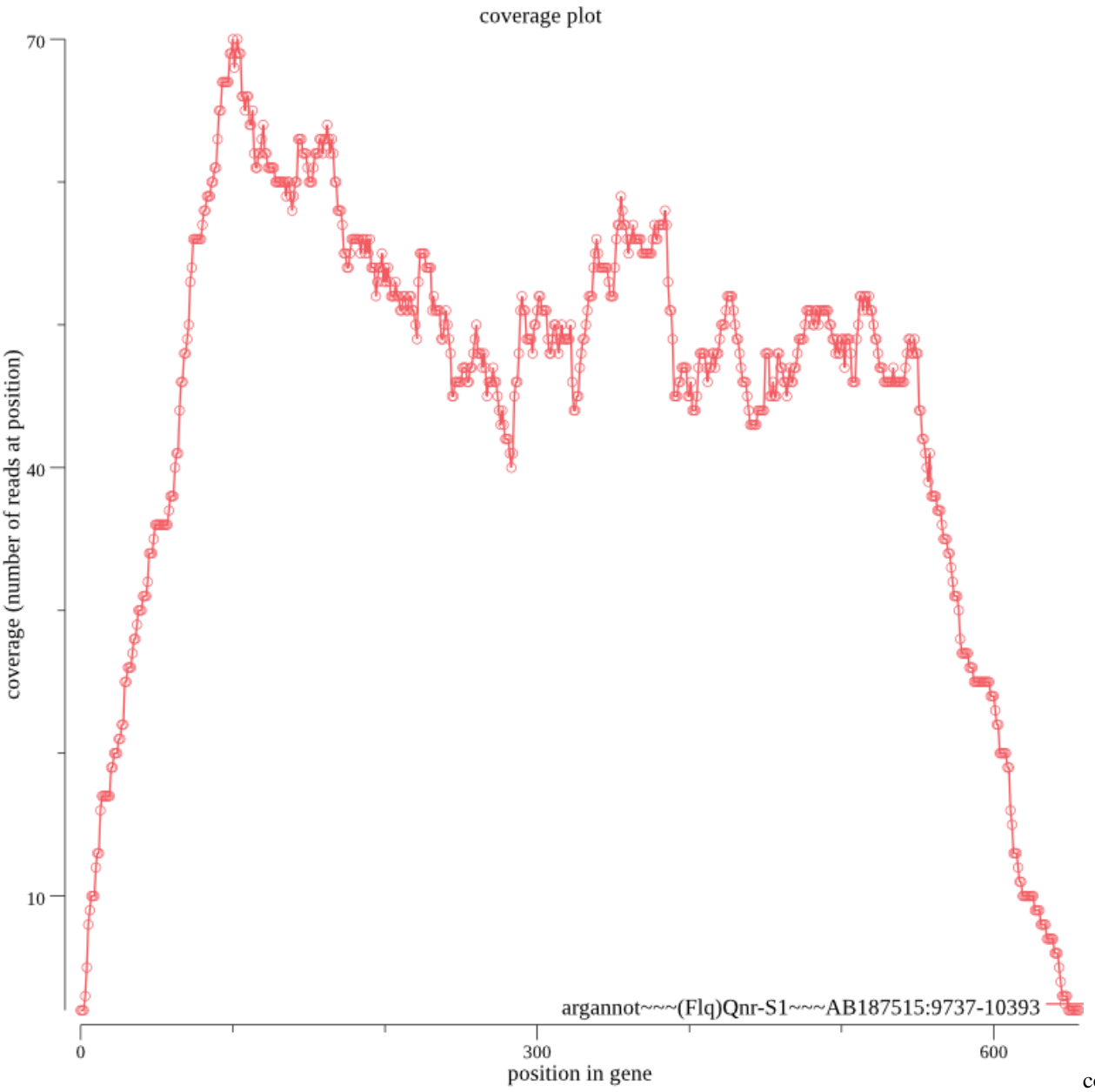
- `-c 1` tells groot to only report ARGs that have been entirely covered by reads, I.E. a full-length,100% identity match
- `--plotCov` tells groot to generate coverage plots for each ARG it reports

  NOTE: `--plotCov` was removed after v.1.0.0 as it relied on a library that could not be packaged for conda. Hopefully I can add it back in soon.

We have now generated a resistome profile for each sample, using only full-length ARG sequences (present in the ARG-ANNOT database). We can sum rural/urban resistome profiles with a little bash loop to combine reports:

```
while read -r line
do
    ID=$(echo $line | cut -f1 -d ' ')
    status=$(echo $line | cut -f2 -d ' ')
    if [ -s $ID.report ]
        then
            cat $ID.report >> combined-profiles.$status.tsv
    fi
done < samples.txt
wc -l combined-profiles.*
```

The number of ARGs found isn't very large and the difference between rural and urban isn't that big... Let's take a look at the coverage plots in the `./groot-plots` folder. For example:



coplot

We can see that the 5' and 3' ends of the genes have low coverage compared to the rest of each gene. The way GROOT works by default is to seed reads covering only the ARGs in the database. So reads that only partially cover a gene (at the 5' and 3' ends) are unlikely to seed. In low coverage samples, which are samples are likely to be, this means that we may not annotate some ARGs using the default settings.

We can try 2 things to report more ARGs:

- set the `--lowCov` flag for groot report -> this will report ARGs with no coverage in the first few bases at the 5' or 3' ends
- re-index the database using a shorter MinHash signature (`-s`) or lower Jaccard similarity threshold (`-j`) -> this will increase the chance of seeding reads that partially cover a gene

Option 2 takes a bit longer as we need to go all the way back in the workflow to indexing, so let's try option 1 for now:

```
ls *.bam | parallel --gnu "groot report --bamFile {} --lowCov > {/.}.lowCov.report"
while read -r line
do
    ID=$(echo $line | cut -f1 -d ' ')
    status=$(echo $line | cut -f2 -d ' ')
    if [ -s $ID.lowCov.report ]
        then
            cat $ID.lowCov.report >> combined-profiles.lowCov.$status.tsv
    fi
done < samples.txt
wc -l combined-profiles.lowCov.*
```

We've now ended up with a lot more ARGs being reported thanks to the `--lowCov` option; the downside is that we no longer have 100% length matches and we need to inspect the reports more closely. Here is an example report (SRR4454598):

The final column of the GROOT report output is a CIGAR-like representation of the covered bases: M=covered D=absent. The read count and coverage (plus the coverage plots) can help us assess the confidence in the reported genes when we use `--lowCov`. We can also check if reported ARGs share classified reads (i.e. multimappers). To compare reads aligning to 2 ARGs, you could try some commands like this:

```
# split the bam by ARG
bamtools split -in out.bam -reference
# extract reads aligning to specific ARGs
bedtools bamtofastq -i ARG1.bam -fq reads-aligned-to-ARG1.fq
bedtools bamtofastq -i ARG27.bam -fq reads-aligned-to-ARG27.fq
# find matching reads
bbduk.sh in=reads-aligned-to-ARG1.fq ref=reads-aligned-to-ARG27.fq outm=matched.fq␣
↪k=100 mm=f
```

Now let's try plotting the resistome profiles:

More to content to come...

2.4.5 5. ARG context

Now we have a set of ARGs we know are present in one or more samples, we might want to take a look at what is carrying these genes. We will use metacherchant to determine the genetic context of ARGs using subgraphs. We will also use Bandage and Kraken (not installed as part of our conda environment).

Download the full ARG-ANNOT set of genes and then index with samtools:

```
wget https://github.com/will-rowe/groot/raw/master/db/full-ARG-databases/arg-annot-db/
↪argannot-args.fna
samtools faidx argannot-args.fna
```

Now we can extract all the genes present in our resistome profiles:

```
samtools faidx argannot-args.fna `cut -f1 combined-profiles.urban.tsv` > urban-args.
↪fna
samtools faidx argannot-args.fna `cut -f1 combined-profiles.rural.tsv` > rural-args.
↪fna
```

Remove any duplicate genes:

```
cat urban-args.fna | seqkit rmdup -s -o urban-args.fna
cat rural-args.fna | seqkit rmdup -s -o rural-args.fna
```

As a first example, let's look at the genomic context of one ARG (`(Bla)OXA-347`) in urban samples. To begin, get the samples that contain this ARG:

```
grep -l "argannot~~~(Bla)OXA-347~~~JN086160:1583-2407" *.lowCov.report | sed 's/.
↪lowCov.report//' > samples-containing-blaOXA-347.list
```

We also need to store the reference sequence:

```
samtools faidx argannot-args.fna "argannot~~~(Bla)OXA-347~~~JN086160:1583-2407" >␣
↪blaOXA-347.fna
```

Now we can run metacherchant:

```
while read -r sample
    do
        metacherchant.sh --tool environment-finder \
        -k 31 \
        --coverage=2 \
        --maxradius 1000 \
        --reads ${sample}.fastq \
        --seq blaOXA-347.fna \
        --output "./metacherchant/${sample}/output" \
        --work-dir "./metacherchant/${sample}/workDir" \
            -p 42 \
            --trim
    done < samples-containing-blaOXA-347.list
```

To classify the contigs assembled around the detected ARGs, try Kraken:

```
while read -r sample
    do
        kraken --threads 8 --preload --fasta-input --db /path/to/minikraken_20141208␣
↪metacherchant/${sample}/output/seqs.fasta | kraken-report --db /path/to/minikraken_
↪20141208 > metacherchant/${sample}/${sample}-kraken.report
    done < samples-containing-blaOXA-347.list
```

* you will need to install kraken and download minikraken for this

To visualise the ARG context, first load the assembly graph into Bandage:

```
Bandage load metacherchant/SRR4454592/output/graph.gfa
```

Next,

More content to come soon. . .