Metagenomic analysis of Campylobacter at a wildlife–livestock–human interface in Uganda reveals novel species associated with human disease

AUTHORS
Valter Almeida
Matthew A. Knox
Kim M. Handley
Anne C. Midwinter
Patrick J. Biggs
Agata H. Dziegiel
Gladys Kalema-Zikusoka
Stephen Rubanga
Alex Ngabirano
Willy A. Valdivia-Granda
Hayley MacGregor
Nigel P. French
Alison E. Mather
James O. Lloyd-Smith
David T. S. Hayman

Supplementary Materials:

Bioinformatic analysis

Bioinformatic analyses were performed on New Zealand eScience Infrastructure (NeSI) high-performance computing platforms. The sequencing data generated in this study have been deposited in the NCBI Sequence Read Archive (SRA) under the BioProject accession code PRJNA1213876.

Quality control

Raw sequence reads were assessed for quality using FastQC (v0.11.9) to provide a preliminary quality report. This step was performed to identify potential issues, such as adapter contamination or low-quality bases, before trimming. Reads were trimmed to remove adapter sequences and low-quality bases using Trimmomatic (v0.39) to ensure high-quality data for downstream analysis. The key parameters used for trimming were:

- **PE**: Paired-end reads were processed.
- **ILLUMINACLIP**: Illumina adapter sequences were removed using a single mismatch tolerance (`1`), a minimum score of `30`, and a palindromic alignment length of `11`.
- **LEADING:10**: Bases with a quality score below `10` were removed from the start of a read.
- **TRAILING:10**: Bases with a quality score below `10` were removed from the end of a read.
- **MINLEN:80**: Reads shorter than `80` bases were discarded after trimming.
- **HEADCROP:10**: The first `10` bases were trimmed from the start of the read.

The quality of the trimmed reads was re-evaluated using FastQC (v0.11.9) to confirm the effectiveness of the trimming process and to ensure the data was suitable for further analysis.

Host read filtering

To remove host-derived reads and minimise contamination, all reads were mapped to their respective host reference genomes using BBMap (v38.95). Reference indexed files were built for each host:

- **Gorilla**: `GCF_008122165.1_Kamilah_GGO_v0_genomic.fna`
- **Human**: `GCF_000001405.39_GRCh38.p13_genomic.fna`
- **Cattle**: `GCF_002263795.1_ARS-UCD1.2_genomic.fna`
- **Goat**: `GCF_001704415.1_ARS1_genomic.fna`

The key parameters used for host filtering were:

- `minid=0.95`: Reads were filtered if they had a minimum identity of 95% to the host reference genome. This is a stringent threshold to ensure only reads highly likely to be from the host are removed.
- `maxindel=3`: The maximum allowed number of insertions or deletions was set to 3. This helps account for minor sequencing errors or genetic variations.
- `bwr=0.16` and `bw=12`: These parameters control the bandwidth, which influences how BBMap handles alignments with insertions or deletions, optimising performance.
- `quickmatch`: This parameter enables a faster, less sensitive alignment mode, which is suitable for high-stringency host filtering.
- `fast`: Another parameter that enables faster mapping at the cost of some sensitivity, which is acceptable when the goal is to remove highly similar host reads.
- `minhits=2`: Reads were required to have a minimum of two hits to the reference genome to be considered for filtering.
- `qtrim=rl` and `trimq=10`: Low-quality bases at the ends of reads were trimmed before mapping to improve alignment accuracy.
- `untrim`: This ensures that even if part of a read is trimmed, the entire original read is retained if it passes the filtering criteria.

An example of the script used for human samples is below:

```
module load BBMap/38.95-gimkl-2020a

for file in *_R1.trim.fastq.gz
  do
    base=$(basename ${file} _R1.trim.fastq.gz)
    srun bbmap.sh in1=${file} in2=${base}_R2.trim.fastq.gz -Xmx27g -t=20 \
    minid=0.95 maxindel=3 bwr=0.16 bw=12 quickmatch fast minhits=2 qtrim=rl trimq=10
```

```
untrim \

path=/nesi/project/massey03212/AGRF/AGRF_CAGRF21098330_H3CGTDSX3/Trimmomatic/bbmap/Hum
an_ref \

out1=/nesi/project/massey03212/AGRF/AGRF_CAGRF21098330_H3CGTDSX3/Trimmomatic/bbmap/Hum
an_community/host_filtered_reads/${base}_R1_hostFilt.fastq.gz
out2=/nesi/project/massey03212/AGRF/AGRF_CAGRF21098330_H3CGTDSX3/Trimmomatic/bbmap/Hum
an_community/host_filtered_reads/${base}_R2_hostFilt.fastq.gz
done
```

Metagenomic assembly

Following host-read filtering, the remaining reads were assembled into contigs using MEGAHIT (v1.2.9). Both individual sample assemblies and co-assemblies were performed to maximise the recovery of high-quality metagenome-assembled genomes (MAGs).

The key parameters used for assembly were:

- **-1 and -2**: These parameters specified the paired-end read files for the forward and reverse reads.
- **-o**: This defined the output directory for the assembly results.

To automate the assembly of each individual sample, a SLURM array job was used. This approach allowed for the parallel processing of multiple samples, with each array task (`$SLURM_ARRAY_TASK_ID`) processing a single sample's paired-end reads.

An example of the script used for a single sample assembly via the array job is shown below:

```
module load MEGAHIT/1.2.9-gimkl-2022a-Python-3.10.5

# Get the list of FASTA files and select the one corresponding to the current array
job index
fastq_files=("$INPUT_DIR"/*_R1_hostFilt.fastq.gz)
fastq_file_R1="${fastq_files[$SLURM_ARRAY_TASK_ID]}"
fastq_file_R2="${fastq_file_R1/_R1/_R2}"

# Extract the base name of the file (without the directory and extension)
base_name=$(basename "$fastq_file_R1" _R1_hostFilt.fastq.gz)

# Run MEGAHIT
megahit -1 ${fastq_file_R1} -2 ${fastq_file_R2} -o
${OUTPUT_DIR}/${base_name}_megahit_out
```

Contigs with a length of less than 1,000 bp were filtered out using seqmagick (v0.8.4). This step was performed to remove short, potentially erroneous contigs and improve the quality of the final assembly.

The key parameters used were:

- `convert`: This subcommand was used to perform a conversion on the input file.
- `--min-length 1000`: This parameter specified that only contigs with a minimum length of 1,000 base pairs were retained.

An example of the script used is shown below:

```
module load seqmagick/0.8.4-gimkl-2020a-Python-3.8.2

seqmagick convert --min-length 1000 final.contigs.fa Hc.m1000.fna
```

Read mapping and sorting

Following contig filtering, the host-filtered reads were mapped back to their corresponding assembled contigs. This was a two-step process using Bowtie2 (v.2.5.4) and SAMtools (v.1.9).

First, a Bowtie2 index was built for each sample's filtered contigs.

```
module load Bowtie2/2.5.4-GCC-12.3.0

bowtie2-build --threads $SLURM_CPUS_PER_TASK "samplename".m1000.fna "samplename"_bt2
```

Next, reads were mapped to the indexed contigs. The key parameters used for mapping were:

- `--minins 200` and `--maxins 800`: These parameters set the minimum and maximum insert sizes for valid paired-end alignments, ensuring that only reads from appropriately sized fragments were considered.
- `--sensitive`: This preset was used to ensure high mapping sensitivity, optimising for a high alignment rate.
- `--threads $SLURM_CPUS_PER_TASK`: This specified the number of threads to use, enabling parallel processing.
- `-x`: This parameter specified the name of the Bowtie2 index to use for mapping.
- `-1` and `-2`: These specified the paired-end read files.
- `-S`: The output file was saved in SAM format, which was then converted to BAM.

```
module load Bowtie2/2.5.4-GCC-12.3.0

bowtie2 --minins 200 --maxins 800 --threads $SLURM_CPUS_PER_TASK --sensitive -x Hc_bt2
-1 Hc_R1_hostFilt.fastq.gz -2 Hc_R2_hostFilt.fastq.gz -S Hc.bam
```

Finally, the output BAM file was sorted and indexed using SAMtools (v1.9) to prepare it for subsequent analyses, such as abundance estimation and genome binning.

```
module load SAMtools/1.9-GCC-7.4.0

samtools sort -@ $SLURM_CPUS_PER_TASK Hc.bam -o Hc.sorted.bam
samtools index -@ $SLURM_CPUS_PER_TASK Hc.sorted.bam
```

Following read mapping, contigs were binned into MAGs using CONCOCT, MetaBAT2, and MaxBin2. A consensus approach was taken by running all three binning tools independently to maximize the recovery of high-quality bins.

CONCOCT (v1.1.0) was used to cluster contigs into bins based on both coverage and composition. The pipeline involved the following steps:

- `cut_up_fasta.py`: This script was used to split the assembled FASTA files into 10 kb fragments. This fragmentation is a required step for CONCOCT's binning algorithm.
- `concoct_coverage_table.py`: This script created a coverage table from the sorted BAM files, which is used by CONCOCT to determine the abundance of each contig fragment across the samples.
- `concoct`: The main CONCOCT program was then run using the composition and coverage files to perform the binning.
- `merge_cutup_clustering.py`: This script merged the clustered fragments back into full-length contigs.
- `extract_fasta_bins.py`: This final script extracted the full-length contigs for each bin into separate FASTA files.

```
module load CONCOCT/1.1.0-gimkl-2020a-Python-3.8.2

# Example script for a single sample
cut_up_fasta.py "sample_name.m1000.fna" -c 10000 -o 0 --merge_last -b
"sample_name_10K.bed" > "sample_name_10K.fa"
concoct_coverage_table.py "sample_name_10K.bed" "sample_name.sorted.bam" >
"coverage_table_sample_name.tsv"
concoct --composition_file "sample_name_10K.fa" \
        --coverage_file "coverage_table_sample_name.tsv" \
        -b "concoct_output/sample_name_concoct_output/"
merge_cutup_clustering.py
"concoct_output/sample_name_concoct_output/clustering_gt1000.csv" > \

"concoct_output/sample_name_concoct_output/clustering_merged.csv"
extract_fasta_bins.py "sample_name.m1000.fna" \

"concoct_output/sample_name_concoct_output/clustering_merged.csv" \
                        --output_path
"concoct_output/sample_name_concoct_output/fasta_bins"
```

MetaBAT2 (v2.17) was used to bin contigs based on their coverage and tetranucleotide frequencies. Read depth for each contig was first calculated from the sorted BAM file.

```
module load MetaBAT/2.17-GCC-12.3.0

# Calculate contig depth
jgi_summarize_bam_contig_depths --outputDepth "metabat_depth.txt"
"sample_name.sorted.bam"

# Run MetaBAT2
metabat2 -t $SLURM_CPUS_PER_TASK -m 1500 \
         -i "sample_name.m1000.fna" \
         -a "metabat_depth.txt" \
         -o "metabat_bins/sample_name_metabat"
```

The key parameters were:

- **-m 1500**: This option set the minimum contig length to 1,500 bp, ensuring that only longer, more reliable contigs were included in the binning process.
- **-i**: The input file of contigs to be binned.
- **-a**: The file containing contig abundance and depth information.
- **-o**: The base output name for the generated bins.

MaxBin2 (v2.2.7) was used as a third binning method. The contig abundance file from the MetaBAT2 step was prepared for use with MaxBin2.

```
module load MaxBin/2.2.7-GCC-11.3.0-Perl-5.34.1

# Prepare abundance file
cut -f1,4,6,8,10 "metabat_depth.txt" > "maxbin_abundance.txt"

# Run MaxBin2
run_MaxBin.pl -thread $SLURM_CPUS_PER_TASK -min_contig_length 1500 \
              -contig "sample_name.m1000.fna" \
              -abund "maxbin_abundance.txt" \
              -out "maxbin_bins/sample_name_maxbin"
```

The key parameters were:

- **-min_contig_length 1500**: This parameter set the minimum contig length to 1,500 bp, matching the threshold used in MetaBAT2.
- **-contig**: The input file containing the assembled contigs.
- **-abund**: The file containing contig abundance data.
- **-out**: The base output name for the generated bins.

To consolidate the results from the different binning algorithms and obtain a high-quality set of non-redundant bins, DAS Tool (v1.1.5) was used. This process involved two main steps: first, creating a file that associated each contig with its corresponding bin for each binning method; and second, running DAS Tool to select the best bins based on quality metrics like completeness and contamination.

For each binning method, a text file was created that associated every contig with its respective bin. An example of the commands used for MetaBAT2 is provided below. The same approach was applied to MaxBin2 and CONCOCT.

```
for bin_path in metabat/*.fa;
do
    bin_name=$(basename ${bin_path} .fa)

    grep ">" ${bin_path} | sed 's/>//g' | sed "s/$/\t${bin_name}/g" >>
metabat_associations.txt
done
```

The association files from all three binning methods were then used as input for DAS Tool. The software selected the best representative bins from each set, resulting in a high-quality, consolidated set of MAGs.

The key parameters used were:

- `-i`: This parameter specified the input association files from the different binning tools.
- `-l`: This provided labels for each of the input binning methods (e.g., MetaBAT, MaxBin, Concoct).
- `-t`: This set the number of threads for parallel processing.
- `--write_bins`: This option generated the final consolidated bins as a new set of FASTA files.
- `-c`: This specified the input contigs file, which was used by DAS Tool to determine bin quality.
- `-o`: This defined the output directory for the final results.

```
module load DAS_Tool/1.1.5-gimkl-2022a-R-4.2.1

# Run DAS_Tool
DAS_Tool -i metabat_associations.txt,maxbin_associations.txt,concoct_associations.txt
\
         -l MetaBAT,MaxBin,Concoct \
         -t 2 --write_bins --search_engine diamond \
         -c all_bins/Cattle.m1000.fna \
         -o dastool_out/
```

To assess the quality of the final consolidated bins, CheckM2 (v1.0.1) was used. This software predicts the completeness and contamination of the MAGs, as well as other general characteristics such as genome size and GC content, which are key metrics for determining their quality.

The key parameters used were:

- `predict`: This subcommand was used to predict the quality of the bins.

- `--threads` : The number of threads used for parallel processing.
- `--extension` : This specified the file extension of the input bins (e.g., `.fa`).
- `--input` : The directory containing the input bin files.
- `--output-directory` : The directory where the CheckM2 results were saved.
- `--force` : This option forced the analysis to overwrite any existing output.

```
module load CheckM2/1.0.1-Miniconda3

# Run CheckM2 to assess MAG quality
checkm2 predict \
    --threads $SLURM_CPUS_PER_TASK \
    --extension fa \
    --input "Campylobacter_sp900" \
    --output-directory "checkm_out" \
    --force > checkm_results.tsv
```

Then, dRep (v3.4.2), Mash (v.2.3) and CheckM2 (v.1.0.1) were used to dereplicate the bins and create a final dataset of unique MAGs. This step is crucial for removing redundant or highly similar genomes from the combined binning results. The *Campylobacter* MAGs obtained from this step were then selected for downstream analysis.

The key parameters used for this process were:

- `-g` : This specified the directory containing all the input FASTA files from the binning and consolidation steps.
- `-p` : This set the number of CPU threads for parallel processing.
- `--genomeInfo` : This provided a file containing genome quality metrics (e.g., from CheckM2), which dRep uses to select the highest-quality genome from a cluster of similar genomes.
- `-str 100` : This specified that all genomes were to be compared, regardless of whether they belonged to the same species, by setting the primary clustering strategy to `100` (meaning, `100%` of genomes should be considered in the all-vs-all comparison).
- `-pa 0.95` : The primary average nucleotide identity (ANI) threshold was set to `95%` . Genomes were clustered if their ANI was above this threshold.
- `-sa 0.99` : The secondary ANI threshold was set to `99%` for final clustering, ensuring that the final representative MAGs were at least `99%` identical to their cluster members.

```
module load CheckM2/1.0.1-Miniconda3
module load drep/3.4.2-gimkl-2022a-Python-3.10.5
module load Mash/2.3-GCC-12.3.0

dRep dereplicate dRep_output/ -p $SLURM_CPUS_PER_TASK --genomeInfo dRep.genomeInfo -g
../03_Quality_MAGs/*.fa -str 100 -pa 0.95 -sa 0.99
```

Publicly available genome download

Publicly available *Campylobacter* genomes were downloaded from NCBI to serve as a reference dataset for phylogenetic and comparative genomic analyses. This was a multi-step process.

First, the latest bacterial taxonomy file was downloaded from the GTDB database. Then, a script was used to select a single accession for each unique *Campylobacter* species (**s__**), which ensured that the final dataset contained only one representative genome per species.

```
# Download the latest GTDB bacterial taxonomy metadata file
wget
https://data.ace.uq.edu.au/public/gtdb/data/releases/latest/bac120_taxonomy.tsv.gz
gunzip bac120_taxonomy.tsv.gz

# Select one genome per species from the GTDB taxonomy file
awk -F'\t|;' '$8 ~ /^s__Campylobacter(_[A-Z]+)? / {if (!seen[$8]++) print $1}'
bac120_taxonomy.tsv | sed 's/^[A-Z]*_//' > 02_Campylobacter_accession_references.txt

# Create a directory to store the downloaded genomes
mkdir -p downloaded_genomes

# Loop through each accession and download the genome from NCBI's FTP server
list_file="02_campylobacter_accession_references.txt"
while read accession; do

url="https://ftp.ncbi.nlm.nih.gov/genomes/all/${accession:0:3}/${accession:4:3}/${acce
ssion:7:3}/${accession:10:3}/"
    filename=$(curl -s "$url" | grep -oP "href=\"${accession}.*?_genomic.fna.gz\"" |
head -n 1 | cut -d '"' -f 2)
    if [ ! -z "$filename" ]; then
        wget "${url}/${filename}" -O "downloaded_genomes/${accession}_genomic.fna.gz"
    else
        echo "No assembly found for accession: $accession"
    fi
done < "$list_file"

# Clean up temporary files
rm temp*
rm *.html
gunzip downloaded_genomes/*.fna.gz

# Remove headers and ensure correct FASTA format for each downloaded genome
for i in downloaded_genomes/*.fna; do
    file_base=$(basename "${i}" .fna)
    sed 's/ .*//g' "${i}" > "downloaded_genomes/${file_base}.fa"
done
```

Taxonomic classification

Then, GTDB-Tk (v2.4.0) was used for taxonomic classification and multiple sequence alignment.

The key parameters used for this analysis were:

- `classify_wf`: This subcommand initiated the main workflow for classifying genomes and performing multiple sequence alignment.
- `-x fa`: This specified the input file format as FASTA.
- `--cpus`: This set the number of CPU threads for parallel processing.
- `--skip_ani_screen`: This parameter was used to skip the average nucleotide identity (ANI) screening step, which can speed up the classification process.
- `--genome_dir`: This provided the path to the directory containing all the input MAGs and reference genomes.
- `--out_dir`: This defined the output directory for the results.

```
module load GTDB-Tk/2.4.0-foss-2023a-Python-3.11.6

gtdbtk classify_wf -x fa --cpus $SLURM_CPUS_PER_TASK --skip_ani_screen --genome_dir /n
```

After taxonomic classification and multiple sequence alignment, a phylogenetic tree was inferred using IQ-TREE2 (v2.2.2.2). Branch support was assessed with 1000 ultrafast bootstrap replicates and the phylogenetic tree was visualised and annotated using the Interactive Tree of Life (iTOL v6).

The key parameters used were:

- `-s`: This specified the input multiple sequence alignment file.
- `-T`: This set the number of CPU threads for parallel processing.
- `-mem 8GB`: This allocated 8 GB of memory for the analysis.
- `-m TEST`: This option automatically selected the best-fit substitution model for the data, which is crucial for accurate phylogenetic inference.
- `-b 1000`: This performed 1,000 ultrafast bootstrap replicates to assess the robustness of the tree branches.

```
module load IQ-TREE/2.2.2.2-gimpi-2022a

iqtree2 -s gtdbtk.bac120.user_msa.fasta.gz -T $SLURM_CPUS_PER_TASK -mem 8GB -m TEST -b
1000
```

Genome coverage and abundance normalisation

Genome coverage and relative abundance were determined by mapping host-filtered reads to the final set of unique MAGs. This process involved read mapping, sorting, coverage calculation, and subsequent normalisation.

Read mapping and read counting

Host-filtered reads were first mapped to the final set of MAGs using Bowtie2 (v2.5.4). The resulting alignment files were then converted to BAM format, sorted, and indexed using SAMtools (v1.21).

Once the BAM files were sorted, MetaBAT2 (v2.15) was used to calculate the per-sample coverage statistics for each contig within the MAGs. The number of mapped reads was also extracted using SAMtools.

An example of the combined script used for this process is below:

```
module load Bowtie2/2.5.4-GCC-12.3.0
module load SAMtools/1.21-GCC-12.3.0
module load MetaBAT/2.15-GCC-11.3.0

# Define a single sample name for clarity in this example
sample_name="UG_DNA_sampleA"

# Map reads to MAGs using Bowtie2 and pipe the output to SAMtools for conversion and
sorting
# '-S -' tells samtools to expect input from a pipe
bowtie2 -x "campy_index" -1 "${sample_name}_R1.fastq.gz" -2
"${sample_name}_R2.fastq.gz" | \
samtools view -bS - > "${sample_name}.bam"

# Sort the BAM file
samtools sort -o "${sample_name}.sorted.bam" "${sample_name}.bam"

# Remove the unsorted BAM file to save space
rm "${sample_name}.bam"

# Calculate contig depth with MetaBAT2's helper script
jgi_summarize_bam_contig_depths --outputDepth bins_cov_table.txt mapped/*.bam
```

The relative abundance of each *Campylobacter* species was then calculated by normalising the number of reads mapped to their best-quality MAGs (and all unique MAGs per species) by the total sequencing depth of each sample and the length of the corresponding MAGs.

Genome coverage normalisation

Following the initial coverage and abundance calculations, a custom Python script, **norm_jgi_cov_table.py**, was used to normalise the coverage values. This script corrected for variations in sequencing depth between samples, ensuring that abundance comparisons were robust.

The script normalises the average fold coverage for contigs based on the output from `jgi_summarize_bam_contig_depths`. It achieves this by first retrieving the per-sample read depths from the `bowtie2` standard error output and then adjusting the coverage values accordingly. This produces a final, normalised coverage table suitable for downstream analysis.

Script: `norm_jgi_cov_table.py`

Python

```python
#!/usr/bin/env python

# norm_jgi_cov_table.py
## Calculate normalised average fold coverage for contigs based on output from
jgi_summarize_bam_contig_depths
## Per-sample read depths are retrieved from the standard error output of the prior
bowtie2 read mapping

# Required parameters:
## --cov_table (-c) cov_table.txt : coverage table output from
jgi_summarize_bam_contig_depths
## --bowtie2_err (-e) mapping.err : output file of captured standard error output from
bowtie2 read mapping

# Optional parameters:
## --output_path (-o) output_directory : Path for output directory. Default = './'

####################
# Importing libraries and setting input arguments
####################

# import packages
from argparse import ArgumentParser
import os
import pandas as pd
import numpy as np
import re

# Set arguments
parser = ArgumentParser()
parser.add_argument("-c", "--cov_table", dest="cov_table_in",
                    help="Coverage table output from jgi_summarize_bam_contig_depths",
                    metavar='cov_table', required=True)
parser.add_argument("-e", "--bowtie2_err", dest="bowtie2_err_in",
                    help="Output file of captured standard error output from bowtie2
read mapping",
                    metavar='bowtie2_err', required=True)
parser.add_argument("-o", "--output_path", dest="out_path",
                    help="Path/to/output/directory/. Default = current directory",
                    metavar='output_path', default='./')
args = parser.parse_args()

####################
# Main script
####################

print("\nRunning norm_jgi_cov_table.py\r\n")

## Calculate average reads/sample (library size)

# Generate empty list of read counts and add results for each sample via looping over
the relevant lines in bowtie2_err
```

```
read_counts = []
with open(args.bowtie2_err_in, 'r') as readfile:
    for line in readfile:
        if "reads; of these:" in line:
            sample_read_count = int(re.sub(r'(\d+).*', r'\1', line))
            read_counts.append(sample_read_count)

# Calculate average library size
avg_read_counts = round(sum(read_counts) / len(read_counts))

## Calculate normalised coverage per sample and generate normalised count table

# covstats.txt files
# files = [entry.path for entry in filepath if '.covstats.txt' in entry.name]

# Read in coverage table
cov_table = pd.read_csv(args.cov_table_in, sep='\t')

# For each column that ends in .bam (the coverage values for each sample), normalise
values by: coverage value / sample read count (from read_counts list) *
avg_read_counts
i=0
for column in cov_table.columns[cov_table.columns.str.endswith('bam')]:
    cov_table.loc[:, column] = round((cov_table.loc[:, column] / read_counts[i]) *
avg_read_counts, 4)
    i += 1

# Subset cov_table to retain only the columns of interest
cov_table = cov_table.loc[:, (['contigName', 'contigLen'] +
cov_table.columns[cov_table.columns.str.endswith('bam')].tolist())]

# Write output table to file
cov_table.to_csv(args.out_path+'normalised_'+args.cov_table_in, sep='\t', index=False)

print("Output:\r\n")
print(args.out_path+'normalised_'+args.cov_table_in+' : coverage table output from
jgi_summarize_bam_contig_depths, normalised by average sample read depth.\r\n')
print("Completed norm_jgi_cov_table.py\r\n")

####################
# END
####################
```

Gene prediction and functional annotation

Gene prediction was carried out using Prodigal (v2.6.3), which identified protein-coding genes in the assembled MAGs. Prokka (v1.14.5) was used as a rapid annotation tool that generates GFF files, which were subsequently used for downstream analyses. The `groL` gene from our annotated MAGs and publicly available reference genomes was aligned using MAFFT (v7.450). This multiple sequence alignment was then imported into Geneious (v10.6.2). Primers from a previous study were mapped to the `groL` alignment using Geneious's live annotation tool. This process was performed in silico to test for the cross-reactivity of the primers with the `groL` gene across the diverse *Campylobacter* species in our dataset.

Prodigal was run on each MAG to predict genes, creating a gene file in FASTA format (`.fna`), an amino acid file in FASTA format (`.faa`), and a GenBank format file (`.gbk`). The `-p single` parameter was used to ensure that the gene prediction was optimised for single, un-concatenated genomes.

```
module load prodigal/2.6.3-GCCcore-7.4.0

# Working directory
cd
/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/10_Metabolic_functions/03_gene_p
rediction

# Output directory
mkdir -p predictions/

# List of all MAGs (full paths)
bin_list=
(/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/10_Metabolic_functions/Mags_ref
s/*.fa)

# Get the file for this SLURM array task
bin_file=${bin_list[$SLURM_ARRAY_TASK_ID]}
pred_file=$(basename "$bin_file" .fa)

# Run prodigal
prodigal -i "$bin_file" -p single \
         -d predictions/"${pred_file}".genes.fna \
         -a predictions/"${pred_file}".genes.faa \
         -o predictions/"${pred_file}".genes.gbk
```

The following Prokka script was used to automate the annotation of each FASTA file.

```
module load prokka/1.14.5-GCC-11.3.0
module load tbl2asn/20230926

INPUT_DIR="/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/10_Metabolic_function
s/Mags_refs"
OUTPUT_DIR="/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/10_Metabolic_functio
ns/02_Prokka"

mkdir -p "$OUTPUT_DIR"

for file in "${INPUT_DIR}"/*.fa; do
    base=$(basename "${file}" .fa)
    outdir="${OUTPUT_DIR}/${base}_output"

    prokka "${file}" \
        --prefix "${base}" \
        --outdir "${outdir}" \
        --cpus "$SLURM_CPUS_PER_TASK"
done
```

Comparative genomics

Average Nucleotide Identity (ANI)

ANI comparisons between all MAGs and publicly available genomes were performed with FastANI (v1.33). This tool provides a measure of genomic similarity, with values above 95% often used to delineate species boundaries. The following parameters were used to ensure a comprehensive all-vs-all comparison:

- `--minFraction 0.1`: This required a minimum of 10% of the query genome to have a match in the reference genome, which helped to filter out false alignments.
- `--fragLen 1000`: This set the fragment length for alignment to 1,000 bp, which is an optimal length for accurate ANI calculation.

```
module load FastANI/1.33-intel-2022a

# Paths
GENOME_DIR="/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/02_gtdbtk/new_datase
t_corrected"
OUTPUT="fastani_results.tsv"
GENOME_LIST="genome_list.txt"

# Generate genome list with absolute paths
find "$GENOME_DIR" -type f \( -name "*.fa" -o -name "*.fna" -o -name "*.fasta" \) -
exec realpath {} \; > "$GENOME_LIST"

# Read genomes into array
mapfile -t GENOMES < "$GENOME_LIST"
NUM_GENOMES=${#GENOMES[@]}

# Create output directory for temp files
mkdir -p tmp_results

# Main ANI calculation loop
for (( i=0; i<NUM_GENOMES; i++ )); do
  for (( j=i; j<NUM_GENOMES; j++ )); do
    OUT_FILE="tmp_results/ani_${i}_${j}.txt"
    if ! fastANI -q "${GENOMES[$i]}" \
                 -r "${GENOMES[$j]}" \
                 --minFraction 0.1 \
                 --fragLen 1000 \
                 -o "$OUT_FILE" 2>> fastani_errors.log
    then
      echo "ERROR comparing ${GENOMES[$i]} vs ${GENOMES[$j]}" >> fastani_errors.log
      touch "$OUT_FILE"
    fi
  done
```

```
done

# Combine results
cat tmp_results/*.txt > "$OUTPUT"
```

Virulence and antibiotic resistance gene analysis

To identify genes conferring antibiotic resistance and virulence, Abricate (v1.0.0) was used. This tool screens contigs for the presence of target genes from various databases, providing a rapid and comprehensive assessment. The analysis was conducted against the ResFinder database for antibiotic resistance genes and the VFDB (Virulence Factor Database) for virulence genes.

The key parameters used for this process were:

- `--db`: This specified the database to be used for the screen.
- `*.fa`: This wildcard was used to process all FASTA files in the input directory.

```
module load ABRicate/1.0.0-GCC-11.3.0-Perl-5.34.1

# Set working and output directories
WORKDIR=/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/06_Abricate/Campylobacte
r_infans
OUTDIR=/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/06_Abricate/Campylobacter
_infans

# Create output directory if it doesn't exist
mkdir -p "$OUTDIR"

# Move to working directory
cd "$WORKDIR"

# Run ABRicate and save outputs to OUTDIR
abricate --db resfinder *.fa > "$OUTDIR/resfinder.tsv"
abricate --db vfdb *.fa > "$OUTDIR/vfdb.tsv"
```

Comparative alignments to the *C. jejuni* reference genome (GTDB ID GCA_000009085.1; NCTC11168) were conducted using MUMmer (v4.0). This pipeline was designed to identify, filter, and combine shared regions between the MAGs and the reference genome.

Initially, `dnadiff` was used to perform a whole-genome alignment between the MAGs and the reference genome. The resulting delta file was then filtered to retain only the best-filtered alignments, removing redundant or poor-quality matches. The `show-coords` command was used to generate a tabular file summarising the alignment coordinates.

Example:

```
## MUMmer - Pipeline:

dnadiff -p C_vicugnae_vs_cjejuni GCA_000009085.1_C_jejuni_genomic.fa
```

```
maxbin.030.go30.fa

show-coords -rclTH C_vicugnae_vs_cjejuni.delta > C_vicugnae_vs_cjejuni.coords.tab

delta-filter -1 C_vicugnae_vs_cjejuni.delta > C_vicugnae_vs_cjejuni.filtered.delta

show-coords -rclTH C_vicugnae_vs_cjejuni.filtered.delta > filtered.coords.tab
```

To obtain the full aligned sequences, the filtered delta file was processed to extract longer mapped regions using `show-aligns`. The aligned sequences from this step were then combined into a single alignment per genome pair using a custom AWK script (`combine_alignments.awk`).

```
## Longer mapped regions:

#!/bin/bash

DELTA_FILE="C_vicugnae_vs_cjejuni.filtered.delta"

grep "^>" "$DELTA_FILE" | cut -d' ' -f1,2 | sed 's/^>//' | while read ref query; do
    echo "Processing $ref vs $query"
    outname="${ref}_vs_${query}.txt"
    show-aligns "$DELTA_FILE" "$ref" "$query" > "$outname"
done

## To combine mapped sequences:

mkdir -p combined_sequences

# combine_alignments.awk
BEGIN {
    RS = "-- BEGIN alignment";
    FS = "\n";
    ORS = "";
    combined_ref = "";
    combined_query = "";
}
{
    ref_seq = "";
    query_seq = "";
    for (i = 3; i <= NF; i++) {
        if ($i ~ /^[0-9]/) {
            split($i, parts, / +/);
            ref_line = parts[2];
            i++;
            split($i, parts, / +/);
            query_line = parts[2];
            for (j = 1; j <= length(ref_line); j++) {
                ref_char = substr(ref_line, j, 1);
                query_char = substr(query_line, j, 1);
                query_seq = query_seq (query_char == "." ? ref_char : query_char);
            }
            ref_seq = ref_seq ref_line;
        }
    }
    combined_ref = combined_ref ref_seq;
    combined_query = combined_query query_seq;
}
END {
    print "Combined Reference (" ref_name "):\n" combined_ref "\n";
    print "Combined Query (" query_name "):\n" combined_query "\n";
}

## Run the script on all files:

#!/bin/bash

# Directory containing alignment files
ALIGNMENTS_DIR="combined_sequences"

# Output directory
OUTPUT_DIR="combined_sequences"

# Process each file
for file in "$ALIGNMENTS_DIR"/AL111168.1_vs_*.txt; do
    # Extract reference and query names from the filename
    base=$(basename "$file" .txt)
    ref_name="AL111168.1"
    query_name=$(echo "$base" | cut -d'_' -f4- | sed 's/_vs_//')
    # Run the AWK script with variables for ref/query names
    awk -v ref_name="$ref_name" -v query_name="$query_name" \
        -f combine_alignments.awk "$file" > "$OUTPUT_DIR/${base}_combined.txt"

    echo "Processed: $file -> $OUTPUT_DIR/${base}_combined.txt"
done
```

Metabolic Analysis

Metabolic analysis was performed using DRAM (Distillate and Annotate a Metagenome) (v1.3.5) to provide a detailed overview of the metabolic potential of each *Campylobacter* MAG. The `distill` subcommand was used to generate a concise summary of the metabolic functions identified in the annotated MAGs.

The key parameters used for this step were:

- `-i`: The input annotation file, which contained the predicted gene annotations for the MAGs.
- `-o`: The output directory for the distilled results.
- `--trna_path`: The path to the tRNA annotation file.
- `--rrna_path`: The path to the rRNA annotation file.

```
module load DRAM/1.3.5-Miniconda3

cd
```

```
/nesi/nobackup/massey03212/Nat_comms/Genomic_analysis/10_Metabolic_functions/04_gene_a
nnotation_and_coverage/C_infans_C_sp900_DRAM

# Prokaryote annotations
DRAM.py distill -i dram_annotations/annotations.tsv \
                -o dram_distillation \
                --trna_path dram_annotations/trnas.tsv \
                --rrna_path dram_annotations/rrnas.tsv
```

Read-based k-mer Taxonomic Classification

For an additional, independent taxonomic classification of the metagenomic reads, Kraken2 (v2.1.3) and Bracken (v2.7) were used. This approach provided a more nuanced view of the microbial community composition by identifying organisms at a species level. Kraken2 first classified reads against the standard database (12/2024), and Bracken then re-estimated the abundance of each taxon to correct for shared sequences.

The key parameters used for Kraken2 were:

- `--db`: The path to the Kraken2 database.
- `--threads`: The number of CPU threads used for processing.
- `--use-names`: This option was used to display the full taxonomic names in the output.
- `--output`: The path for the main output file.
- `--report`: The path for the classification report.

The key parameters used for Bracken were:

- `-db`: The path to the Kraken2 database used by Bracken.
- `-i`: The Kraken2 report file.
- `-l S`: This specified that Bracken should estimate abundance at the species level.
- `-o`: The output path for the Bracken report.
- `-t 10`: A confidence threshold for Bracken.
- `-r 100`: This set the minimum number of reads required for a taxon to be included in the report.

```
module load Kraken2/2.1.3-GCC-11.3.0
module load Bracken/2.7-GCC-11.3.0

# Set input and output directories
INPUT_DIR="/nesi/nobackup/massey03212/Nat_comms/orion_fastq"
OUTPUT_DIR="/nesi/nobackup/massey03212/Nat_comms/Kraken2"

# Create an array of paired files
declare -A file_pairs
for file in "$INPUT_DIR"/*_R1.fastq
```

```
do
  base_name=$(basename "$file" _R1.fastq)
  paired_file="$INPUT_DIR/${base_name}_R2.fastq"
  if [[ -f "$paired_file" ]]; then
    file_pairs["$file"]="$paired_file"
  fi
done

# First step: Kraken2
for file1 in "${!file_pairs[@]}"
do
  file2=${file_pairs[$file1]}
  base1=$(basename "$file1" .fastq)
  base2=$(basename "$file2" .fastq)
  kraken2 --db /nesi/nobackup/massey03212/Kraken2_DB_Standard_12_24/Standard_12_2024 -
-threads $SLURM_CPUS_PER_TASK --use-names "$file1" "$file2" --output
"$OUTPUT_DIR/${base1}_${base2}.txt" --report "$OUTPUT_DIR/${base1}_${base2}.kreport"
done

# Second step: Bracken
for file in "$OUTPUT_DIR"/*.kreport
do
  base=$(basename "$file" .kreport)
  bracken -db /nesi/nobackup/massey03212/Kraken2_DB_Standard_12_24/Standard_12_2024 -i
"$file" -l S -o "$OUTPUT_DIR/${base}_bracken.report" -t 10 -r 100
done
```

Statistical Analysis

Statistical analyses were conducted in **R (v4.5.1)**. These analyses included a one-way analysis of variance (ANOVA) to compare the means of different groups, followed by Tukey’s honestly significant difference (HSD) test to perform pairwise comparisons between group means.