ONE ARM Metagenomics Pipeline

Prepared for and by: Project 4

Last updated September 19, 2024

Contents

Part I

Project overview

Metagenomic analysis of Antibiotic/Antimicrobial Resistance Genes (ARGs) in NCR (Metro Manila) hospitals, wastewaters, and surface waters.

- ✓ Done
- ☐ Pending
- ⟳ Needs refinement
- △ Unexpected issues
- ✎ Drafted
- ➔ Moved

- Essential
- Optional
- Robust
- ~~Deprecated~~

1 HPC Preparation

- ☐ Confirmation of Robustness of analysis
  - ☐ Agree upon the type of analyses
- ☐ `SLURM` request management
  - ☐ Send email for HPC usage requests
  - ☐ Setup Docker containers for `SLURM`
  - ☐ `SLURM` container for simulations
  - ☐ Automation of `SLURM` requests

File Management

- ☐ `BAM` file parsing
- ☐ Interconversion between `SAM` and `BAM`

Raw Read Processing

- ☐ Data visualization
- ☐ Determination of optimal tool/s
- ☐ Determination of optimal parameters
- ☐ Tool combination randomization script

Assembly & Binning

- ☐ Testing binning pipeline

Creation of assembly scripts

- ☐ metaSPAdes
- ☐ MASURCA
- ☐ PLASS
- ☐ AbySS

Testing of Assembly Scripts

- ☐ metaSPAdes
- ☐ MASURCA
- ☐ PLASS
- ☐ AbySS
- ☐ KMA-iterative **Further investigation of scripts**
- ☐ Benchmarking between all of them
- ✎ Iterative assembly
- ✎ Contig quality checking **Pipeline integration**
- △ `MetaWrap` binning
- ☐ Testing on datasets of higher depth for MetaWrap tests
- ☐ Refinement of bins

Annotation

ARG-related scripts

- ⟳ RGI
- ⟳ ShortBRED
- ⟳ AMRFinder

Testing of Annotation Scripts

- ☐ RGI
- ☐ ShortBRED
- ☐ AMRFinder

Taxonomy releated scripts

- ☐ MetaPhlan4

Complete

General Taxo-Metagenomics Pipeline

Creation of modular scripts

- ✓ FasQC script
- ✓ Trimmomatic script
- ✓ Kraken2 script
- ✓ Bracken script
- ✓ In-house diversity calcs script

Test Run of Modular Scripts

- ✓ FasQC script
- ✓ Trimmomatic script
- ✓ Kraken2 script
- ✓ Bracken script
- ✓ In-house diversity calcs script

Other Completed Pipeline Tasks

- ✓ Integration with config folder
- ✓ Testing with simulated datasets
- ✓ Pipeline integration (metagenomics_general.smk)
  - ✓ FastQC
  - ✓ Trimmomatic
  - ✓ Kraken2
  - ✓ Bracken
  - △ ~~QIIME2~~
  - ✓ Change QIIME2 to in-house diversity script
  - ✓ Aggregate diversity metrics
- ✓ Testing pipeline with simulated datasets

File Management

- ✓ Compression and decompression scripts
- ✓ Demultiplexing script
- ✓ Testing compression and decompression script on multiple d
- ✓ Testing demultiplexing script on multiple datasets

Read Quality Control Script Creation

- ✓ Creation of parsing script for FastQC metrics for multiple si
- ✓ Parametric randomization script
- ✓ Parsing QC metrics of all iterations
- ✓ Bootstrapping script

Script Test-Runs

- ✓ Parsing and aggregation of FastQC metrics from multi-site data (N=10)
- ✓ Randomization of trimming parameters from different trimmers simultaneously (6 trimmers total)
- ✓ Bootstrapping of randomization of trimming parameters. 6 Trimmers, 1000 bootstraps

Pipeline Preparation for HPC

- ✓ Calculate using quotation from PGC
- ✓ Set up config files
- ✓ Taxo-metagenomics pipeline

Script Descriptions

Pipelines

This section covers all the scripts that were created during the Project.

Listen, I'm eternally curious and I don't just want to settle for any random journal. No offense, but I'm aiming for something high-impact!

These were created during off hours or during work hours, so some scripts might seem irrelevant at first, ***but trust me, there's a conscientiousness or meticulousness to this madness.***

I have separated them into folders/repositories (shameless plug here: https://github.com/GABallena) based on their relevance to the project.

- **Project4** - Essential scripts directly related to the core analyses of the project.
- **Side** - Scripts that can potentially be used to increase the robustness of the paper.
- **Main** - Scripts related to file organization and data management, crucial for handling large datasets.

Project 4 Scripts

1 ARG-MGE.smk

Stage: Draft General Purpose

This pipeline is designed for **comprehensive metagenomic analysis of ARGs**, it also includes placeholders for analyses of **mobile genetic elements (MGEs)**, and **plasmid detection**. It integrates tools for **read quality control, assembly, annotation, taxonomic profiling, and structural variant (SV) detection** to provide a high-resolution view of the genetic components in metagenomic samples.

Specifics:

1.1 Preprocessing

- **It first uses** `PEAR` **to merge paired-end reads.**
  Technical Notes: `PEAR (Paired-End reAd mergeR)` compares paired-reads to correct infer the likely bases on its associated pair
  Rationale: The main goal here is to ensure good read quality for downstream analyses and to maximize the amount of data by reducing gaps in the sequence and improving confidence of base calls within that region.
  Note: Another tool, `PandaSeq` which does the same thing is used later, this usage of `PEAR` here is because it is more optimized for larger datasets - which in this case are trimmed reads.

- **Followed by conversion from** `FASTQ` **to** `FASTA` **using** `seqtk`**.**
  Technical Notes: `seqtk` converts compressed `FASTQ` files to `FASTA`.
  Rationale: Some tools cannot work on `FASTQ` formats.

- **Translation of the** `CARD-protein homologues database` `transeq`**, and then reverse-translates these proteins to nucleotides for alignment.**
  Technical Notes: `transeq` from `EMBOSS` translates nucleotide sequences to protein sequences based on **standard genetic code**. `backtranseq` reverses this to allow iterative alignments with nucleotide sequences.
  Rationale: This leverages conserved protein-level information, which is lost at the nucleotide level due to synonymous mutations - while also increasing the sensitivity to potential ARG proteins from k-mer alignment. See Nature Methods, doi: doi.org/10.1038/s41592-019-0437-4 (2019) for more details.

1.2 Metagenomic assembly of ARG-contigs

- **Iterative alignment is performed using** `KMA` **to align reads against reverse-translated CARD sequences for high-sensitivity identification.**
  Technical Notes: `KMA` (K-mer Alignment) is used find the reverse-translated ARGs with the proximity filtering option to determine the surrounding regions of ARGs. This is done iteratively for each gene increasing the ARG-associated database.
  Rationale: Firstly, `KMA` is used because, unlike `Bowtie2` and `BWA-MEM`, which were created specifically for Human metagenomics, KMA does not suffer (or suffers less) from multi-allelic databases. Secondly, iterative alignment using this process allows us to contextualize the region

wherein ARGs reside - thereby narrowing our focus onto these local regions instead of looking at the global genomic context. **Notes:** The script is designed to have a cap on the number of iterations KMA creates, increasing the database size.

- **Merging ARG-related reads database with** `PANDASeq`**.**
  Technical Notes: `PANDASeq` is then used to further refine the paired-reads collated in the ARG-related genes database.
  Rationale: `PANDASeq` was chosen as, while being slower than `PEAR`, it is more accurate. This mergins step is included to ensure that only high-confidence reads are assembled. **Note:** We leverage the fact that the ARG-related genes database is smaller compared to the raw metagenomic reads database.

- **Merged reads are assembled into contigs using** `metaSPAdes`**.**
  Technical Notes: `metaSPAdes` is then used to create contiguous sequences from these local regions by extending them using reads from the whole metagenomic pool. Additionally, Contigs are filtered by length here to remove possible artefacts.
  Rationale: `metaSPAdes` was chosen as, while being slower than `MEGAHIT`, it is optimized in handling highly diverse and mixed microbial populations. CARD is used here because it is manually curated and updated regularly - in order to be included in the database, there must be clinical data (e.g. ASTs(Antimicrobial susceptibility testing)) involved in the study.
  Notably, this would decrease the sensitivity of our ARGs - and would mostly be biased towards those reported in the clinical setting. To counter this, we could also incorporate other tools such as `ResFinder`, the `NCBI AMR Database`, and `ARG-ANNOT`
  Notes: Filtering of contig length is handled by a python script that determines the minimum ARG sequence dictated by `CARD`.
  Notes: Contigs are further extended using contigextender to form scaffolds
  Notes: Might add other contig extendending programs like `GapFiller` - which leverages mate-pair information

- `Python` **script is created to do another round of checking contig quality for downstream analysis,** `R` **scripts are used to visualize the data.**
  Technical Notes: The `Python` script will measure standard contig quality metrics e.g. N50, L50 etc.
  Rationale: Should be obvious why

- **Confirmation of contigs with ARGs using** `RGI`**.**
  Technical Notes: `RGI` scan the contigs and check whether which contigs created by `metaSPAdes` have ARGs in them.
  Rationale: `metaSPAdes` may have created contigs that DO NOT contain ARGs, and have instead assembled them into a more matching contig (a false-positive misassembly) - this can happen because of the different databases being used; also parallelization of methods like this increases robustness because it has been confirmed independently from different starting points (bottom-up vs top-down approach). This allows us to filter ARG-containing contigs.
  Notes: `RGI` is the official scanner of `CARD`.

1.3 Read-mapping

- **C**hecking coverage statistics on contigs using raw reads
  Technical Notes: `Samtools` is used here to map the raw reads from the larger database back to the assembled contigs and then calculates the coverage over the entire contig.
  Rationale: Read-mapping is a quality control protocol used in metagenomics, to determine the quality of the assembly. High-coverage means that many of the k-mers align well with that region of the contig, while low coverage is evidence of inconsistent mapping and that the contigs should be refined, split, or discarded.
  Note If there are persistent (after further refinement and reassemblies) sudden differences in coverage across a contig, that contig could be chimeric, meaning, it could be from two different populations.
  Extra note A `Python` script is included in the pipeline that is determine visualize and check how the coverage changes over contig regions. In general, they could be interpreted as the following:

  Smooth, Uniform Coverage: Typically shown by well-assembled contigs.
  Sharp Coverage Drop: May need to be split or flagged for reassembly. May also be a misassembly point (chimeric contitg) or caused by a structural variant
  Coverage Gaps: Regions with little to no read support; a strong indicator of misassembly.
  Gradual Drops: Overlapping reads, repetitive or duplicate regions, partial HGT, sequence heterogeneity, or coverage differences due to a mixed population. Repeats and duplications can be filtered out using tools like `RepeatMasker` or `BLAST`
  (to be added to the script)
  Sharp increase: May be due to repetitive or duplicated regions, the assembler decided to collapse all the reads into that one contig, amplification bias from PCR, HGT, SV, chimeras.

Read mapping more details Read mapping tools
Technical Notes: Four (4) Tools will be used in parallel to do the read mapping process `BWA` `Bowtie` `KMA`, and `minimap2`, their parameters have been adjusted to map reads at 95 % identity to the contigs.

Mapper	K-mer Length	Mismatch Penalty	Gap Opening Penalty	Gap Extension Penalty
`BWA`	21	5	7	2
`BWA`	31	4	6	2
`BWA`	51	3	5	1
`Bowtie2`	-	4,2	5,2	5,2
`KMA`	Default	95% identity	Automatic	Automatic
`Minimap2`	-	5	7,2	4,1

Table 1: Example table of mapper configurations without command example

Rationale: 95 % identity is used to increase sensitivity - as is standard for determining homologous sequences. This adjustment was made because k-mers are either attach or don't. Parallelization is used to increase robusness.
Note K-mer extension is used to increase the accuracy of mapping. `BWA` k-mer lengths can be

adjusted, while KMA does it by default. The others, cannot be adjusted.

I chose to change the script to not allow gaps during this phase as we already used reverse translation earlier to correct for synonymous codons, and protein sequences are more important when it comes to ARG function, the new values are below. I also added protein-based read mapping.

Tool	K-mer Length	Mismatch Penalty	Gap Opening Penalty	Gap Extension Penalty
BWA	21, 31, 51	5, 4, 3	1000	1000
Bowtie2	-	4,2	1000,1000	1000,1000
KMA	-	95% identity	Automatic	Automatic
Minimap2	-	5	1000,1000	1000,1000

Table 2: Alignment parameters for ungapped alignments across BWA, Bowtie2, KMA, and Minimap2

Tool	Input Files	Key Parameters
tblastn	– Protein sequences: Translated protein sequences contigs – Nucleotide database: Cleaned sequence database	– --outfmt 6 – --evalue 1e-5 – --gapopen 5 – --gapextend 2 – --matrix BLOSUM62
blastp	– Protein sequences: Translated protein sequences contigs – Protein database: Translated cleaned sequence database	– --outfmt 6 – --evalue 1e-5 – --gapopen 5 – --gapextend 2 – --matrix BLOSUM62

Table 3: Protein read-mapping parameters for tblastn and blastp

Rationale: The main rationale for adding a protein-based read mapping protocol is because ARGs are primarily about their ***protein-protein interactions*** (biological relevance). This method also accounts for frameshifts with higher specificity to homologous regions.

On a personal note: This approach may also require further exploration into whether protein-protein interactions are altered—perhaps by investigating changes in binding sites. Which is a story for another day (***why do I do this to myself?***)

Taxonomic profiling of reads and contigs

Technical Notes: `Kraken2` uses k-mer-based classification to assign taxonomy based on raw-reads. While `SprayNPray` complements this by assigning taxonomy at the contig level.
Rationale: By comparing their respective databases with our ARG-related databases, we will be able to connect our reads and/or contigs to their corresponding taxa, uncovering the microbial hosts responsible for carrying and potentially spreading ARGs in the environment.

Detect structural variants (SVs) in contigs

Technical Notes: `Manta` identifies large genomic rearrangements such as insertions, deletions, and duplications.
Rationale: Chimeric contigs may be due to systematic error or real biological signals. These chimeric contigs can be detected by `Kraken2` and `SprayNPray` (i.e., when a portion of a contig is being assigned to different taxa). The rationale behind this step is to investigate whether structural variations are present — which may be evidence of horizontal gene transfer (HGT) events.

1.4 Mobile Genetic Elements (TBA)

Technical Notes: Tools such as the following can be used:

- `HMMER3` suite
- `Tnppred` - a transposon predictor tool

Rationale:

Sketching contigs followed by calculating `Bray-Curtis` diversity

Technical Notes: `Mash Sketch` uses a MinHash approach to generate a presence/absence profile of ARGs across contigs. This gives us a quick snapshot of what the contigs "look like" in terms of ARG content. For `Bray-Curtis` diversity, we calculate a dissimilarity matrix from the abundance data of ARGs, followed by a PCoA plot to visualize similarities between contigs based on their ARG profiles.
Rationale: The `Mash Sketch` helps rapidly identify the genetic makeup of contigs in terms of ARGs, which provides a foundation for further investigation. By applying `Bray-Curtis` diversity and using PCoA, we can group contigs based on their ARG similarity. If contigs with the same sketch group together, we can trace them back to their taxonomic IDs to identify the microbial hosts. However, if contigs have similar ARG profiles but belong to different taxa, this could serve as **evidence for Horizontal Gene Transfer (HGT)**. This dual approach allows us to trace ARG

spread and potential HGT events in a metagenomic context.
Note: `Bray-Curtis` (dis-)similarity is most often used as a presence or absence diversity metric.

1.5 Plasmids

Determination of putative plasmids

Technical Notes: The tools listed below will be used in parallel. Plasmids are considered valid when all 4 tools predict plasmid signatures in the contig.

- `PlasPredict` pipeline
- `Recycler`
- `PlasmidFinder`
- `MOBSuite` plasmid marker annotator

If plasmid signatures are present, `plasmidSPAdes` along with `GapFiller` will be used to check if the contig can circularize. `oriTfinder` will then be applied to contigs with fewer than 4 fragments. A `Python` script will calculate GC skews of the chimeric contig and compare it to its taxonomic counterparts. Another `Python` script will normalize the data according to `16S rRNA` from trimmed reads. Lastly, plasmid percentage will be calculated based on reads mapped to putative plasmids over the total reads.

Equation:

$$\text{Plasmid Percentage} = \left(\frac{\text{Plasmid Reads}}{\text{Total Reads}}\right) \times 100$$

Rationale: `oriTfinder` looks for `origin of transfer` sites (`oriT`), which are characteristics of conjugative plasmid . This whole sub-pipeline is to look for evidence of conjugative plasmid transfer as the cause of these chimeric contigs. Normalization and percentage counts are used here to further check whether these ”plasmids” align with our understanding of the average plasmid copy number.
Note: Will also be drafting a script to do sliding window analysis of `GC-skews` - as different characteristics of this curve can be interpreted in different ways.

1.6 Phages

Phage influence signatures

Technical Notes: They will be determined using a variety of tools:

- `VirSorter`: Identifies viral signatures within microbial genomes and separates prophages from bacterial sequences.
- `PHASTER`: A web-based tool for phage search and annotation, identifying integrated prophages.
- `VIBRANT`: A tool that combines several approaches to identify and annotate phage elements in metagenomic sequences.

Rationale:This analysis aims to detect potential phage signatures in the chimeric contigs. Since phages are mobile genetic elements, their involvement in transferring ARGs through transduction is highly relevant.Phages, especially temperate phages, can integrate into bacterial genomes and excise themselves, sometimes carrying host genetic material, such as ARGs, with them.The integration and excision signatures detected in contigs will provide evidence of possible transduction events in our datasets, supporting the hypothesis of ARG dissemination via phages.

Phage Signature Extraction and Phylogenetic Analysis

Technical Notes: Phage-associated genes will be extracted from the chimeric contigs, followed by phylogenetic analysis to uncover evolutionary relationships.`FastTree` will be used to build a phylogenetic tree based on the extracted phage genes.For visualization, tools like `iTOL` or `FigTree` can be used to generate an interpretable phylogenetic tree.

Rationale:Phage genes embedded in chimeric contigs (their taxonomy) may serve as strong evidence of horizontal gene transfer (HGT) events. The aim here is to check the evolutionary origins of the phage genes found in our dataset and their potential involvement in the dissemination of ARGs.

Future Considerations

: Will continue improving this section (everything regarding HGT) by evaluating the results from these tools, aligning them with ARG presence, and refining the approach for identifying conjugation, transposon, and transduction events within chimeric contigs. This may also involve validating phage activity—*again, why do I do this to myself?*

2 metagenomics_general.smk

Stage: Done

General Purpose taxo-metagenomics

This pipeline is designed for **The essentials in metagenomics**, which includes **quality checking, filtering, and trimming of raw reads to clean reads**. As well as the usual **taxo-metagenomic analysis**.

Specifics:

2.1 Quality control (Pre-processing)

Raw trimming of raw metagenomic data

Technical Notes: `FastQC` is used on **raw reads**

Rationale: This is mainly used as a point of comparison - determine whether the next step (trimming) was effective. This pre-processing step is the starting point in any and all metagenomics pipelines.

Notes: This whole quality control steps are interconnected with each other.

Trimming

Technical Notes: `Trimmomatic` is used on **raw reads**

Rationale: Trimming involves removing low quality bases (often depending on something called the Phred Score - which is just a measure of how ”confident” we are that the base on that site was accurate), adapters, and filtering reads that go below a specific length threshold.

Notes: Journals often report the parameters on `Trimmomatic` (or any trimmer they decide to use); this is often so that the study is reproducible, should one decide to actually reproduce the study starting from scratch (raw reads).

Notes: There are many different trimmers each with their own strengths and weaknesses, `Trimmomatic` is just the most popular trimmer and is thus used here, though studies differ in the parameters they used for trimming - which often dictates how strict they are with what they define as ”good enough”.

Perspective: Why different people choose different trimmers depend on the **strengths and weaknesses** of the trimmer e.g. trimmers like ”fastp” is used because it’s fast making it suitable for very large datasets like deep sequencing. While some trimmers like `Sickle` has automatic adjustment over the entire sequence - which makes it useful for very ancient datasets where DNA is often highly-degraded. Other times, it’s for **convenience** like `Trim-Galore` which combines `FastQC` and `Cutadapt` trimmer in a single command. Another good example is `BBDuk` which is part of a larger package called `BBtools`, `BBDuk` also has an built-in contamination detection - so it’s particularly good at filtering out usual contaminants like sequences known to be from the human genome. so you can simply just use all the modules in that package for all-in-one processing. Other times, it’s just **familiarity**.

Quality checks of post-trimmed data

Technical Notes: `FastQC` is used on **trimmed reads** to determine how effective the trimming process

was.
Rationale: If the trimming process was effective, we should notice a better quality reads here, otherwise, we might have to adjust the trimming parameters.
Notes: Determination of whether the trimmed reads are "clean enough" is more of an art rather than actual science, though thresholds exist like Phred > 20 or Phred > 30 depending on how strict you are as a researcher.

2.2 Metagenomic Taxonomy

Taxonomy from Cleaned Reads
Technical Notes: `Kraken2` is used on **raw reads** to determine which species they came from.
Rationale: Taxonomy based on reads is standard on metagenomics instead of using assembled contigs because information is lost during the assembly process. By using cleaned reads we make sure that:

1. We are maximizing the amount of data
2. We are not being biased by low quality reads

Notes: There are many ways in which taxonomy is assigned to raw reads the `Kraken-based` packages use a curated database that links k-mers from your database to k-mers generated from their database (usually manually curated).
Notes: Others like `MetaPhlan` first create "markers" that are based off of known sequences, and then scan your raw reads for these markers, which it then checks for under what taxon/taxa that marker fell under.

2.3 Diversity analysis

Diversity analysis per site or sample
Technical Notes: Here `Bracken` - an extension of the `Kraken` packages or `Qiime2` are often used to calculate diversity. Instead I opted to create my own `Python` script because:

1. I find that the diversity indices in either tools are limited to only the most often used i.e. the most popular indices - so it is not comprehensive
2. I've had problems integrating them into the pipeline because some dependency limitations, un-updated scripts, and a pre-processing step that requires converting all of `Kraken2, Bracken` files, then importing them to `Qimme2` which is too time-consuming and inefficient - my script just automatically calulates from `Bracken` outputs and just puts out all the possible (non-phylogenetic-based-which you would need a phylogenetic tree to build first) indices out there.

Rationale: Taxonomy based on reads is standard on metagenomics instead of using assembled contigs because information is lost during the assembly process. By using cleaned reads we make sure that:

1. We are maximizing the amount of data

2. We are not being biased by low quality reads

Notes: There are many ways in which taxonomy is assigned to raw reads the `Kraken-based` packages use a curated database that links k-mers from your database to k-mers generated from their database (usually manually curated).
Notes: Others like `MetaPhlan` first create ”markers” that are based off of known sequences, and then scan your raw reads for these markers, which it then checks for under what taxon/taxa that marker fell under.

3 trim_randomizer.smk

Stage: Done

Purpose: Randomization of trimming parameters

This is a module that will be part of a bigger pipeline. The idea here to is randomize parameters in a variety of trimmers in this case `Trimmomatic`, `fastp`, `CutAdapt`, `BBDuk`, and `Sickle` - famous bioinformatics trimming tools. `Trimmomatic` and `Sickle` in particular are widely used in `Illumina`-based data.

3.1 Random Parameter Generation

Technical Notes: Random parameters are generated for each trimming tool (Trimmomatic, Fastp, Cutadapt, BBDuk, Sickle). This is done using the `random module`, which creates random values for parameters such as quality scores, read length, adapter sequences, and error rates. These parameters are stored in the `generated_parameters` dictionary to ensure consistency across iterations for each sample.
Rationale:By randomizing parameters, the workflow allows for testing different parameter sets across multiple iterations to find optimal settings for trimming and quality control.
Notes: This approach helps with parameter exploration, particularly when you are unsure which trimming settings will give the best results. The randomness provides variability, which can highlight which parameters consistently lead to good results.

3.2 Log Parameters to a `TSV` File

Technical Notes: Each set of generated parameters is logged into a separate TSV file (e.g., `trimmomatic_params.tsv`, `fastp_params.tsv`). The file headers are written only once, and parameters are appended as the trimming steps proceed. This is done in a structured way so that you can track the exact parameters used for each sample and iteration.
Rationale:Logging ensures reproducibility and transparency in bioinformatics workflows. Having a record of all the parameter values used in each iteration is crucial for comparing results and for future reference
Notes: This practice is a standard in scientific workflows where random parameter generation is involved. It helps maintain a clear audit trail of the steps performed and aids in troubleshooting or refining workflows later.

3.3 Define `Rules` for Each Tool

Technical Notes: The script uses `Snakemake rules` to define separate rules for each tool which include

1. Input folder (as the script is designed to go through all the `FASTQ` samples within the folder)
2. Output folder, where the processed files will be saved (e.g., trimmed paired and unpaired reads). The script is designed to keep all the `trimmed reads` in separate files.
3. Params: Fetches the parameters to be randomized.

Rationale:Using Snakemake here allows for parallel execution of the workflow. This parallelization if very important as the generation of random paramaters is created using a random `seed`. Using a random `seed` like this allows us to replicate what the parameters that had the optimal results were by tracking down what seed was assigned as dictated in the `TSV` file.
Note keep in mind that this will create a large number of folders if you decide to iterate many times - as each iteration, per tool, will have its own folder full of trimmed reads, per sample/site.

3.4 Interpretation and Analysis of Results (TBA)

Technical Notes: Once all iterations have completed, the trimmed files can be analyzed to determine which parameters led to the best results in terms of quality and length distribution of reads. There are many different metrics that can be used to interpret the results, including (but not limited to)

1. Quality Scores - significant differences in quality among sites and across entire sequences (Phred score, Contamination, Adapter Removal, N Content, Length Distribution, etc.). Can be done using tools like `FastQC`

2. Visualization can be done using `Rstudio` or `Python's matplotlib` to visually look for differences in abnormalities.

Rationale:Evaluating read quality and assessing key metrics post-trimming helps to ensure that the data is suitable for downstream analyses. Optimal trimming should maximize the number of high-quality, usable reads while eliminating low-quality bases and adapter contamination.

A Personal Note

I have long pondered upon the idea of creating a graph with `biological information` on the y-axis and `read length` on the x-axis. I hypothesize that we will find a **"sweet spot"** wherein we can optimize the amount of `information/read` using such trimmers. Unfortunately, it is extremely difficult to define what `a biological sequence` really is, because technically you can generate any random sequence - whether protein or nucleotide. That's the main reason I have been stuck on this problem for quite a while, I've been trying to find the answer first.
The core difficulty here lies in defining what constitutes biological information in a meaningful, quantifiable way. Since any random sequence of nucleotides or proteins could be technically "valid" (false-positives).
When people are asked this question they often give **DESCRIPTIONS** of what a `biological sequence` is but not what **DEFINES** a `biological sequence`. I understand that there are many characteristics that can help in determining the signal from the noise, but I am just not satisfied with whether this "thing" checks most (if not all) the boxes - I need a non-subjective answer to this problem.

4 bootstrapping_rawreads.smk

Stage: Further refinement

Purpose: Bootstrapping the taxo_metagenomic pipeline itself

Bootstrapping is the process of randomly selecting from a pool of samples (with replacement) and using that in a specific process you want to bootstrap. This is a standard method used in molecular phylogenetics to determine the robustness of trees where a > 70 support from bootstrapped data is considered robust enough.

The principle of bootstrapping in phylogenetics

Phylogenetics uses this statistical technique because (in principle) it effectively means that removing parts of the entire sequence does not alter the topology of the tree.
Here I'm adapting this method with `Kraken2`'s taxonomic profiling to see whether the taxonomic support of its k-mer assignment is also consistent even with changes in the sampling sites.
This is still WIP because I plan to go step further and start bootstrapping the raw reads themselves to see if changes in reads changes the topology of taxonomic assignment.

4.1 Directory setup of temporary files

Technical Notes:Snakemake starts by ensuring the existence of necessary directories. Most notably, the temporary bootstrap directories.
Rationale: Bootstrapping is sometimes done iteratively thousands of times, so making this a temporary directory helps manage space.
Notes:

4.2 Sample Identification

Technical Notes:The workflow identifies sample names by parsing filenames in the raw reads directory.
Rationale: Inclusion of these in the script allows the user to flexibly configure the naming convention and the directory in which they want to bootstrap.
Notes: Presently it looks for _R1.fastq.gz and it's associated pair, _R2.fastq.gz in the raw_reads directory.

4.3 Bootstrapping

Technical Notes:This executes `bootstrap_reads.py` in the scripts directory to start bootstrapping the paired-end reads and outputs them in the temporary folders I mentioned earlier.
Rationale: Moving bootstrapping logic into a Python script leverages its ability to create randomizations from its libraries. Also it allows us to define a `seed` so it is reproducible (if you want to reproduce) the results anyway.
Note: The number of bootstraps and the fraction of samples you want to retain can be controlled in the config.yaml file in the configuration folder.
On a personal note: Before writing this part of the document (Sep 19, 2024, 3:00 AM), I decided to change the bootstrapping rule. It used to rely on a simplified approximation. The probability of not being selected after N independent draws from a sample of size N is given by:

$$P = \left(1 - \frac{1}{N}\right)^N$$

This is a well-known mathematical equation describing the probability of not selecting a sample at least once in N draws. As $N \to \infty$, this probability approaches:

$$\frac{1}{e}$$

Previously, I used the probability of a sample *being selected*, which is:

$$1 - \frac{1}{e}$$

This was used as an approximation for bootstrapping by shuffling and adjusting the sample size. However, the updated script now performs **actual bootstrapping**, sampling with replacement, which is a more accurate statistical method for resampling.

4.4 Run `kraken_pipeline.bash`

Technical Notes: This Shellscript (or bash file) is used to automate the processing of all files from the bootstrapping. It runs them under `Kraken2` then `Bracken` to generate taxonomy profiles for all of them. It also creates a log file for each replicate to provide traceability and error checking, helping diagnose any issues with specific replicates or samples.

Rationale: Read why I'm bootstrapping from the textbox above. The reason why I also included `Bracken` and measurements diversity metrics in the analysis per sampling replicate is so we can analyze how the topology of diversity also changes - similiar to how we look at topology of phylogenetic trees. The final process consolidates all the diversity metrics (alpha diversity) and matrices (beta diversity) into a TSV file.

Note: The decision to use Snakemake and Shellscripts here is so that the bootstrapping comes first before the pipeline is introduced. Otherwise Snakemake will run the entire thing in parallel, taking up so much memory because it runs `Kraken2` for every single sample instead of doing it in batches - which takes up so much unnecessary time.

Daily entries

1 September 18, 2024

To-do List

- ☐ Call for a group meeting regarding logistics (or when they will be available via Zoom call)
  - ☐ Discuss the issue about VMMC review fee
  - ☐ Raise concerns about data storage and management
  - ☐ Inquire if there are scripts I can develop to streamline the bureaucratic process
- ☐ Contact engineering/sanitation departments via landline
  - Talk about possible sampling sites and that we will present out authorization upon arrival
  - ☐ VMMC
    - Tell them that we are have a go-signal from the admin, but we have yet to pay for the fee - which can be held off for later
  - ☐ Mary Johnston
    -
  - ☐ ManilaMed
    - Tell them we already have a go signal from Ma'am Eula
  - ☐ St. Lukes
    - Inquire where Engr. Valenzuela is there
    - Inquire the status of their renovation and if they are available for sampling this October already as talked about last year

Completed Tasks

- ✓ raw API results now readable stored in TSV file
- ✓ Created symlinks in private (repo) to .git folders in public repos to enable tracking of Git and Github activities
- ✓ Resolved residual time-tracking activities within the local computer

Scripts Worked On

- ✓ checkingAPI.sh (now updated to make the raw API results readable)

Issues Encountered

-

Need to Troubleshoot

-

2 September 17, 2024

To-do List

- 🡒 Call for a group meeting regarding logistics
    - ✓ Offer to use landline to minimize on-site visits and reduce transportation costs; as point-persons have not been replying via email as quickly lately
    - ✓ Discuss PGC calculations and talk about allocatable budget
    - ✓ On-site orientation with PGC engineering department (OETS) regarding sampling sites **taken care of by James and Roch**
    - 🡒 Raise concerns about data storage and management
    - 🡒 Inquire if there are scripts I can develop to streamline the bureaucratic process

Completed Tasks

- Setup wakatime in VS Code and Ubuntu to track my coding productivity
- Helped in receiving and moving the autoclave and fridge needed by the Program

Scripts Worked On

- ~~rawAPIformatter.sh~~

✓ checkingAPI.sh

- ~~wakatime.sh~~

Issues Encountered

✓ Connection blockage from wakatime API

🡒 raw API results is unreadable

Need to Troubleshoot

- conky display of wakatime API for desktop

3 September 16, 2024

To-do List

- Call for a group meeting regarding logistics
  - Offer to use landline to minimize on-site visits and reduce transportation costs; as point-persons have not been replying via email as quickly lately
  - Discuss PGC calculations and talk about allocatable budget
  - Raise concerns about data storage and management
  - Inquire if there are scripts I can develop to streamline the bureaucratic process

Completed Tasks

✓ Updated GitHub repositories; created new repositories: `Documentation` and `Confidential`

✓ Created a new bash script to track progress across all repositories

✓ Set up this research diary to document daily progress and tasks

✓ Python script that calculates contig quality metrics L50, N50, but also L90, N90, GC skew etc.

✓ Python script that detects and calculates changes in coverage over contigs for read mapping

✓ Typesetting: explaining the process and rationale inside the ARG-MGE.smk pipeline

✓ Integration of the two scripts to the ARG-MGE pipeline

Scripts Worked On

- `Project4/ARG_MGE.smk` - updated

✓ `Project4/calculate_contig_quality.py`

✓ `Project4/plot_and_detect_intermediate_coverage.py`

Issues Encountered

✓ Could not access GitHub, needed to reset the SSH keys and update my Git histories for version control

Need to Troubleshoot

- None identified at this time