THE DEVELOPMENT AND APPLICATION OF NEW COMPUTATIONAL TOOLS FOR WORKING WITH VIRAL METAGENOMIC DATA

by

Ezra Kitson

B.Sc., Imperial College London, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Microbiology and Immunology)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, a thesis/dissertation entitled:

The Development and Application of New Computational Tools for Working with Viral Metagenomic Data

submitted by Ezra Kitson in partial fulfillment of the requirements for

the degree of Master of Science

in Microbiology and Immunology

Examining Committee:

Curtis Suttle, Microbiology and Immunology
Supervisor

Steven Hallam, Microbiology and Immunology
Supervisory Committee Member

Supervisory Committee Member

William Hsiao, BC Centre for Disease Control
Additional Examiner

Abstract

Next generation, high throughput sequencing has revolutionised the way in which we are able to view the microbial world. We have now generated a large volume of metagenomic sequence data describing viruses and bacteria in diverse environments across the planet. These data require computational processing in order to be used in further analysis. Manipulating the data in the way a bioinformatician wants is often a major difficulty in a metagenomic research. There are two reasons for this. One is that, as the field is nascent, there are many useful data processing tasks that do not yet have published computational tools. A second is that the computational tools that have been published to date are often poorly documented and complicated making them difficult to use in a routine application.

Research in this thesis focusses on developing simple computational tools for managing viral metagenomic data. Viral metagenomic data presents the bioinformatician with specific difficulties owing to its size, poor quality and largely novel sequence content. Four new computational tools for managing viral metagenomic data are presented and benchmarked here. Three of these tools expedite everyday researching tasks, automating a process that would otherwise be done manually. The fourth, VHost-Classifier, allows a new scientific question to be asked using viral metagenomic data.

In the final chapter VHost-Classifier is applied to analyse viruses published in the NCBI taxonomy database by host organism. The results reveal a large anthropocentric bias in viral sequencing.

Lay Summary

Viral metagenomics datasets describe the genetic composition of virus communities in the environment. The raw metagenomic data requires extensive computational processing to be used to make meaningful analysis. Four new software tools are presented that enable processing of the data; three automate routine tasks that would otherwise be done manually, and the fourth, VHost-Classifier enables new questions to be asked of viral metagenomic data. In the final chapter VHost-Classifier is used to analyse the sequencing bias to date in viral taxa published to NCBI taxonomy.

Preface

The thesis was written by me with the guidance and encouragement of my colleagues and supervisor, and with editorial input from my supervisor and Dr. Chris Deeg, Dr. Gideon Mordecai, Dr. Jan Finke, Dr. Steven Hallam and Dr. William Hsiao. All four pieces of software showcased in chapter two are designed and written by me and published on Github with an MIT License. The analysis performed in chapter three was conceived in discussions with my supervisor and performed by me; the raw data was accessed on the NCBI taxonomy database, which is freely available online to download.

Chapter 2.1 of this thesis has been published in Oxford Bioinformatics as an Application Note. Kitson,E. and Suttle,C.A. (2019) VHost-Classifier: Virus-Host Classification using natural language processing. Bioinformatics, btz151 (https://doi.org/10.1093/bioinformatics/btz151).

Table of Contents

List of Tables

List of Figures

Glossary

Assembly: The process by which short reads of nucleic acid are aligned based upon their overlapping regions and combined into longer sequences called contigs.

BLAST (Basic Local Alignment Search Tool): Bioinformatics algorithm that assigns a biological identity to nucleic acid sequences based on homology to sequences in a database.

Bootstrap value: A confidence score assigned to each node in a phylogenetic tree representing the statistical uncertainty about the node topology. The tree data is randomly subsampled and the phylogenetic analysis is rerun, the bootstrap value is the proportion of replicates in which a node was present.

Contig: A longer, contiguous sequence of nucleic acids produced from the assembly of short reads.

DIAMOND (Double Index AlignMent Of Next-generation sequencing Data: Bioinformatics algorithm that assigns a biological identity to nucleic acid sequences based on homology to sequences in a database.

Metadata: Data that describes data. The metadata about my thesis would include the title, author name, page length, formatting, subject, abstract and keywords.

NCBI taxonomy database (National Center for Biotechnology Information): A database curating taxonomic information on all biological entities discovered on the planet.

Open reading frame (ORF): Stretch of nucleic acid that codes for a gene between a start and stop codon.

Read: A nucleotide sequence generated during a metagenomic sequencing run.

TaxonID: A unique number that is given to each genome uploaded onto NCBI taxonomy. The number is the index position of the organism name in the NCBI database.

VtaxonID: A taxonID belonging to a virus.

Acknowledgements

Work on this thesis was disrupted at the beginning of September 2018 by a bicycle accident in which I sustained a concussion. The aftermath of the accident was of the most disorienting and difficult periods of my life and I am immensely grateful for the support of my family, colleagues, committee and supervisor; alone, I would not have been able to complete this work.

From an academic perspective, I would like to thank my supervisor, Prof. Curtis Suttle, for enabling me to follow my passions in computational science, providing advice and feedback on manuscripts, and supporting me through difficult times. I would also like to thank my thesis committee, Prof. Philippe Tortell and Prof. Steven Hallam, who gave me encouragement and helped shape the direction of this thesis.

I would like to thank members of the Suttle Lab, past and present who helped in many ways. I would especially like to thank Dr. Marli Vlok, who introduced me to the field of viral metagenomics and provided invaluable assistance in analysing the Watershed Discovery Project data - she inspired me to create the tools presented here - and Dr. Chris Deeg who mentored me throughout my first year and gave feedback on several manuscripts, including Chapters 2 and 3 of this thesis. Special thanks also to Dr. Jan Finke and Dr. Gideon Mordecai, who gave feedback on Chapters 1 and 3, respectively, of this thesis.

Outside of the academic sphere, I would like to thank Grant and Jenny who discovered me after my accident, took me to safety and called me an ambulance. Their good will and quick thinking prevented much harm. I would also like to thank the staff at Vancouver General Hospital who looked after me during my stay in hospital and my friends from Green College who came to keep me company in the most trying hours.

Finally, I wish to thank my family who I have worried sick and who have never failed me in their love and support.

Dedication

To my parents, Jim and Bev.

Chapter 1: Introduction

Viruses are the most abundant biological entity on the planet and a significant driver of mortality and gene transfer in any ecosystem (Suttle, 2007) Viruses in the natural environment regulate fundamental biogeochemical processes and likely constitute the biggest reservoir of genetic diversity on the planet (Suttle, 2005). Research on environmental viruses is therefore very important; however, it is handicapped by our inability to culture representative isolates.

With the advent of metagenomics in the early 2000s, which used high-throughput sequencing to study entire communities of microorganisms without the need for laboratory cultivation, viral metagenomic data quickly became a fundamental resource for exploring the diversity and impact of viruses in nature. Now there are wealth of metagenomic data from a range of interesting environments, but a lack of accessible, widely used and standardized computational tools to analyse these data. This thesis focused on the development and application of new tools for working with viral metagenomic data.

In this introduction, I first explain the process of viral metagenomics, and briefly compare it to cultivation-based approaches of virus characterisation. I then discuss the difficulties in doing bioinformatic analysis on viral metagenomes. Some of the issues covered are general to any bioinformatic analysis while others are specific to viral metagenomics. Finally, I introduce the software tools I have developed that help address some of these issues.

1.1. Viral Metagenomics

Metagenomics is a cultivation-independent technique which uses high-throughput sequencing and bioinformatic analysis to snapshot the genetic content of an entire community of microorganisms. It has been used to explore microbial diversity in a plethora of contexts, from the human microbiome to the upper atmosphere (Turnbaugh *et al.*, 2007; Whon *et al.*, 2012).

1.1.1. Protocol for Viral Metagenomics

Viral metagenomics focuses on building an unbiased representation of the viral nucleic acid present in an environmental sample. In conventional protocols this is achieved in three stages (Anon, 2018). First, an environmental sample is obtained and filtered to separate the virion components from the cellular ones, to reduce contamination of the genetic signal from the

cellular components. Next high-throughput sequencing techniques are used to generate millions of short segments of nucleic acid, known as reads, which are fragments of all the virus genomes present in the sample. For further analysis of the data, the short reads have to be processed in a bioinformatic pipeline.

Bioinformatic analysis normally proceeds in three stages. Firstly, there is a cleaning process, in which the reads are trimmed to remove adaptor and index sequences (small oligonucleotides ligated to the ends of reads during the sequencing process) and low-quality reads are removed. Secondly, short reads are assembled into longer stretches of nucleic acid known as contiguous sequences (contigs). This is achieved by algorithms that compare overlapping regions at the ends of reads. Next, contigs or reads are compared to a genetic database and assigned a biological identity according to their similarity to sequences stored in this database. Conventionally this is done using tBLASTx (basic local alignment search tool), which translates the query sequences into all possible six reading frames, and compares these primary structures with the NCBI non-redundant (nt) nucleotide database (McGinnis & Madden, 2004). Subsequent bioinformatic analyses will be informed by the context of the sample and the purpose of the study; for example, it might include placement of sequences on phylogenetic trees, correlation analysis of sequence information alongside environmental data and statistical analysis of species diversity among samples.

1.1.2. Advantages and Drawbacks of Viral Metagenomics

The primary advantage of the process described above over cultivation dependent approaches is that there is no requirement for the presence of a host cell line. This greatly increases the range of viral sequences that can be described, including viruses for which the host is not known, and the amount of sequence data that can be generated, as there is no need to culture a host.

Additionally, because environmental samples are sequenced directly and not heavily manipulated in the lab, the results provide a more representative snapshot of viral species composition in the natural environment. Viral metagenomics is therefore being widely used for understanding viral ecology and is being used to characterise viral "communities" from the surface atmosphere (Whon *et al.*, 2012) to the deep ocean (Mizuno *et al.*, 2016).

Viral metagenomics then, generates large volumes of data that are problematic to analyse. Metagenomic data is big, typically accounting for many gigabytes, even terabytes of information, and therefore a substantial investment of human and computational time and

effort must be dedicated to its analysis. It is also a nascent field of research and, as in all emerging subjects of scientific endeavour, there are new and significant hurdles to overcome for accurate analysis. Some of these problems are common to all bioinformatic analyses, but the large size and poor quality of viral metagenomic data creates other problems that are specific to this field (Figure 1).

1.2. Difficulties faced by researchers undertaking Bioinformatic analysis

Many of the difficulties in performing bioinformatic analysis on viral metagenomic data are common to any work using bioinformatics software. Whilst undertaking research for my Masters degree, I specifically identified the three following issues:

i. Dependency reliant software;

ii. A lack of standardized formats for modelling and data representation;

iii. A lack of appropriate tools to perform simple computational tasks.

1.2.1. Prohibitive software dependencies

Much bioinformatics software is difficult to run due to a high number of dependencies on other programs that have to be installed prior to using the software. Dependencies are unavoidable in creating software with some complexity; however, often bioinformatics software is published with dependencies that aren't relevant to the function of the software.

A high number of dependencies can arise when bioinformatics tools are created *ad hoc* by research groups during data analysis. If the tool is useful, it is often published as a subsidiary to a data-driven research article. Programs created in this manner are often built into large bioinformatic pipelines that have many version-specific software dependencies. Thus, to run the programs outside of the pipelines, all of the other software on which the programs are dependent, must be installed. Not only is this computationally inefficient, it can prevent users running the software if, for example, version conflicts arise.

This problem has contributed to a lack of reproducibility in computational biology. Installing just one piece of bioinformatics software requires such an investment of time that it becomes unfeasible to try and replicate entire workflows on different machines. Garijo *et al.*, (2013) found that attempting to replicate the bioinformatic analysis of a drug discovery paper took 280 h, 160 of which was spent installing and configuring new software. The authors

highlighted that aside from the time, installing new software can incur a high financial price, as some software requires installation of proprietary dependencies in order to run properly.

Thankfully, protocols are being developed to address the issue of reproducibility. The first involves more stringent rules for accessibility in publishing application notes on bioinformatics software. For example, in order to be published as an Application Note, *Oxford Bioinformatics* requires software to be 'run on nearly all conditions on a wide range of machines' and not 'involve significant investment of time for the user to install' (Anon, n.d.). The second is package management systems like Bioconda, which allow users to install and switch between different versions of software packages on the same machine (Dale *et al.*, 2017).

The final is the use of software containers like Docker (http://www.docker.com). These programs allow developers to host a virtual environment on the cloud in which users can run software. This virtual environment contains all the dependencies required to run the software and users need only supply the input data. Docker represents a big step forward for computational research, essentially future-proofing software from version conflicts, and eliminating the need to install dependencies. In addition, computational performance appears to be only negligibly affected by the use of Docker (Di Tommaso *et al.*, 2015).

Software containers do not however address all the issues outlined above. First, because software containers are not yet widely implemented in bioinformatics software publications, and secondly because software containers are unlikely to improve reproducibility in computational biology. Software containers only allow users to clone and run the workflow of the original authors. They do not make it easier to reproduce a workflow from its constituent parts on a different machine, and crucially do not allow users to freely interrogate and configure those parts, a vital requirement in allowing users to independently validate the performance of software.

1.2.2. Lack of Standardized Formats

Another difficulty is the lack of unanimously used, standardized formats in computational biology. For example, currently there are seven commonly used formats for storing sequence information, eight for sequence alignments and four for phylogenetic trees (Leonard, Littlejohn & Baxevanis, 2007). Outside of the common formats, there are many more hardware- and software-specific formats, and no list of all the formats used in bioinformatics exists. This is problematic, because most bioinformatics software has specific format

requirements for a datatype, and these often differ among programs, so in any analysis, pipeline steps must be included to convert data between datatypes. This issue is summarised in the Roslin Institute's 'First Law of Bioinformatics' (http://bioinformatics.roslin.ac.uk/lawslaws/):

"The first step in developing a new genetic analysis algorithm is to decide how to make the input data file format different from all pre-existing analysis data file formats."

This issue not only affects the representation of data, but also the modelling of data, as there is no standard language to describe models and process descriptions in systems biology. Biological metadata (data describing biological data) is also affected by a lack of standardization. For example, a recent study has shown the variable quality of metadata records in two biomedical databases, NCBI BioSample and EBI BioSamples; many authors omitted field names or populated fields with incorrect data types when publishing biological samples (Gonçalves & Musen, 2019).

Lack of standardized formats means computational analysis is prone to misuse. One example of this is the placement of bootstrap support values on phylogenetic trees. The Newick format, introduced in 1986, is one of the most widely used tree display formats; however, there is still no standard for its use, and therefore no official way it should be interpreted or processed into other formats. A study by Czech *et al.*, (2017) showed how the support values on the same tree are displayed differently on 20 tree viewers.

Recently there have been successful initiatives to address this issue in systems biology. CoMBiNE (Computational Modelling in Biology Network) is a consortium that works to promote open source standardized formats for computational modelling in biology (Hucka *et al.*, 2015), while the Systems biology format converter provides a web platform for converting models among commonly used languages (Rodriguez *et al.*, 2016). In metagenomics, loose standards already exist for describing the entire analysis workflow (STERK *et al.*, 2009), which includes some guidance on output data formats. More stringent guidelines have been proposed that specifically include formats for data analysis outputs (ten Hoopen *et al.*, 2017).

1.2.3. Lack of tools for simple tasks

A final difficulty is the lack of well documented tools to perform simple tasks in computational biology. This difficulty arises because many bioinformatics toolkits are created by research groups made up of computer scientists, who are not interested in, or aware of, the simple tools needed to expedite 'everyday' bioinformatic research. This is compounded by the fact that as more biological data are generated, and new data formats arise, a large variety of simple format-specific programs are required. A cursory scroll through bioinformatics forums reveals frustrated researchers trying to accomplish many simple tasks without the tools to do so.

1.3. Difficulties faced by researchers undertaking Bioinformatic analysis on Viral Metagenomic Data

For researchers doing analysis on viral metagenomic data there are several field-specific issues that must be navigated. These are:

i. The size of the data is 'big'[1],

ii. The quality of the data is often poor,

iii. The sequence data produced often has little similarity to other sequences in databases.

1.3.1. Size of the Data

One of the foremost difficulties in working with metagenomic data is that the data are 'big'; sequencing runs can generate millions of short reads that occupy terabytes of space. Assembly reduces the size of these files, but the output files still often occupy several hundred gigabytes and consist of hundreds-of-thousands of contigs. The size of these datasets is such that visualising them with GUI-driven software (e.g. Excel, Geneious) is unfeasible; therefore, analysis must be automated from the command line. The size also means that any software involved in analysis, even if the task is simple, must be designed to process the data in a "reasonable" time. In recent years, assembly and alignment algorithms have been

[1] The question of whether genomic data is 'big data' is unclear. 'Big Data' is a loosely defined and by some interpretations would include metagenome studies. An early and much cited definition of Big Data by the Gartner IT glossary (https://www.gartner.com/it-glossary/) precludes genomic data. Gartner defines big data as data meeting the three V criteria: Velocity (meaning the data is received in a continuous stream and processed in real-time), Variety (the data is unstructured and consists of several formats: e.g. image, text, audio) and Volume (the data is so big it can't be handled by standard machines and software). Genomic data meets only the latter of these criteria.

developed to handle large metagenomic datasets; for example, DIAMOND is a sequence aligner for translated DNA searches against the NR protein database. It essentially performs the same function as BLASTx, but 500 to 20,000 times faster (Buchfink, Xie & Huson, 2015).

1.3.2. Quality of the Data

Another problem is the quality of the sequence data produced. The quality of viral metagenome data is affected by contamination, either introduced because of ineffective filtering or cross-contamination in the laboratory, and errors introduced during assembly (Thurber *et al.*, 2009).

Contamination primarily occurs because the size and density of microbial fractions overlap, so it is very difficult to completely filter the viral components away from the cellular ones. Another drawback of filtering is that it cannot separate viral sequences integrated into a host-cell genome, from those of viruses replicating in cells, or attached to particles.; this may constitute a significant amount of viral diversity, as it has been shown 60% of sequenced bacterial genomes contain at least one prophage (Casjens, 2003).

A second cause of contamination is that, viral metagenomic data are often generated as part of wider-scope metagenome studies, which also look at prokaryotic and eukaryotic microbial communities. Often the preparation of all three fractions occurs in the same laboratory, so the risk of cross-contamination is high.

Overall, the combined effect of laboratory contamination and poor filtration mean that often a large proportion of the viral fraction in a metagenome study is non-viral - for an extreme example over 90% of the DNA viral fraction produced during the watershed metagenome project was of bacterial origin (unpublished findings, Uyaguari-Diaz *et al.*, 2016), and this observation has been made elsewhere (Rose *et al.*, 2016) - while certain components of the viral community, including giant viruses and prophages, are lost in the cellular fraction. Hence, analysis of viral metagenomic data requires computational filtering to remove contigs that are cellular in origin.

The quality of the sequence data is also affected by errors introduced from assembling contigs. The assembly program used, and the parameters it is configured with, affect the accuracy of the output assembly (Sczyrba *et al.*, 2017). Misassemblies occur for a variety of reasons. Assembly algorithms may 1) confuse closely related genomes by considering SNPs

to be errors in base-calling (Edwards & Rohwer, 2005), 2) be unable to identify repeated sequences, and exclude them from the assembly, or insert identical repeats where there is repeat polymorphism (Phillippy, Schatz & Pop, 2008), 3) and fail to circularise genomes and add repeated regions to the end of assemblies (Hunt *et al.*, 2015). Creating an effective assembly process requires testing various assemblers against one another to achieve the optimum assembly, and then identifying and correcting errors in the assembly.

1.3.3. Content of the Data

A final problem specific to viral metagenomic data is that the majority of sequences returned are do not match any known genome sequences, and thus hard to correctly identify and annotate as viral sequences. In the first environmental metagenomic sequencing studies 65% of the viral sequences uncovered had no recognizable sequence identity to anything in Genbank; in bacteria this figure was 10% (Edwards and Rohwer, 2005). Assigning these sequences to an existing taxon can be problematic using a conventional tBLASTx approach as a significant proportion (10 – 90%) will be returned as unknown (Huson *et al.*, 2007). The problem is compounded by the occurrence of lateral gene transfer between virus and host, which means that virus genomes can share homologous regions with eukaryotic and bacterial genomes.

Alternative approaches to taxonomic identification have been proposed. These include the following: 1) referencing unknown sequences against a database of conserved protein domains (as opposed to referencing against the NR database which contains all published protein sequences), as conserved protein domains are more informative for purposes of viral taxonomy (Zhang & Sun, 2011); 2) doing phylogenetic analysis on query sequences (i.e. assume that homologous sequences will group together on a phylogenetic tree and build the tree from query sequences and reference sequences) (von Mering *et al.*, 2007) and 3) identifying sequences based on nucleotide characteristics rather than sequence identity (e.g.. extract features of the sequence, such as the percent of A,C,G and T and use this to identify the genome (Perry & Beiko, 2010)), virfinder, for example, uses a machine-learning-derived algorithm to identify viral contigs by the k-mer frequency (i.e. the length of the contig expressed in the number of short reads of length k that would fit into the contig) (Ren *et al.*, 2017).

For sequences identified as viral, it can be difficult to place them phylogenetically. Unlike bacteria and eukaryotes, there is no gene universally found in all viruses that can be used to

make phylogenetic comparisons. Viruses must instead be distinguished by common genes such as replication or capsid proteins; however, there are difficulties associated with the use of these genes, such as genes with different evolutionary histories which encode the major capsid & replication proteins (Zhong & Jacquet, 2014; Schulz *et al.*, 2018).

Making accurate trees is also exacerbated because assembly often results in contigs that do not cover the entire gene and the sequence reading frame can be ambiguous. When generating phylogenies using gene fragments trees can have large branch lengths and low support values. One solution is to use a tool such as the RaXML Evolutionary Placement Algorithm (EPA), or pplacer, which shows the possible positions of a gene fragment on a reference tree with statistical likelihood scores; thus, providing a more appropriate alternative to BLAST-based phylogenetic placement (Berger, Krompass & Stamatakis, 2011; Matsen, Kodner & Armbrust, 2010).

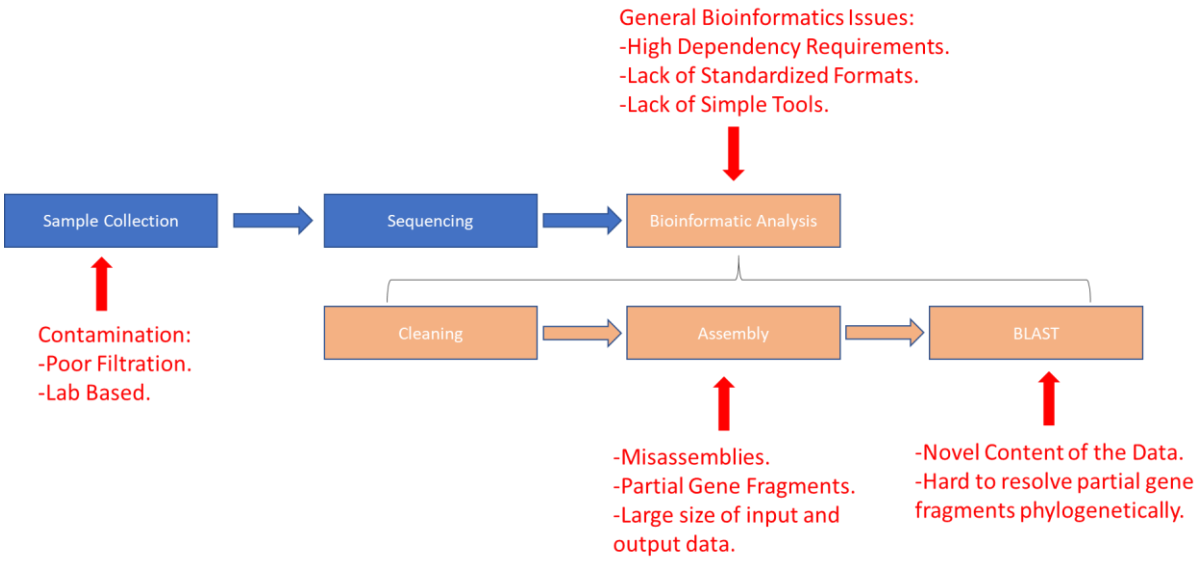

Figure 1 Viral Metagenomic Pipeline. The general and field specific difficulties of doing computational analysis on viral metagenomic data are displayed with red text.

1.4. Research objectives and thesis structure

Overall, bioinformatic analysis of viral metagenomes is made difficult by a wide range of factors; some of these are due to issues in the way bioinformatics software is written and published, and others because of the exceptional quality and content of viral metagenomic data

Driven by a desire to improve bioinformatic analysis of viral metagenomes, I wanted to create software capable of automating a variety of data processing tasks that would otherwise have to be done manually. I also wanted to write software that allowed viral metagenomes to be analysed in new ways.

My philosophy in writing software was to make programs that were openly accessible, well documented, fast and easy to use. To achieve this, I created programs in Python or BASH that required only one or no dependencies to run, processed easy to obtain file formats and ran quickly on big datasets, even on a midrange spec laptop.

In chapter two, I describe four new computational tools, VHost-Classifier, Optimal-Translate, Simple-Circularise and Bootstrap-Jplace, that I created for working with viral metagenomic data. The latter three tools automate data processing tasks, increasing the speed at which further bioinformatic analysis can be performed. The first tool, VHost-Classifier, allows viral metagenomic datasets to be analysed in a new way, namely by filtering the viruses returned from a metagenomic sequencing run by their host organism. In the final chapter, I perform a host analysis of all viruses currently published on the NCBI taxonomy database using this software. The results highlight a major sequencing bias toward viruses infecting humans.

Chapter 2: Development of software tools for working with viral metagenomic data.

2.1. VHost-Classifier

2.1.1. Introduction

BLAST analysis of viral metagenomic data against other databases often returns many matches (i.e. hits) to other sequences from a diverse range of viruses; however, a researcher may only be interested in viruses that infect specific hosts. In order to filter the results to return only those sequences associated with viruses infecting specific hosts, the analysis must manually be filtered using string lookups, or referenced against a database that has curated virus host information, such as the Virus-Host DB (Mihara et al., 2016). Manually filtering the results is untenable given that metagenomic runs produce hundreds of thousands of virus reads. Filtering the results using the Virus-Host DB is more viable, however as the database contains <10% of viral sequences on NCBI, many hits from the analysis will not have a host assigned within the database.

Here I present VHost-Classifier, a natural language processing algorithm to automate virus-host classification on a list of viral taxon IDs (vtaxonIDs) that are assigned to sequences during a BLAST search. It groups vtaxonIDs based on the evolutionary lineage of their host and is therefore useful for filtering vtaxonIDs, or for giving an overview of host diversity for viruses found in a BLAST search, by listing hosts of related viruses.

2.1.2. Materials and Methods

VHost-Classifier is written in Python 3 and requires pre-installation of the ETE3 toolkit to run (Huerta-Cepas et al., 2016). In the default behavior it takes a list of Taxon IDs as input (one Taxon ID per row), which can be extracted as a column from the output of a BLAST analysis. For more information visit: https://github.com/Kzra/VHost-Classifier (Code Appendix, C1).

The pipeline used by the VHost-Classifier algorithm (Figure 2 - 1) first references the taxonID against the Virus-Host DB and if a host taxon is found it is taken directly from the database. If not found, the taxonID is converted to its English name and checked to make sure it is a virus, based on whether it contains a virus descriptive string (e.g. ‘Virus’,’Phage’,’Satellite’) in its name. If it is a virus the English name is parsed against a

database of common animal names, and if any common animals names are found they are converted to scientific names that can be used as look-up strings against the NCBI taxonomy database.

Once the conversion is done, the various words in the virus name are parsed to identify the most likely look-up string that identifies the host. To achieve high accuracy, several rules are followed, as detailed below.

First, the viral prefix (-noro -mega etc.) is ignored, as it is might be confused for the name of a host taxon (e.g. '-noro' may be confused for 'Noronhia', a genus of flowering plant).

Second, if a string is a genus name, it is concatenated with the following string to make genus and species a single string. This is because species and genus names are confounded across the tree of life, so must be used together in order to be a unique look-up string (e.g. if the virus name is 'Prunella vulgaris virus 1', the string searched is 'Prunella vulgaris', as using either the genus or species name alone is ambiguous; both describe multiple taxa in different evolutionary lineages).

Third, if the virus is a strain of Influenza or Norovirus (> 75% of all virus taxonIDs belong to these two taxa), the strain information is also parsed to identify a host. If a specific host cannot be identified for these viruses, because the words in the name do not contain useful host information, it is set as default to be Mammals or Aves depending on the virus subtype (e.g. Influenza A subtypes without strain information are assigned to Aves whilst Influenza B, C, D are assigned to Mammals).

Finally, when choosing between multiple valid look-up strings, basic conventions of English are followed. For example, 'Elephant seal virus' is assigned to a seal not an elephant, as in this case 'Elephant' is acting as an adjective and the virus infects a seal.

Once a look-up string is identified it is used to reference the NCBI taxonomy database, using the ETE3 python toolkit (Huerta-Cepas et al., 2016). The evolutionary lineage of the host is parsed and used to bin the input taxonIDs into a directory tree resembling a phylogenetic tree (Figure 2). Each directory contains csv files that contain the taxonIDs belonging to a particular taxon, and the index positions of these taxonIDs in the original input file. There is also a Counts.csv file that gives the numbers of vtaxonIDs assigned to each taxon within that rank. By default, hosts are binned to the ranks phylum, class and order but this can be set by the user to phylum, order and family.

If a host cannot be assigned to a virus, the taxonID is referenced against a customized version of the IMG/VR database (containing only metagenomic and isolate viral sequences) and a set of if-statement rules in order to predict the environment it was sequenced from (Paez-Espino et al., 2017). These taxonIDs are binned according to environment in a separate lineage of the directory tree.

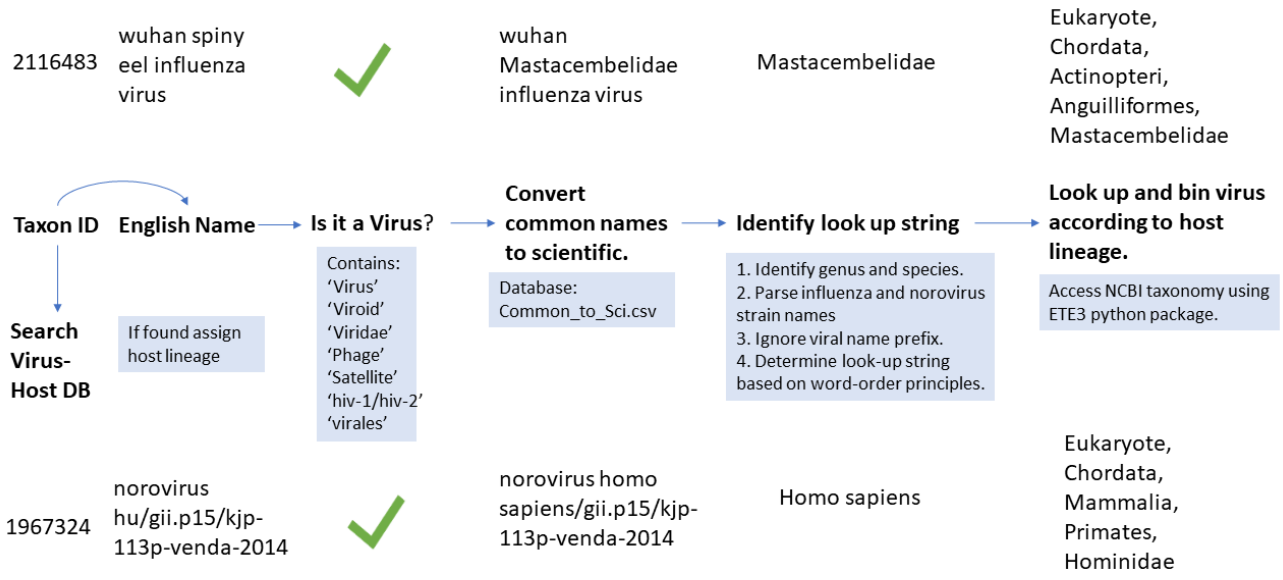

Figure 2 Pipeline used by VHost-Classifier Algorithm.

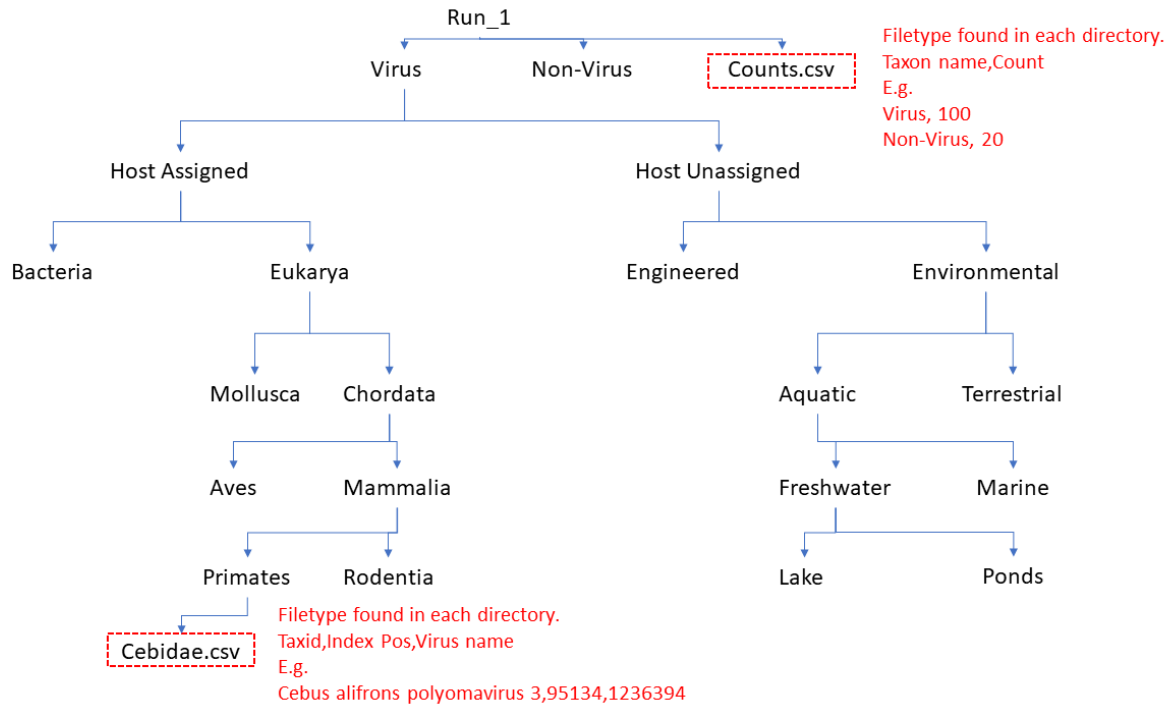

Figure 3 Directory tree that VHost-Classifier bins input taxonIDs into.

2.1.3. Results

When tested and manually checked on a subsample of 1000 vtaxonIDs chosen at random using the Python 'random' module, from the 191,408 vtaxonIDs present on the NCBI taxonomy database (downloaded June 2018), the program assigned a host to >95% of viruses with an overall accuracy of 100%. Over 90% of hosts were resolved to the rank of class, and 37% could be resolved to the rank of family (Table 1). Coverage of the subsample dropped sharply when assigning a host to the rank of order or family; this is because many viruses on NCBI taxonomy are strains of Influenza or Norovirus that do not have host information in the name. The software assigns a class to these viruses depending on their subtype (e.g. Influenza A is assigned to Aves) but cannot resolve the host any further.

The recall score reflects the percentage of vtaxonIDs to which the authors could assign a host when the software could not. For each rank assignment the recall score was substantially higher than the coverage, demonstrating that in most cases when the software couldn't assign a host to a vtaxonID, it was because the vtaxonID did not have enough informative host information in its virus name, and not because the software overlooked useful information.

Based on the virus names, of the 1000 vtaxonIDs queried the authors could assign a host to nine of the 47 viruses, when the software could not assign a host or habitat (Table 2). In these cases, the virus names either contained common names of animals, not present in the Common_to_Sci conversion database (e.g. 'threespine stickleback iridovirus') or unusual characters in the name string (e.g. 'cyclovirus ng_chicken 3). A full list of the virus taxonIDs in the subsample and their names can be found on the GitHub page.

The software was able to resolve a host to the rank of class for 93% of the 191,408 vtaxonIDs in the NCBI taxonomy database in under three hours using our lab server (2x Intel Xeon 2GHz, 32 cores, 512 GB RAM). It is worth noting that there is a strong bias in the hosts of published viruses toward viruses of Chordates (>90% of viruses published on NCBI), and this was reflected in the subsample.

To demonstrate the ability of the software to classify hosts from a real-world metagenomic study, we ran the software on the taxonIDs of ~30k viruses returned from a BLAST analysis of the DNA fraction generated by the Watershed Metagenome Project (Uyaguari-Diaz et al., 2016). It is typical in viral metagenomic studies for many sequences to be assigned as uncharacterized viruses or environmental sequences, without discernible host information in the name. Therefore, in this analysis the coverage scores were lower: 52.6%, 51.4%, 46%, 51.8% and 50.8% for superkingdom, phylum, class, order and family host assignments respectively. In this case, there is an increase in coverage from class to order because some hosts lack taxonomic information for class, but have it for order.

2.1.4. Conclusions

VHost-Classifier is a tool that will facilitate researchers working on viral metagenomic data by enabling fast filtering of the output from a BLAST search by virus host. In addition, it can give a broad insight into the composition of viruses in an environmental system by their host, enabling new and interesting questions to be asked during viral metagenomic research.

Table 1 Benchmarking accuracy and recall on 1000 randomly selected viral taxonIDs.

Assignation	Accuracy (%)	Recall (%)	Coverage (%)
Superkingdom	100	100	95.5
Phlyum	100	99.7	95.5
Class	100	99.7	93.2
Order	100	84.0	45.5
Family	100	90.7	37.4

Note: "Accuracy" corresponds to the placement of a vtaxonID in the correct taxon corresponding to rank. "Coverage" corresponds to the percentage of vtaxonIDs for which an assignment was made. "Recall" is a measure of the number of vtaxonIDs the authors could assign a host for, based on the virus name, when the software could not (if 100% the software did not overlook any vtaxonIDs).

Table 2 Virus names VHost-Classifier was unable to process.

Listed below are the VTaxids in the random subsample that VHost-Classifier was unable to assign a host to.

VTaxids highlighted in light blue are ones that the authors could assign a host for, and thus reduced the recall (%) of the software.

For a full list of the VTaxids in the subsample and their English names, see the GitHub page.

VTaxid	English Name
908071	bhendi yellow vein mosaic betasatellite [india:aurangabad:oy165:2006]
2023329	phage gp23
1195210	picobirnavirus monkey/chn-41/2002
1128121	threespine stickleback iridovirus
908074	bhendi yellow vein mosaic betasatellite [india:kaivara:oykaivara:2006]
1927956	levior01 phage
409026	phage swp3
1977143	saetivirus
1717154	astrovirus predict mastv-128
1168596	betanodavirus s1.tun01.11
2056547	yado-kari virus 4
444049	tt virus sle2052
1511856	alphaspiravirus
1164802	reovirus gcrv104
1690708	picobirnavirus predict_pbv-12
1972615	joinjakaka hapavirus
742991	cyclovirus ng_chicken3
1888317	duwamo virus
1511778	mischivirus
1678142	piscihepevirus
11094	negishi virus
221982	recombinant m-mulv/ralv retrovirus
562945	mesta yellow vein mosaic virus-[india:haringhata:2008]
1189585	mimivirus cher
1195231	picobirnavirus monkey/chn-76/2002
11742	visna lentivirus (strain 1514)
430592	visna/maedi virus 1514
1679901	coronavirus rcov/d_rmu10_592/mic_arv/ger/2010
1086585	flavivirus cjd05/chn/2010
2021933	lothians earthworm bunya/arena-like virus 1
931923	t7-like bacteriophage eptesicus fuscus/p1/it/usa/2009
1795436	guarani virus
1346296	betacoronavirus btcov/rhi_bla/bb98-22/bgr/2008
186768	mitovirus
1690702	picobirnavirus predict_pbv-114
694466	mulberry cryptic virus 1
12283	nodaviridae
693997	alphacoronavirus 1
1268011	tigray hantavirus
1123947	phlebovirus jn1/china/2010
35276	unclassified retroviridae
436675	jabora hantavirus
2021952	caledonia partiti-like virus
478853	picornaviridae str. 06-646
2170595	notori virus
741337	bocavirus pig/china/2009
1961786	taupapillomavirus 3

2.2 Optimal Translate

2.2.1. Introduction

The process of sequence assembly results in the creation of contigs that may represent full genomes, full genes, or partial gene fragments. If the gene is incomplete it is not always apparent which of the six possible reading frames (three forward, three reverse) is the correct translation. This information is required in order to make accurate amino-acid-based alignments and phylogenetic trees, as the primary amino-acid sequence is dependent on the translation frame.

One solution is to use an ORF (open reading frame) finder which locates stretches of DNA between a START (e.g. AUG) and a STOP codon (e.g. TAA, TAG or TGA); however, START or STOP codons may not be present in an incomplete gene fragment, or there may be multiple START/STOP stretches that could be used and is not immediately obvious which one is correct. On Geneious, a widely used GUI (graphical user interface) driven bioinformatics program, there is no alternate tool to determine the optimal reading frame, nor is there a way to automate ORF detection on multiple sequences (Kearse *et al.*, 2012). Currently, users must manually calculate, choose and translate the correct reading frame, a procedure which becomes laborious for long lists of sequences. A simple computational solution is to translate a sequence in every possible reading frame and choose the one that has the least number of STOP codons, as this is most likely to represent the true reading frame.

2.2.2. Materials and Methods

Optimal Translate is written in Python 3 and requires no additional dependencies to run (Code Appendix, C2). It is available to be installed as a plugin extension to Geneious 11. It can be run on a single sequence or several sequences in a FASTA file. For more information visit https://github.com/Kzra/Optimal-Translate.

The software transcribes each sequence in all six frames and counts the number of STOP codons in each. It then writes the original sequence in the optimal frame to an output FASTA file, with the frame name appended to the contig name. In cases where there are multiple optimal frames Optimal Translate writes both into the output FASTA file.

2.2.3. Results

When nine circovirus replication-protein contigs extracted from DNA virus data from the Watershed Metagenome Project (Uyaguari-Diaz *et al.*, 2016) were aligned before and after running Optimal translate, the software greatly improved the quality of the alignment (Figure 3).

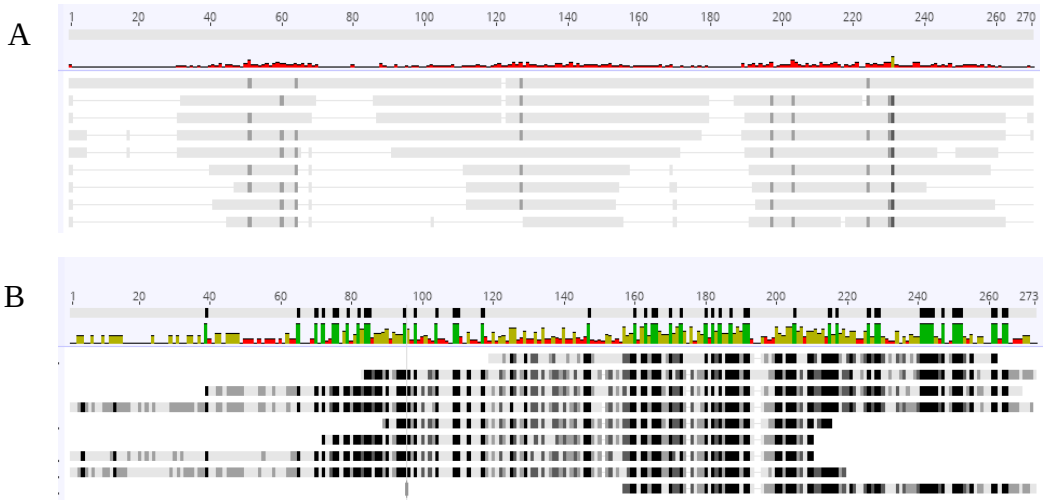

Figure 4 Multiple alignment of nine circovirus replication protein fragments, before (A) and after (B) running optimal translate.
The images are screen shots of alignments made in Geneious 11. The numbers refer to nucleotide positions, the light grey segments are sequences, the black segments are regions of similarity among the sequences. The histogram records the mean pairwise identity among sequences at a given nucleotide position, if the bar is green it means all sequences are 100% identical at that position. In both cases the following alignment conditions were used, Type: Global Alignment with free end gaps, Cost Matrix: Blosum 32, Gap open penalty: 12, Gap extension penalty: 3, Refinement iterations: 2.

2.2.4. Conclusions

Optimal Translate is one of the few tools available online that allows contigs to be translated in the correct reading frame without relying on ORF detection of START and STOP codons. Additionally, it is the only plugin available on Geneious 11 that can automate translation in different frames on a multi-fasta file. This makes it a useful tool for metagenomic researchers who are dealing with gene fragments and analysing them using Geneious.

2.3. Simple Circularise

2.3.1. Introduction

Once full sequences have been acquired and translated in the right frame, there is often additional processing that needs to be done to remove artifacts acquired in the process of assembly. One difficulty researchers encounter is that circular genomes are sometimes shown by the assembler as being linear. This may be due to the assembler missing a region of the contig, so the circular genome is incomplete, or because the assembler overlaps the ends of the contig, so the circular genome appears linear.

Currently, the only software that corrects for either error is Circlator (Hunt *et al.*, 2015), which re-assembles reads on the ends of contigs, with algorithmic preference to producing a circular genome. The process of re-assembly is computationally laborious and requires a great deal of pre-requisite software. In cases where a genome hasn't circularised due to overlapping ends, it is also redundant; re-assembly of terminal reads is not needed to circularise a contig, as the full circular genome lies inside the contig. A simple computational solution to circularising such a contig is to find repeated sequences inside the contig of such length as to be statistically significant, and to circularise the genome around these regions.

2.3.2. Materials and Methods

Simple Circularise is written in Python 3 and requires no additional dependencies to run (Code Appendix, C3). It can be run on a single sequence or list of sequences and used as a plugin extension on Geneious 11. For more information, visit https://github.com/Kzra/Simple-Circularise.

With its default settings, Simple Circularise uses the binomial formula, or for genome sizes over 10kb the poisson approximation of the binomial, to estimate the minimum size of repeated sequence to look for (Equation 1: A & B). Users are able to change the probability threshold for determining repeat size (default 0.005) and specify a minimum and maximum output genome size. Simple-Circularise will return the largest sequence it can for the criteria given; this is the closest approximation of the circular genome. Additionally, users can change the behaviour of the software to maximise repeat size (as opposed to output genome size) (Figure

4).

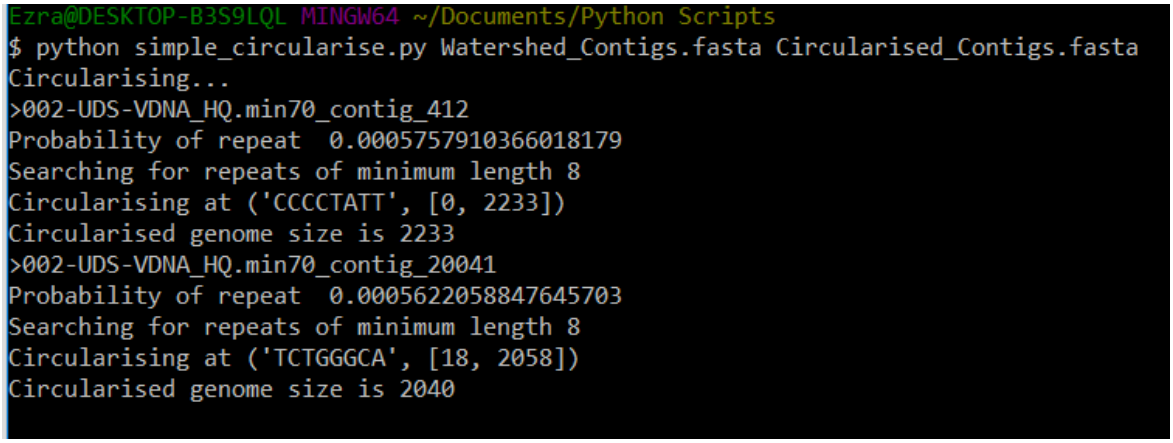

Figure 5 Terminal display of Simple Circularise working on several sequences in a FASTA file.

$$\mathbf{A} \quad P(x) = \frac{n!}{(n-x)!\,x!}.p^x.q^{n-x} \qquad \mathbf{B} \quad P(x) = \frac{\lambda^x.e^{-\lambda}}{x!}$$

Equation 1 Binomial formula (A) and Poisson Approximation of Binomial (B) used by Simple Circularise to compute the length of sequence overlap to search for. Where P(x) is the probability of x successes, n is the number of events, x is the number of successes, p is the probability of a single success, q is the probability of a single failure, and $\boldsymbol{\lambda}$ = np.

2.3.3. Results

Compared with Circlator, Simple Circularise is a faster and more accessible tool for circularising genome assemblies that have failed to circularise because of a fragment or whole genome overlap. However, because the software does not involve a re-assembly process, it is unable to extend a contig with a missing region, and therefore will not work on truncated assemblies. (Table 3).

Table 3 Comparing performance between Circlator and Simple Circularise.

Software:	Circlator	Simple Circularise
Dependencies:	BWA version >= 0.7.12 prodigal version >= 2.6 SAMtools (versions 0.1.9 to 1.3) MUMmer version >= 3.23 Canu and/or SPAdes. SPAdes version 3.6.2 or higher is required, but 3.7.1 is recommended (marginally gave the best results on NCTC data from the Circlator publication, tested on all SPAdes versions 3.6.2-3.9.0).	Python 3
Works on:	Fragment Overlap, Genome Overlap, Missing Region	Fragment Overlap, Genome Overlap
Speed (default behaviour):	Slow: > 4 minutes for 3GB genome circularization.	Fast: < 1 minute for 3GB genome circularization.

Note: Speed benchmarks for Simple-Circularise were performed on Dell XPS 15 laptop, with 16GB RAM and i7-7700 HQ CPU, Circlator speed benchmarks were taken from the supplementary information of the original publication (Hunt *et al.*, 2015).

2.3.4. Conclusions

Simple Circularise is an easy to use tool that allows metagenomic researchers to circularise overlapped genome assemblies without having to install and run assembly software. As such it has become a popular tool with metagenomic researchers. Since it was published on GitHub it has been downloaded over forty times, starred (highlighted on another researcher's software page) and forked (incorporated into another researcher's pipeline).

2.4. Bootstrap Jplace

2.4.1. Introduction

When partial gene fragments are assembled from a metagenomic sequencing run, to assess biological diversity it is of interest to see where the fragments are located on a reference phylogenetic tree. Creating a tree through standard alignment algorithms and tree builders is not suitable for visualizing partial gene fragments as they align poorly with reference sequences, so produce unrealistic trees with long branch lengths and low bootstrap values. One solution conceived by the RaXML research group is the evolutionary placement algorithm (EPA), instead of making additional branches, reads are assigned to the node metadata and can be visualised as branch symbols (Berger, Krompass & Stamatakis, 2011). This is useful as it results in neatly arranged trees that give information about the diversity of short reads. EPA produces trees in a unique jplace format, which is supported by several tree viewers including interactive Tree Of Life (iTOL) (Letunic & Bork, 2007).

One issue with the jplace format is that it cannot support bootstrap values, so it is impossible to have a tree displaying both evolutionary placements of short reads and confidence values. This is a frustrating for researchers who want to publish the results of EPA, and the current solution is to publish two trees, one with confidence values (in newick format) and the other with short read placements (in jplace format), side by side (https://groups.google.com/forum/#!topic/raxml/V7ZS5dhffgQ).

A computational solution to this problem is to to use the dataset feature included with iTOL; this allows the labelling of a tree with additional metadata not included in the tree file. By comparing the newick and jplace files it is possible to write a dataset file that can retroactively label a jplace tree with bootstrap values.

2.4.2. Methods and Materials

Bootstrap Jplace is written in BASH and requires no additional dependencies to run (Code Appendix, C4). It takes a reference tree in newick format and an EPA tree in jplace format as input and produces a dataset that can be used to add bootstrap values to an EPA jplace tree viewed on iTOL. For more information visit https://github.com/Kzra/Bootstrap-Jplace.The script works by comparing the jplace and newick trees; it reads the bootstrap node values of the newick tree and assigns them to the corresponding node on the jplace tree. It stores this

information in the format of an iTOL dataset, which can then be used to add bootstrap values to the jplace format produced by EPA.

2.4.3. Results

Using the Bootstrap Jplace BASH script it is possible to create bootstrapped jplace trees showing both placement information and confidence values.

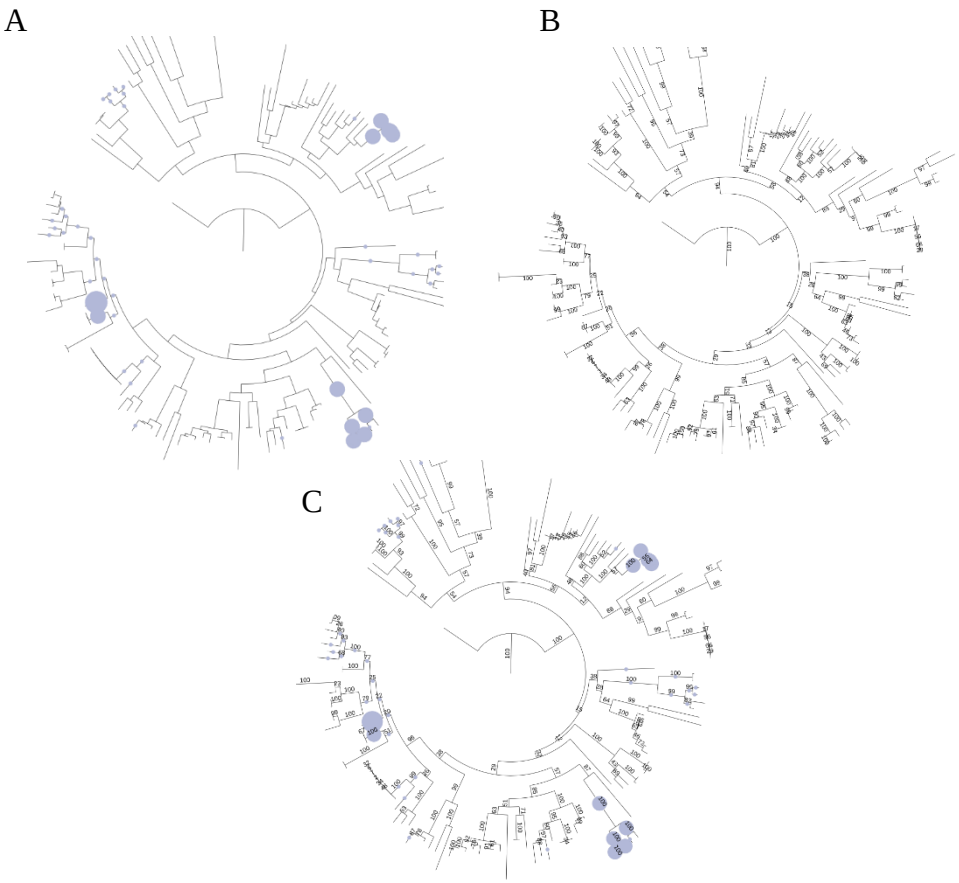

Figure 6 Phylogenetic trees visualised on iTOL.
Tree (A) is a jplace tree showing evolutionary placement, tree (B) is a newick tree showing bootstrap values and tree (C) is a combined tree made using Bootstrap Jplace showing both placement position and bootstrap values.

Chapter 3: Virus and Prejudice: >80% of viral taxa on the NCBI taxonomy database belong to five genera, all infectious to humans.

3.1. Summary

Viruses are the most abundant biological entity on the planet; despite this, our understanding of viruses is heavily skewed toward those that cause or are associated with disease in humans. To quantify the scale of this bias, this study performed a comprehensive host analysis of virus taxonIDs in the NCBI taxonomy database. The analysis was performed using VHost-Classifier, a program that uses a natural language processing algorithm to assign a host to a virus based on words in its name. More than 80% of virus taxonIDs belong to just five genera of viruses, all of which are pathogenic to humans; moreover, virus taxonIDs were assigned to only 39 host phyla, which is less than 20% of the phyla identified from nature. Understanding the current sequencing effort bias highlights the vast reservoir of unexplored genetic diversity in viruses and reveals neglected areas of concern, include viruses of antibiotic producing bacteria, zoonotic viruses, viruses infecting wild animals and plants, and viruses of microbes that play key roles in biogeochemical cycling and carbon sequestration.

3.2. Background

Viruses have been studied for decades; yet, there is a lack of knowledge about the overall host range of viruses we have described (i.e. how phylogenetically diverse are the hosts of sequenced viruses), and the weighting of published virus genomes per host taxon (i.e. for a given host genus, family, order etc. how many virus sequences are published on NCBI?). This lack of knowledge may in part be due to difficulties in assigning viruses to organismal hosts. One issue is that viruses cultured in the laboratory may be isolated from a host but infect a different host in nature. Another is that an increasing number of viruses are discovered through metagenomics and, therefore, lack cultured representatives with host information. Finally, a single virus may be able to infect cells belonging to different species; thus, designating a single host can be subjective. Nevertheless, many virus genomes are published with a host organism in the name and supporting metadata, and it is therefore possible to get at least a coarse overview of virus-host assignment. Two reasons to be interested in virus host assignment are outlined in the following paragraphs.

First, it is useful to know the number of described viruses that are associated with each host taxonomic rank. For example, how many viruses have been sequenced for each of the

recognized 226 phyla of organisms, one of the most basal ranks? If the number is low, it suggests there is a wealth of untapped viral genetic information to be discovered, as viruses are genetically specialised to their host organism(s).

Second, quantifying the scale of anthropocentric bias in the selection of viruses that are sequenced, can help guide future sequencing efforts. It is unsurprising that most sequenced viruses cause human disease, but this bias ignores emerging concerns of zoonosis, the transmission of viral infections from one host species to another, which can occur between distantly related organisms (Yolken *et al.*, 2014). Recent studies estimate that unknown viruses account for over 99.9% of potential zoonoses (Carroll *et al.*, 2018b). If only viruses associated with known human diseases are sequenced, it handicaps our ability to recognize emerging zoonotic diseases. In order to prevent significant loss of life during the early stages of an outbreak, epidemiologists need to act on sequence data within a matter of weeks (Gardy, Loman & Rambaut, 2015). Retroactively sequencing a new pathogen is not nearly as effective as having a database of sequence information about related pathogens in advance.

In this chapter I use Natural Language Processing (NLP), the interpretation of human language by computers, to process virus names using rules of the English language and taxonomic naming conventions to predict their hosts. I use VHost-Classifier, a NLP algorithm to assign a host to the current 191,463 viruses with taxonIDs in the NCBI taxonomy database.

3.3. Materials and methods

3.3.1 Virus Host Classification

Virus host classification was performed on all 191,463 virus taxon IDs (vtaxonIDs) in the NCBI taxonomy database downloaded in late June 2018. Virus classification was done using VHost-Classifier, a program that predicts viral hosts based on words in the names of the viruses; it then groups viruses based on the evolutionary lineage of their predicted host (Kitson & Suttle, 2019). For more information visit: https://github.com/Kzra/VHost-Classifier. (Code Appendix, C1).

VHost-Classifier assigned a host at the resolution of phylum, class, order and family to 95%, 93%, 43% and 37% of viruses in the NCBI database, respectively. The 5% of viruses for which the software did not assign a host, were largely environmental viruses with no host information in the name.

The raw results from the virus host classification used to make the analyses presented in the results are available on the Github page.

3.3.2. Rank Information

Data on phyla, order, class and genera of hosts were downloaded from NCBI taxonomy in late June 2018.

3.4. Results

3.4.1. For each taxonomic rank the majority of potential host taxa did not have viruses assigned to them.

First, to see the coverage of viruses for each rank, the number of taxa in a rank to which virus taxa were assigned was compared to the total number of taxa in that rank. At the highest taxonomic resolution, virus taxonIDs were assigned to 39 phyla of a total of 226. The best coverage was for Eukaryota (38%) and the worst for Bacteria (9.2%). At the lowest taxonomic resolution, viruses were assigned to 738 out of 8881 families. The best coverage was for Archaea (27.27%) and the worst for Eukaryota (7.60%). The best overall coverage was for class (27.16%) and the worst for family (8.32%) (Table 4). It is important to note that ranks (in terms of genetic distance) are not directly comparable between super kingdoms, nor are the bacterial and eukaryotic taxonomic trees resolved to the same extent; this may account for the opposite trends in coverage from Phylum to Family between Bacteria and Eukaryota.

Table 4: Viruses assigned to rank.

	Viruses Assigned	Total Number	% Coverage
Phylum	**39**	**226**	**17.26**
Archaea	*4*	22	18.18
Bacteria	*14*	152	9.21
Eukaryota	21	55	38.18
Class	**93**	**335**	**27.76**
Archaea	7	16	43.75
Bacteria	23	90	25.55
Eukaryota	62	229	27.07
Order	**311**	**1471**	**21.14**
Archaea	10	26	38.46
Bacteria	57	203	28.08
Eukaryota	244	1233	19.79
Family	**738**	**8881**	**8.32**
Archaea	12	44	27.27
Bacteria	102	492	20.73
Eukaryota	624	8210	7.60

3.4.2. The host distribution of published virus taxa is disproportionately skewed toward certain host taxa.

Next, among the host taxa for which viruses were assigned, the ten taxa with the most viruses assigned were plotted. Virus sequencing efforts are heavily biased towards viruses of birds and mammals, in particular the families comprising humans (Hominidae), pigs (Suidae), waterfowl (Anatidae) and chickens (Phasianidae). Chordates are hosts to above 90% of the viruses, based on the virus taxonIDs on NCBI, while Aves and Mammals account for 52% and 45%, respectively. The high numbers of virus taxonIDs assigned to Aves in the rank of class is likely an artefact of the VHost-Classifier analysis which assigns Influenza A subtypes without host information in the name to Aves. (Figure 6). In reality, based on naming convention (Anonymous, 1980) these should be assigned to human hosts.

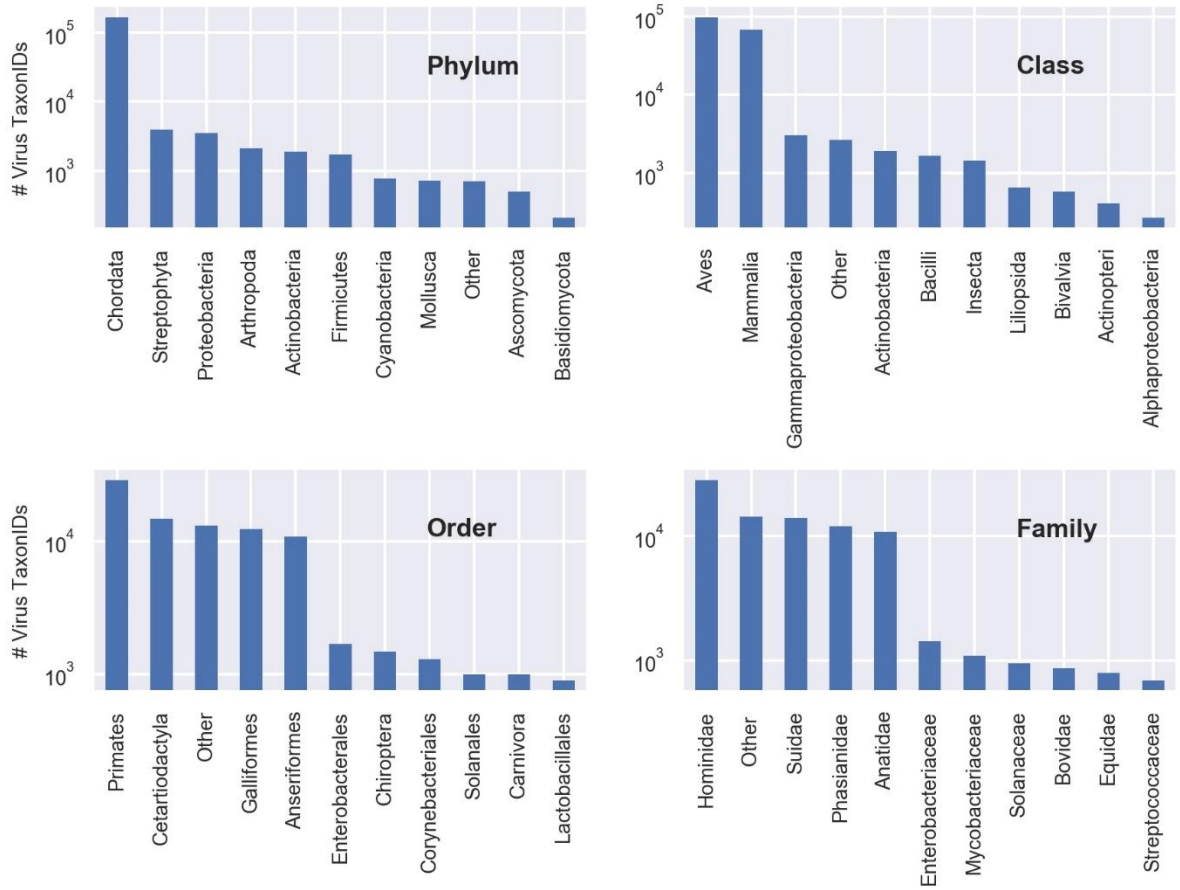

Figure 7 Hosts of Viruses. Hosts of virus taxa classified to phylum, class, order and family rank. The top ten hosts for each rank are shown; other comprises all other hosts within that rank. Virus taxonIDs to which a rank couldn't be assigned are not shown.

3.4.3. The number of members within a host taxon is a moderate predictor of the number of viruses assigned to that taxon.

To determine whether the number of taxa within a class is related to the number of virus taxonIDs assigned to that class, 91 classes with virus taxonIDs assigned to them were plotted against the number of taxonIDs assigned to each class. There was a moderate correlation between the number of taxonIDs within a class and the number of virus taxonIDs assigned to that class ($\tau b = .55$, $p = 4e\text{-}14$), even if chordates are excluded ($\tau b = .52$, $p = 6e\text{-}12$). Especially neglected classes are Insecta and Gammaproteobacteria, while overrepresented classes include viruses of Chordates (Aves, Mammalia), as well as the Mamiellophyceae (Green Algae) & Meristoma (Horseshoe crabs). (Figure 7)

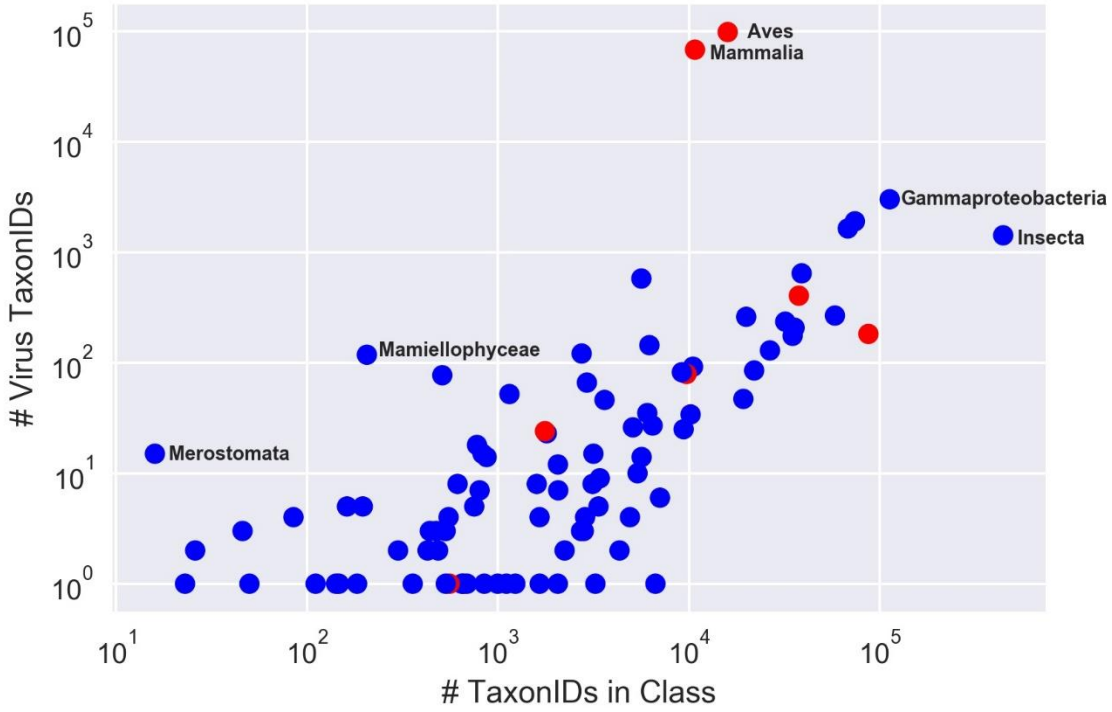

Figure 8: Scatter plot of Class Size against Viruses. The number of taxonIDs within a class was plotted against the number of virus taxonIDs assigned that class. Members of the phylum Chordata are shown in red. Notable outliers have been labelled.

3.4.4. The overrepresentation of Chordata is accounted for by high sequencing of human infectious viruses.

To determine the reason for the high number of viral taxa assigned to Chordates, the number of virus taxonIDs in the five viral genera with the most virus taxonIDs on the NCBI taxonomy database was plotted. More than 80% of all virus taxonIDs on the NCBI taxonomy database are accounted for by five genera, all of which can be pathogenic to humans. Influenza A, whose primary host is Aves, accounted for >55% of virus taxonIDs; whereas, Norovirus, Influenza B, HIV-1 and Sapovirus, which infect humans and other mammals, accounted for >20% of virus taxonIDs (Figure 8).

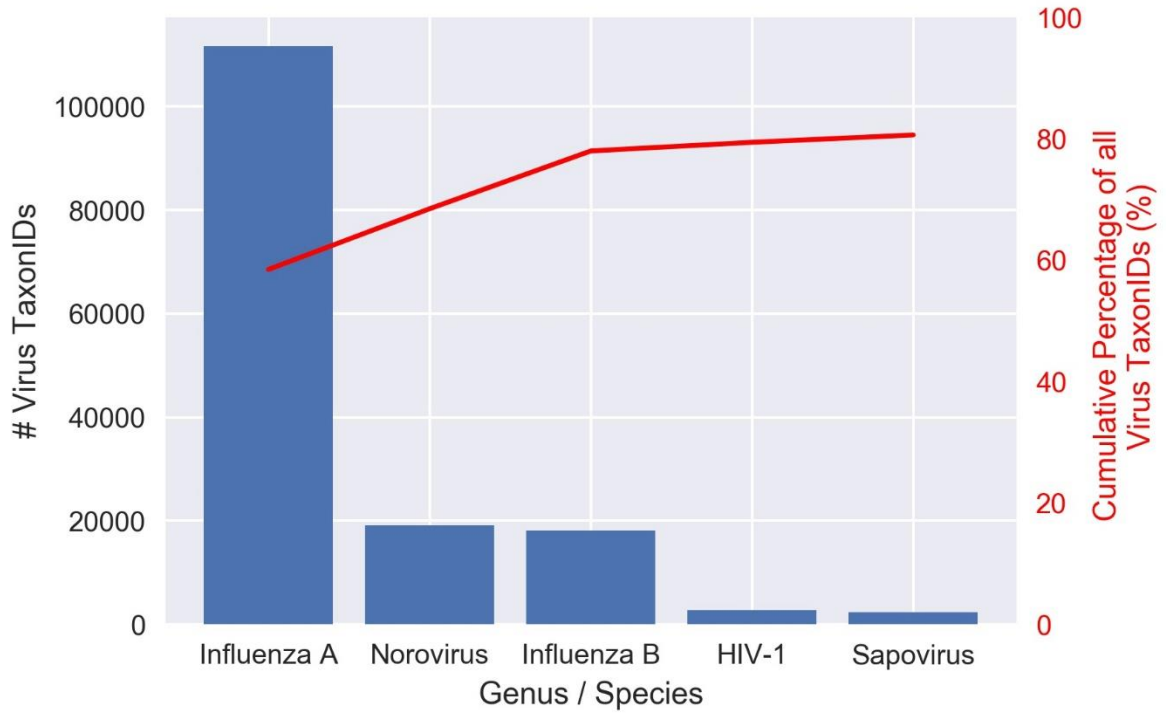

Figure 9: Viruses Infecting Chordates. The top-five virus genera or species that infect chordates organised by number of published genomes. The red line shows the % of all virus taxonIDs that these taxa account for.

3.4.5. In bacteriophages there is a bias toward medically relevant taxa.

To see whether a similar anthropocentric bias applies to viruses of bacteria, the number of published bacteriophages was plotted against host taxa. The sequencing bias for viruses of bacteria is less uneven than for all viruses. However, there is still a notable bias toward viruses of Proteobacteria, in particular the family Enterobacteriacea which contains pathogens such as members of the genera *Salmonella*, *Escherechia*, *Klebsiella* and *Shigella*; also highly represented are viruses of Firmicutes and Actinobacteria, phyla of gram-positive bacteria. Firmicutes includes the order Lactobacillales, members of which are involved in bio-industrial processes such as carbohydrate fermentation, as well as medically relevant taxa such as members of the families Streptococccaceae (*Streptococcus* spp.) and Bacillaceae (*Staphylocccus* spp.). Actinobacteria contains the Mycobateriaceae, members of which include the pathogens that cause leprosy (*Mycobacterium leprae*) and tuberculosis (*Mycobacterium tuberculosis*) (Figure 9).

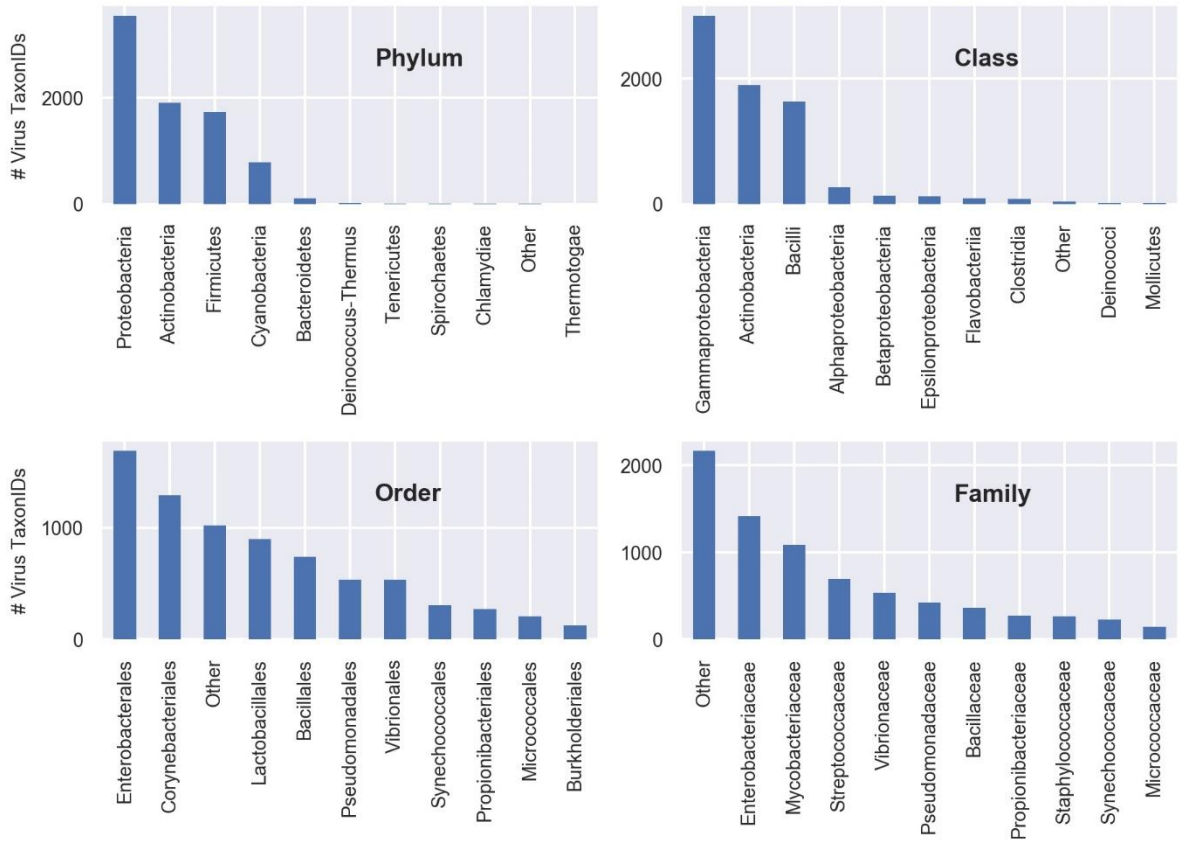

Figure 10: Hosts of Bacteriophages. Hosts of published bacteriophages classified to the ranks of phylum, class, order and family. The top-ten hosts for each rank are shown; "other" comprises all other hosts within that rank. Virus taxonIDs to which a rank couldn't be assigned are not shown.

3.5. Discussion

Despite the immense diversity of viruses, sequencing efforts are heavily skewed toward viruses infecting a few host taxa. More than 80% of virus taxonIDs on the NCBI taxonomy database belong to only five genera, while no viruses have been assigned to >80% of potential host phyla. This highlights how little of virus diversity has been explored.

Sequencing efforts for viruses have been largely focused on those which are known causative agents of human disease. Efforts to explore viral diversity more broadly have been modest, including potential zoonotic viruses from other species. This is important because a zoonotic epidemic will potentially be deadlier if caused be a completely unknown virus, as it may delay the development of an effective treatment and epidemiological strategy (Gardy, Loman & Rambaut, 2015).

In order to address the deficit in our understanding of potentially zoonotic viruses, in 2018 the Global Virome Project (GVP) was initiated(Carroll *et al.*, 2018a). The project has identified 25 viral families which infect mammals and birds, which include up to 827,000 zoonotic viruses. The project is focussed on sequencing most of these viruses in the next ten years and has assembled a wide range of stakeholders, including members of industry, academia and inter-governmental organisations in order to do so. Using current sequencing technology the projected cost is 7 billion USD, but it is anticipated that sequencing costs will continue to decline (Anon, n.d.).

Projects like the GVP may increase our knowledge of viruses of birds and mammals, but not of viruses in more diverse hosts. This is an issue because viruses play crucial roles from biogeochemical cycling in soils and oceans, to controlling microbial community composition in bioreactors such as sewage treatment systems, or of the microbial consortia in the guts of organisms from termites to elephants.

Global warming is predicted to dysregulate virus-host interactions that govern environmental biomes to an unprecedented degree (Danovaro *et al.*, 2011; Roberts *et al.*, 2018), which in turn could affect fundamental biogeochemical processes and the industrial use of microorganisms. In order to understand the impact that global warming and climate change will have on microbial communities, it is essential to understand the viruses infecting them.

An additional reason to isolate and sequence viruses from a broad range of hosts is the wealth of new genetic information the viruses will contain. The isolation of two bacteriophages,

lambda and T4, uncovered genetic tools that shaped molecular biology for a century (Delbrück, 1940; Lederberg & Lederberg, 1953). Given that for >90% of bacterial phyla there are no corresponding virus taxonIDs in NCBI taxonomy, there is a high potential to discover new viruses with biotechnological application.

Metagenomics is a field that is helping to diversify the hosts of sequenced viruses. The advent of viral metagenomic sequencing over a decade ago uncovered a wealth of virus genomes with no recognizable similarity to known sequences (Edwards & Rohwer, 2005; Angly *et al.*, 2006). Metagenomic sequencing is being used to explore microbial diversity at high resolution across the planet (Whon *et al.*, 2012; Mizuno *et al.*, 2016; Roux *et al.*, 2016). The results often uncover a plethora of new viruses infecting microbial life including protists (Flaviani *et al.*, 2017), cyanobacteria (Flores-Uribe *et al.*, 2018) and Bacteriodetes (Dutilh *et al.*, 2014).

However, environmental sequences still only contribute a small minority of the total virus taxonIDs on NCBI Taxonomy (unpublished data). Moreover, host information is often absent from metagenomically assembled virus genomes, and thus cannot be processed using VHost-Classifier. Metagenomics may provide many new virus sequences belonging to new species, but they are not named sufficiently to predict the hosts they infect. This trend is unlikely to change as the ICTV (International Committee on Taxonomy of Viruses) does not require that new taxa are named after their host (King *et al.*, 2018), to predict a host for newly sequenced viruses, analysis of the sequence submission metadata may be required.

It is important to consider several caveats when interpreting the results presented in this chapter. First, the analysis was done based on all the viruses with taxon IDs in the NCBI taxonomy database. This is not the same as a comprehensive analysis of all sequenced virus genomes, as not all virus genome submissions to GenBank are assigned a virus taxonID on NCBI. Additionally, some NCBI virus taxon IDs refer to taxonomic ranks not associated with a particular genome (e.g. virus subfamilies or genera), while other virus taxonIDs are associated with several genomes. However, analysing virus taxonIDs does give a robust indication of sequencing trend, as most published virus genomes are assigned a virus taxon ID.

The software was unable to assign a host to about 5% of viruses on NCBI Taxonomy, most of which were environmental sequences without host information in the name. We have not manually ascertained what hosts each of these viruses infect; if the majority don’t infect

Chordates, their exclusion may have exaggerated the reported sequencing bias. Another shortcoming of the software is that it assigned Influenza A viruses that do not have host information in the name to the class 'Aves'; many of these viruses may infect humans, as the Influenza WHO naming conventions only require host information to be present in the name of viruses that do not infect humans (Anonymous, 1980). Correcting for this in subsequent analysis would skew the host weighting further toward mammals and increase the coverage of virus taxonIDs in the database at the resolution of Order and Family.

In the analysis of virus taxonIDs, virus-host pairings were made by parsing words in the virus names. By assigning a single host based on the virus name, the potential for a virus to infect multiple hosts is ignored and it is assumed that the host included in the virus name is accurate. For viruses published without cultivation it is not possible to state that the host assignment is correct. Nonetheless, most virus genomes in NCBI are from viruses in culture, and most viruses are thought to be host specific; thus, the approach of assigning a virus to a single host taxon, is reasonably robust.

Recently NCBI Virus has released an experimental interface which connects GenBank virus accessions with host information; as of 10 April 2019 there were 200,217 complete virus genomes, a large proportion of which have been assigned a host (Brister *et al.*, 2015). Analysing the host distribution of viruses in this dataset would provide secondary validation of the trends described here.

The results of this study give quantitative support to the virus sequencing bias that has already been identified (Carroll *et al.*, 2018a). They should encourage further exploration of the Earth's virosphere in the example of initiatives such as the Tara Oceans Project and the Global Virome Project (Bork *et al.*, 2015; Carroll *et al.*, 2018a). This study also highlights the power of simple NLP algorithms when applied to biological metadata. Analysing biological metadata (i.e. data describing the contents of biological data) uncovers trends in the overall sequencing and publication of biological data that might be missed when our focus is on single studies. These trends may be used to infer socioeconomic factors that bias the way biological research is undertaken and to highlight research areas that are currently neglected. To date, few studies have made use of biological metadata, and it remains a powerful untapped resource.

References

Angly, F.E., Felts, B., Breitbart, M., Salamon, P., et al. (2006) The Marine Viromes of Four Oceanic Regions Nancy A Moran (ed.). *PLoS Biology*. [Online] 4 (11), e368. Available from: doi:10.1371/journal.pbio.0040368 [Accessed: 13 March 2019].

Anon (1980) A revision of the system of nomenclature for influenza viruses: a WHO memorandum. [Online] 58 (4), 585–591. Available from: http://www.ncbi.nlm.nih.gov/pubmed/6969132 [Accessed: 11 April 2019].

Anon (n.d.) *Bioinformatics.*

Anon (n.d.) *DNA Sequencing Costs: Data - National Human Genome Research Institute (NHGRI)*. [Online]. Available from: https://www.genome.gov/27541954/dna-sequencing-costs-data/ [Accessed: 3 January 2019b].

Berger, S.A., Krompass, D. & Stamatakis, A. (2011) Performance, Accuracy, and Web Server for Evolutionary Placement of Short Sequence Reads under Maximum Likelihood. *Systematic Biology*. [Online] 60 (3), 291–302. Available from: doi:10.1093/sysbio/syr010 [Accessed: 21 November 2018].

Bork, P., Bowler, C., de Vargas, C., Gorsky, G., et al. (2015) Tara Oceans. Tara Oceans studies plankton at planetary scale. Introduction. *Science (New York, N.Y.)*. [Online] 348 (6237), 873. Available from: doi:10.1126/science.aac5605 [Accessed: 13 March 2019].

Brister, J.R., Ako-adjei, D., Bao, Y. & Blinkova, O. (2015) NCBI Viral Genomes Resource. *Nucleic Acids Research*. [Online] 43 (D1), D571–D577. Available from: doi:10.1093/nar/gku1207 [Accessed: 11 April 2019].

Buchfink, B., Xie, C. & Huson, D.H. (2015) Fast and sensitive protein alignment using DIAMOND. *Nature Methods*. [Online] 12 (1), 59–60. Available from: doi:10.1038/nmeth.3176 [Accessed: 22 November 2018].

Carroll, D., Daszak, P., Wolfe, N.D., Gao, G.F., et al. (2018a) The Global Virome Project. *Science*. [Online] 359 (6378), 872 LP-874. Available from: doi:10.1126/science.aap7463.

Carroll, D., Watson, B., Togami, E., Daszak, P., et al. (2018b) Building a global atlas of zoonotic viruses. *Bulletin of the World Health Organization*. [Online] 96 (4), 292–294.

Available from: doi:10.2471/BLT.17.205005 [Accessed: 22 November 2018].

Casjens, S. (2003) Prophages and bacterial genomics: what have we learned so far? *Molecular Microbiology*. [Online] 49 (2), 277–300. Available from: doi:10.1046/j.1365-2958.2003.03580.x [Accessed: 13 November 2018].

Czech, L., Huerta-Cepas, J. & Stamatakis, A. (2017) A Critical Review on the Use of Support Values in Tree Viewers and Bioinformatics Toolkits. *Molecular biology and evolution*. [Online] 34 (6), 1535–1542. Available from: doi:10.1093/molbev/msx055 [Accessed: 17 November 2018].

Dale, R., Grüning, B., Sjödin, A., Rowe, J., et al. (2017) Bioconda: A sustainable and comprehensive software distribution for the life sciences. *bioRxiv*. [Online] 207092. Available from: doi:10.1101/207092 [Accessed: 11 April 2019].

Danovaro, R., Corinaldesi, C., Dell'Anno, A., Fuhrman, J.A., et al. (2011) Marine viruses and global climate change. *FEMS Microbiology Reviews*. [Online] 35 (6), 993–1034. Available from: doi:10.1111/j.1574-6976.2010.00258.x [Accessed: 21 December 2018].

Delbrück, M. (1940) THE GROWTH OF BACTERIOPHAGE AND LYSIS OF THE HOST. *The Journal of general physiology*. [Online] 23 (5), 643–660. Available from: http://www.ncbi.nlm.nih.gov/pubmed/19873180 [Accessed: 7 January 2019].

Dutilh, B.E., Cassman, N., McNair, K., Sanchez, S.E., et al. (2014) A highly abundant bacteriophage discovered in the unknown sequences of human faecal metagenomes. *Nature Communications*. [Online] 5 (1), 4498. Available from: doi:10.1038/ncomms5498 [Accessed: 3 January 2019].

Edwards, R.A. & Rohwer, F. (2005) Viral metagenomics. *Nature Reviews Microbiology*. [Online] 3 (6), 504–510. Available from: doi:10.1038/nrmicro1163 [Accessed: 13 November 2018].

Flaviani, F., Schroeder, D.C., Balestreri, C., Schroeder, J.L., et al. (2017) A Pelagic Microbiome (Viruses to Protists) from a Small Cup of Seawater. *Viruses*. [Online] 9 (3). Available from: doi:10.3390/v9030047 [Accessed: 3 January 2019].

Flores-Uribe, J., Philosof, A., Sharon, I. & Beja, O. (2018) A novel oceanic uncultured temperate cyanophage lineage. *bioRxiv*. [Online] 325100. Available from: doi:10.1101/325100 [Accessed: 3 January 2019].

Gardy, J., Loman, N.J. & Rambaut, A. (2015) Real-time digital pathogen surveillance — the time is now. *Genome Biology*. [Online] 16 (1), 155. Available from: doi:10.1186/s13059-015-0726-x [Accessed: 18 December 2018].

Garijo, D., Kinnings, S., Xie, L., Xie, L., et al. (2013) Quantifying reproducibility in computational biology: the case of the tuberculosis drugome. *PloS one*. [Online] 8 (11), e80278. Available from: doi:10.1371/journal.pone.0080278 [Accessed: 17 November 2018].

Gonçalves, R.S. & Musen, M.A. (2019) The variable quality of metadata about biological samples used in biomedical experiments. *Scientific Data*. [Online] 6, 190021. Available from: doi:10.1038/sdata.2019.21 [Accessed: 1 April 2019].

ten Hoopen, P., Finn, R.D., Bongo, L.A., Corre, E., et al. (2017) The metagenomic data life-cycle: standards and best practices. *GigaScience*. [Online] 6 (8), 1–11. Available from: doi:10.1093/gigascience/gix047 [Accessed: 20 November 2018].

Hucka, M., Nickerson, D.P., Bader, G.D., Bergmann, F.T., et al. (2015) Promoting Coordinated Development of Community-Based Information Standards for Modeling in Biology: The COMBINE Initiative. *Frontiers in bioengineering and biotechnology*. [Online] 3, 19. Available from: doi:10.3389/fbioe.2015.00019 [Accessed: 19 November 2018].

Hunt, M., Silva, N. De, Otto, T.D., Parkhill, J., et al. (2015) Circlator: automated circularization of genome assemblies using long sequencing reads. *Genome biology*. [Online] 16, 294. Available from: doi:10.1186/s13059-015-0849-0 [Accessed: 21 November 2018].

Huson, D.H., Auch, A.F., Qi, J. & Schuster, S.C. (2007) MEGAN analysis of metagenomic data. *Genome research*. [Online] 17 (3), 377–386. Available from: doi:10.1101/gr.5969107 [Accessed: 21 November 2018].

Kearse, M., Moir, R., Wilson, A., Stones-Havas, S., et al. (2012) Geneious Basic: An integrated and extendable desktop software platform for the organization and analysis of sequence data. *Bioinformatics*. [Online] 28 (12), 1647–1649. Available from: doi:10.1093/bioinformatics/bts199 [Accessed: 17 December 2018].

King, A.M.Q., Lefkowitz, E.J., Mushegian, A.R., Adams, M.J., et al. (2018) Changes to

taxonomy and the International Code of Virus Classification and Nomenclature ratified by the International Committee on Taxonomy of Viruses (2018). *Archives of Virology*. [Online] 163 (9), 2601–2631. Available from: doi:10.1007/s00705-018-3847-1 [Accessed: 11 April 2019].

Kitson, E. & Suttle, C.A. (2019) VHost-Classifier: Virus-Host Classification using natural language processing Jonathan Wren (ed.). *Bioinformatics*. [Online] Available from: doi:10.1093/bioinformatics/btz151 [Accessed: 3 March 2019].

Lederberg, E.M. & Lederberg, J. (1953) Genetic Studies of Lysogenicity in Escherichia Coli. *Genetics*. [Online] 38 (1), 51–64. Available from: http://www.ncbi.nlm.nih.gov/pubmed/17247421 [Accessed: 7 January 2019].

Leonard, S.A., Littlejohn, T.G. & Baxevanis, A.D. (2007) Common File Formats. In: *Current Protocols in Bioinformatics*. [Online]. Hoboken, NJ, USA, John Wiley & Sons, Inc. p. Appendix 1B. Available from: doi:10.1002/0471250953.bia01bs16 [Accessed: 19 November 2018].

Letunic, I. & Bork, P. (2007) Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display and annotation. *Bioinformatics*. [Online] 23 (1), 127–128. Available from: doi:10.1093/bioinformatics/btl529 [Accessed: 22 November 2018].

Matsen, F.A., Kodner, R.B. & Armbrust, E.V. (2010) pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics*. [Online] 11 (1), 538. Available from: doi:10.1186/1471-2105-11-538 [Accessed: 3 January 2019].

McGinnis, S. & Madden, T.L. (2004) BLAST: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research*. [Online] 32 (Web Server), W20–W25. Available from: doi:10.1093/nar/gkh435 [Accessed: 3 January 2019].

von Mering, C., Hugenholtz, P., Raes, J., Tringe, S.G., et al. (2007) Quantitative phylogenetic assessment of microbial communities in diverse environments. *Science (New York, N.Y.)*. [Online] 315 (5815), 1126–1130. Available from: doi:10.1126/science.1133420 [Accessed: 22 November 2018].

Mizuno, C.M., Ghai, R., Saghaï, A., López-García, P., et al. (2016) Genomes of Abundant and Widespread Viruses from the Deep Ocean. *mBio*. [Online] 7 (4). Available from:

doi:10.1128/mBio.00805-16 [Accessed: 3 January 2019].

Vitantonio Pantaleo & Michela Chiumenti (eds.) (2018) *Viral Metagenomics*. Methods in Molecular Biology. [Online]. New York, NY, Springer New York. Available from: doi:10.1007/978-1-4939-7683-6 [Accessed: 13 November 2018].

Perry, S.C. & Beiko, R.G. (2010) Distinguishing microbial genome fragments based on their composition: evolutionary and comparative genomic perspectives. *Genome biology and evolution*. [Online] 2, 117–131. Available from: doi:10.1093/gbe/evq004 [Accessed: 22 November 2018].

Phillippy, A.M., Schatz, M.C. & Pop, M. (2008) Genome assembly forensics: finding the elusive mis-assembly. *Genome biology*. [Online] 9 (3), R55. Available from: doi:10.1186/gb-2008-9-3-r55 [Accessed: 21 November 2018].

Ren, J., Ahlgren, N.A., Lu, Y.Y., Fuhrman, J.A., et al. (2017) VirFinder: a novel k-mer based tool for identifying viral sequences from assembled metagenomic data. *Microbiome*. [Online] 5 (1), 69. Available from: doi:10.1186/s40168-017-0283-5 [Accessed: 3 January 2019].

Roberts, K.E., Hadfield, J.D., Sharma, M.D. & Longdon, B. (2018) Changes in temperature alter the potential outcomes of virus host shifts David S. Schneider (ed.). *PLOS Pathogens*. [Online] 14 (10), e1007185. Available from: doi:10.1371/journal.ppat.1007185 [Accessed: 3 January 2019].

Rodriguez, N., Pettit, J.-B., Dalle Pezze, P., Li, L., et al. (2016) The systems biology format converter. *BMC bioinformatics*. [Online] 17, 154. Available from: doi:10.1186/s12859-016-1000-2 [Accessed: 19 November 2018].

Rose, R., Constantinides, B., Tapinos, A., Robertson, D.L., et al. (2016) Challenges in the analysis of viral metagenomes. *Virus evolution*. [Online] 2 (2), vew022. Available from: doi:10.1093/ve/vew022 [Accessed: 3 January 2019].

Roux, S., Brum, J.R., Dutilh, B.E., Sunagawa, S., et al. (2016) Ecogenomics and potential biogeochemical impacts of globally abundant ocean viruses. *Nature*. [Online] 537 (7622), 689–693. Available from: doi:10.1038/nature19366 [Accessed: 13 March 2019].

Schulz, F., Alteio, L., Goudeau, D., Ryan, E.M., et al. (2018) Hidden diversity of soil giant viruses. *Nature Communications*. [Online] 9 (1), 4881. Available from:

doi:10.1038/s41467-018-07335-2 [Accessed: 23 November 2018].

Sczyrba, A., Hofmann, P., Belmann, P., Koslicki, D., et al. (2017) Critical Assessment of Metagenome Interpretation—a benchmark of metagenomics software. *Nature Methods*. [Online] 14 (11), 1063–1071. Available from: doi:10.1038/nmeth.4458 [Accessed: 21 November 2018].

STERK, P., HIRSCHMAN, L., FIELD, D. & WOOLEY, J. (2009) GENOMIC STANDARDS CONSORTIUM WORKSHOP: In: *Biocomputing 2010*. [Online]. WORLD SCIENTIFIC. pp. 481–484. Available from: doi:10.1142/9789814295291_0050 [Accessed: 20 November 2018].

Suttle, C.A. (2007) Marine viruses — major players in the global ecosystem. *Nature Reviews Microbiology*. [Online] 5 (10), 801–812. Available from: doi:10.1038/nrmicro1750 [Accessed: 11 March 2019].

Thurber, R. V, Haynes, M., Breitbart, M., Wegley, L., et al. (2009) Laboratory procedures to generate viral metagenomes. *Nature Protocols*. [Online] 4 (4), 470–483. Available from: doi:10.1038/nprot.2009.10 [Accessed: 20 November 2018].

Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., et al. (2015) The impact of Docker containers on the performance of genomic pipelines. *PeerJ*. [Online] 3, e1273. Available from: doi:10.7717/peerj.1273 [Accessed: 17 November 2018].

Turnbaugh, P.J., Ley, R.E., Hamady, M., Fraser-Liggett, C.M., et al. (2007) The human microbiome project. *Nature*. [Online] 449 (7164), 804–810. Available from: doi:10.1038/nature06244 [Accessed: 8 January 2019].

Uyaguari-Diaz, M.I., Chan, M., Chaban, B.L., Croxen, M.A., et al. (2016) A comprehensive method for amplicon-based and metagenomic characterization of viruses, bacteria, and eukaryotes in freshwater samples. *Microbiome*. [Online] 4 (1), 20. Available from: doi:10.1186/s40168-016-0166-1 [Accessed: 21 November 2018].

Whon, T.W., Kim, M.-S., Roh, S.W., Shin, N.-R., et al. (2012) Metagenomic Characterization of Airborne Viral DNA Diversity in the Near-Surface Atmosphere. *Journal of Virology*. [Online] 86 (15), 8221–8231. Available from: doi:10.1128/JVI.00293-12 [Accessed: 3 January 2019].

Yolken, R.H., Jones-Brando, L., Dunigan, D.D., Kannan, G., et al. (2014) Chlorovirus

ATCV-1 is part of the human oropharyngeal virome and is associated with changes in cognitive functions in humans and mice. *Proceedings of the National Academy of Sciences of the United States of America*. [Online] 111 (45), 16106–16111. Available from: doi:10.1073/pnas.1418895111 [Accessed: 22 November 2018].

Zhang, Y. & Sun, Y. (2011) HMM-FRAME: accurate protein domain classification for metagenomic sequences containing frameshift errors. *BMC Bioinformatics*. [Online] 12 (1), 198. Available from: doi:10.1186/1471-2105-12-198 [Accessed: 22 November 2018].

Zhong, X. & Jacquet, S. (2014) Contrasting diversity of phycodnavirus signature genes in two large and deep western European lakes. *Environmental Microbiology*. [Online] 16 (3), 759–773. Available from: doi:10.1111/1462-2920.12201 [Accessed: 23 November 2018].

CODE APPENDIX

The code below represents the most up to date software versions for each of the computational tools described in Chapter 2.

C1: VHost-Classifier

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 27 17:20:30 2018

@author: Ezra
"""
#!usr/bin/env python 3

#################################################################################################
#FUNCTIONS
#################################################################################################

def tax_groups (choice):

    group_one = []
    group_two = []
    group_three = []
   #make groups Phylum class order or phylum order family
    with open ('Phyla_in_NCBI.txt') as phyla, open('Order_in_NCBI.txt') as Orders,
open('Class_in_NCBI.txt') as Classs, open('Families_in_NCBI.txt') as Families:
        pat = csv.reader(phyla)
        cat = csv.reader(Classs)
        oat = csv.reader(Orders)
        fat = csv.reader(Families)
        if choice == 'PCO':
            for rowe in pat:
                row = rowe[0]
                group_one.append(row)
            for rowe in cat:
                row = rowe[0]
                group_two.append(row)
            for rowe in oat:
                row = rowe[0]
                group three.append(row)
        if choice == 'POF':
            for rowe in pat:
                row = rowe[0]
                group_one.append(row)
            for rowe in oat:
                row = rowe[0]
                group_two.append(row)
            for rowe in fat:
                row = rowe[0]
                group_three.append(row)

    #get rid of empty elements
    group_one = list(filter(None, group_one))
    group_one.append('NA')
    group_two = list(filter(None, group_two))
    group_two.append('NA')
    group_three = list(filter(None, group_three))
    group_three.append('NA')

    #sort out some discrepancy between databases
    if choice == 'PCO':
        group_three.append('Cetartiodactyla')
    if choice == 'POF':
        group_two.append('Cetartiodactyla')


    g1 = {}
    keys = range(len(group_one))
    values = group_one
```

```
    for i in keys:
            g1[values[i]] = [i]

    g2 = {}
    keys = range(len(group_two))
    values = group_two
    for i in keys:
            g2[values[i]] = [i]

    g3 = {}
    keys = range(len(group_three))
    values = group_three
    for i in keys:
            g3[values[i]] = [i]

    return g1,g2,g3,group_one,group_two,group_three



def host_locate(taxonID,genera):
    #don't worry about case
    rhost = 'NA'
    host = 'NA'
    #name_change- the direct viral prefix has a latin root so will misclassify. However it can
be useful to assign host.
    #If we have used it to assign host name_change = true and therefore don't delete it later
in code.
    #Ignore RHOST if we are looking at phage
    name_change = False
    taxonID = taxonID.lower()
    if 'virus' in taxonID:
        host = taxonID.split('virus')[0]
        rhost = taxonID.split('virus')[1]
    if 'phage' in taxonID:
        host = taxonID.split('phage')[0]
    if 'viridae' in taxonID:
        host = taxonID.split('viridae')[0]
        rhost = taxonID.split('viridae')[1]
    if 'viroid' in taxonID:
        host = taxonID.split('viroid')[0]
        rhost = taxonID.split('viroid')[1]
    if 'virinae' in taxonID:
        host = taxonID.split('virinae')[0]
        rhost = taxonID.split('virinae')[1]
    if 'satellite' in taxonID:
        host = taxonID.split('satellite')[0]
        rhost = taxonID.split('satellite')[1]
    if 'virales' in taxonID:
        host = taxonID.split('virales')[0]
        rhost = taxonID.split('virales')[1]
    if 'hiv-1' in taxonID or 'hiv-2' in taxonID:
        host = taxonID
    lhost = host.split(' ')
    rlhost = rhost.split(' ')
    ref = len(lhost)-1
    lhost = lhost+rlhost
    ##associated means it is not an isolate
    ##we ignore this - ver 13
    ##Common names are often used for common animals in place of genus species, this needs to
be corrected
    ###Might be best to turn this into a .csv file and read through it?
    count = range(0,len(lhost))
    for c,w in zip(count,lhost):
            #if there is a genus name, make the next word genus+species providing a) there is
a next word b) the next word isnt a comname or a sci name
            change = True
            for g in genera:
                if w == g:
                    try:
                        if lhost[c+1] == '':
                            change = False
                            break
                        if change == True:
                            for nam,sci in zip(comname,sciname):
                                if lhost[c+1] == nam or lhost[c+1] == sci:
                                    change = False
                                    break
```

```
                        if change == True:
                            lhost[c+1] = lhost[c]+' '+lhost[c+1]
                            break
                        if change == False:
                            break
                    except:
                        break

            ##convert common names to their scientific names so they can be used to look up
NCBI database
            for nam,sci in zip(comname,sciname):
                if nam == w:
                    lhost[c] = sci
                    ##if we are dealing with influenza, find the strain - check it isn't
illegitimate
                    if w == 'influenza':

                        try:
                            lhost[c+4] = lhost[c+4].split('/')[1]
                            for ifs in infstrains:
                                    ifstrain = ifs[0]
                                    if lhost[c+4] == ifstrain:
                                        lhost[c+4] = 'na'
                                        break
                        except:
                            pass
                        if lhost[c+1] == 'a':
                            lhost[c] = 'aves'

                        elif lhost[c+1] == 'b' or lhost[c+1] == 'c':
                            lhost[c] = 'mammalia'

                        else:
                            lhost[c] = 'mammalia'

                        #move the influenza to the first position - spiny eel influenza virus
- pop deletes and returns element
                        lhost.insert(0,lhost.pop(c))

                    #if we are dealing with norovirus
                    if w =='noro':
                     try:
                        lhost[c+2] = lhost[c+2].split('/')[0]
                     except:
                         pass
                    ##if we have changed the virus prefix
                    if lhost[c] == lhost[ref]:
                        name_change = True
                        lhost.insert(0,lhost.pop(c))
                    break

    if name_change == False:
        lhost.remove(lhost[ref])
    ref = len(lhost)-1
    if ref > -1:
        while ref > -1:
            name2taxonID = ncbi.get_name_translator([lhost[ref]])
            if name2taxonID:
                break
            else:
                ref = ref-1
    else:
        name2taxonID = {}
    ##include the IMGER database

    return name2taxonID


def env_groups (iecos,iecoc,iecot,iecost):
    n = 0
    e = 0
    f = 0
    r = 0
    ecosys = ['']*1000
    ecocat = ['']*1000
    ecotyp = ['']*1000
    ecosub = ['']*1000
```

```
    ecos = iecos[1:len(iecos)]
    ecoc = iecoc[1:len(iecoc)]
    ecot = iecot[1:len(iecot)]
    ecost = iecost[1:len(iecost)]



    for item in ecos:
        if item not in ecosys:
                ecosys[n]=item
                n = n + 1
    for item in ecoc:
        if item not in ecocat:
                ecocat[e]=item
                e = e + 1
    for item in ecot:
        if item not in ecotyp:
                ecotyp[f]=item
                f = f+ 1
    for item in ecost:
        if item not in ecosub:
                ecosub[r]=item
                r = r+ 1


    #get rid of empty elements
    ecosys = list(filter(None, ecosys))
    ecosys.append('NA')
    ecocat = list(filter(None, ecocat))
    ecocat.append('NA')
    ecocat.append('Bioreactor')
    ecotyp = list(filter(None, ecotyp))
    ecotyp.append('NA')
    ecosub = list(filter(None, ecosub))
    ecosub.append('NA')



    e1 = {}
    keys = range(len(ecosys))
    values = ecosys
    for i in keys:
            e1[values[i]] = [i]
    e2 = {}
    keys = range(len(ecocat))
    values = ecocat
    for i in keys:
            e2[values[i]] = [i]
    e3 = {}
    keys = range(len(ecotyp))
    values = ecotyp
    for i in keys:
            e3[values[i]] = [i]
    e4 = {}
    keys = range(len(ecosub))
    values = ecosub
    for i in keys:
            e4[values[i]] = [i]


    return e1,e2,e3,e4,ecosys,ecocat,ecotyp,ecosub


def environ_locate(name):
    name = name.lower()
    words = name.split(" ")
    ecoc = 'NA'
    ecos = 'NA'
    ecot = 'NA'
    ecost = 'NA'
    fd = False
    for w in words:
            if fd == True:
                break
            for nam,wa,wb,wc,wd in zip(dw,decoc,decos,decot,decost):
                if nam == w:
                    ecos = wa
```

```
                    ecoc = wb
                    ecot = wc
                    ecost = wd
                    fd = True
                    break
    return ecoc,ecos,ecot,ecost


################################################################################
##MAIN SCRIPT
################################################################################


#Initial filter of all the data for viruses
##The non-indent open and close here is a nice way to deal with multiple files that need to be
read/written simulataneously
# If a taxon ID doesn't have a kingdom assignment - it won't have a name
#newline = '' used to make each entry go to row directly below

################################################################################
#INITALISE DATABASES
################################################################################
print('Initialising databases....')

import argparse
parser = argparse.ArgumentParser()
parser.add_argument("Taxon_IDs", help="List of Taxon IDs to classify")
parser.add_argument("vhost_db", help ="Name of vhost file")
parser.add_argument("output_dir", help="Name of output directory")

parser.add_argument("-i","--index",type=int, help="value to index search terms from [default
0]")
parser.add_argument("-g","--groups",type=str,help="taxonomic groups to bin to [default PCO]")
parser.add_argument("-n","--names",type=str,help="parse file containing scientific names")

args = parser.parse_args()

if args.index:
   print ("Indexing search terms from:")
   print(str(args.index))

if args.names:
    print("parsing names file")

if args.groups == 'PCO':
    print('Classifying virus host to Phylum Class Order')
    choice = args.groups

elif args.groups == 'POF':
    print('Classifying virus host to Phylum Order Family')
    choice = args.groups

else:
    print('Classifying virus host to Phylum Class Order')
    choice = 'PCO'


import os
from ete3 import NCBITaxa
ncbi = NCBITaxa()
import csv

print('Filtering for viruses...')

##open com_sci
comname = []
sciname = []
cs2 = open('Common_to_Sci.csv')
comsci = csv.reader(cs2)
for row in comsci:
    comname.append(row[0])
    sciname.append(row[1])

##open word to env
dw = []
decoc = []
decos = []
```

```
decot = []
decost = []
wh = open('Word_to_Env.csv')
woho =csv.reader(wh)
for row in woho:
    dw.append(row[0])
    decoc.append(row[1])
    decos.append(row[2])
    decot.append(row[3])
    decost.append(row[4])


##create list of illegal influenza names
infstrains = []
with open('infstrains.txt') as ifs:
    infs = csv.reader(ifs)
    for row in infs:
        infstrains.append(row)


##initialise imger database
z1 = open('IMGER.csv', 'r')
imger = csv.reader(z1)

itaxonids = []
incbiids = []
ihost = []
iecos = []
iecoc = []
iecot = []
iecost = []


for rows in imger:
    incbiids.append(rows[7])
    ihost.append(rows[13])
    iecos.append(rows[8])
    iecoc.append(rows[9])
    iecost.append(rows[10])
    iecot.append(rows[11])

f2 = open(args.vhost_db, 'r')
c2 = csv.reader(f2,delimiter='\t')
vhostdb = list(c2)

##create taxonomic groups
(g1,g2,g3,group_one,group_two,group_three) = tax_groups(choice)

##create environmental groups
(e1,e2,e3,e4,ecosys,ecocat,ecotyp,ecosub) = env_groups(iecos,iecoc,iecot,iecost)


##need to remove the '/' character from the group list (this prevents the .csv file writing)
([s.strip('/') for s in group_one]) # remove the / from the string borders
group_one=([s.replace('/', '') for s in group_one]) # remove all the /s
([s.strip('/') for s in group_two]) # remove the 8 from the string borders
group_two=([s.replace('/', '') for s in group_two]) # remove all the /s
([s.strip('/') for s in group_three]) # remove the / from the string borders
group_three=([s.replace('/', '') for s in group_three]) # remove all the /s

#create genus list - used in binning
genera = []
import csv
with open ('Genus_in_NCBI.txt') as genus:
        sat = csv.reader(genus)
        for rowe in sat:
            row = rowe[0].lower()
            genera.append(row)


#############################################################################
##BEGIN HOST ASSIGNMENT
#############################################################################
print('Classifying hosts')
os.makedirs(args.output_dir,exist_ok=True)
e1 = open(args.Taxon_IDs, 'r')
```

```
##set this n to 0 to index from 0
n = 0
if args.index:
    n = args.index

os.chdir(args.output_dir)
with open('Virus.csv','w',newline='') as e3, open('Non-Virus.csv','w',newline='') as e4:
    d1 = csv.reader(e1)
    d3 = csv.writer(e3)
    d4 = csv.writer(e4)
    for row in d1:
                taxonID2name = ncbi.get_taxonID_translator([row[0]])
                ##deal with taxonids in ncbi but not in the database
                try:
                    SN = (list(taxonID2name.values())[0])

               ##added below to support viruses not in NCBI
                except IndexError:
                    if args.names:
                        os.chdir('..')
                        with open(args.names) as r1:
                            #row.append(SN)
                            t1 = csv.reader(r1)
                            SN=[row for idx, row in enumerate(t1) if idx ==n]
                            SN = SN[0][0]
                            os.chdir(args.output_dir)
                    else:
                        SN = 'Not in database'

                SN = SN.lower()
                row.append(str(n))
                row.append(SN)
                if 'virus' in SN:
                    d3.writerow(row)
                elif 'phage' in SN and not 'phagedenis' in SN:
                    d3.writerow(row)
                elif 'viridae' in SN:
                    d3.writerow(row)
                elif 'satellite' in SN:
                    d3.writerow(row)
                elif 'virinae' in SN:
                    d3.writerow(row)
                elif 'viroid' in SN:
                    d3.writerow(row)
                elif 'hiv-1' in SN or 'hiv-2' in SN:
                    d3.writerow(row)
                elif 'virales' in SN:
                    d3.writerow(row)
                else:
                    d4.writerow(row)
                n = n+1

Lifeform = ['Virus','NonVirus']
num_lines = ['Blank']*2
num_lines[0] = sum(1 for line in open('Virus.csv'))
num_lines[1] = sum(1 for line in open('Non-Virus.csv'))
with open ('Counts.csv','w') as counts:
     for i in zip(Lifeform,num_lines):
            co = csv.writer(counts)
            co.writerow(i)
e1.close()

os.makedirs('Virus',exist_ok=True)

f1 = open('Virus.csv', 'r')
os.chdir('Virus')
os.makedirs('Host Assigned',exist_ok=True)
os.chdir('Host Assigned')
f3 = open('Eukaryota.csv', 'w',newline='')
f4 = open('Bacteria.csv','w',newline='')
f5 = open('Virus.csv','w',newline='')
f6 = open('Archaea.csv','w',newline='')

c1 = csv.reader(f1)
c3 = csv.writer(f3)
c4 = csv.writer(f4)
c5 = csv.writer(f5)
```

```
c6 = csv.writer(f6)


icount = range(0,len(incbiids))
f9 = open('NAf.csv','w',newline='')
c9 = csv.writer(f9)

for row in c1:
    qtaxonID = row[0]
    qname = row[2]
    found = False
    ##First parse the vhostdb
    for master_row in vhostdb:
        dbtaxonID = master_row[0]
        host_name = master_row[8]
        if dbtaxonID == qtaxonID:
           #hn = [host_name.split(' ')[0]]
           name2taxonID = ncbi.get_name_translator([host_name])
           if 'root' in name2taxonID or 'bacteria' in name2taxonID:
               name2taxonID = []
           if not name2taxonID:
               pass
           else:
               row.append('VHostDB')
               found = True
           break

    if found == False:
        ##if not present try to predict the host
        name2taxonID = host_locate(qname,genera)


    try:
         lid = (list(name2taxonID.values())[0])
    except IndexError:
            c9.writerow(row)

    else:
        lineage = ncbi.get_lineage(lid[0])
        names = ncbi.get_taxonID_translator(lineage)
        lineage_trace = ([names[taxonID] for taxonID in lineage])
        ##deal with things not in NCBI
        while len(lineage_trace) < 3:
            lineage_trace.append('NA')

        gr1 = 'NA'
        for i in range(0,len(lineage_trace)):
                if lineage_trace[i] in group_one:
                    gr1 = lineage_trace[i]
        gr2 = 'NA'
        for i in range(0,len(lineage_trace)):
                if lineage_trace[i] in group_two:
                    gr2 = lineage_trace[i]
        gr3 = 'NA'
        for i in range(0,len(lineage_trace)):
                if lineage_trace[i] in group_three:
                    gr3 = lineage_trace[i]

        if lineage_trace[2] == 'Eukaryota':
            c3.writerow(row)
            os.makedirs('Eukaryote',exist_ok=True)
            os.chdir('Eukaryote')
            key = g1[gr1]
            family = group_one[key[0]]
            d8 = open(family+'.csv','a',newline='')
            pr = csv.writer(d8)
            pr.writerow(row)
            d8.close()
            ##bin group 2 entries
            os.makedirs(gr1,exist_ok=True)
            os.chdir(gr1)
            key = g2[gr2]
            family = group_two[key[0]]
            ## 'a' to append to file
            d8 = open(family+'.csv','a',newline='')
            pr = csv.writer(d8)
            pr.writerow(row)
```

```
        d8.close()
        ####BIN GROUP 3
        os.makedirs(gr2,exist_ok=True)
        os.chdir(gr2)
        key = g3[gr3]
        family = group_three[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        os.chdir('../../..')

    elif lineage_trace[2] == 'Bacteria':
        c4.writerow(row)
        os.makedirs('Bacteria',exist_ok=True)
        os.chdir('Bacteria')
        key = g1[gr1]
        family = group_one[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        ####BIN GROUP 2 BACTERIA
        os.makedirs(family,exist_ok=True)
        os.chdir(family)
        key = g2[gr2]
        family = group_two[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        ####BIN GROUP 3 EBACTERIA
        os.makedirs(family,exist_ok=True)
        os.chdir(family)
        key = g3[gr3]
        family = group_three[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        os.chdir('../../..')

    elif lineage_trace[2] == 'Viruses':
        c5.writerow(row)
        #bin group 1 entries
        os.makedirs('Virus',exist_ok=True)
        os.chdir('Virus')
        key = g1[gr1]
        family = group_one[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        ####BIN GROUP 2 VIRUS
        os.makedirs(gr1,exist_ok=True)
        os.chdir(gr1)
        key = g2[gr2]
        family = group_two[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        ####BIN GROUP 3 VIRUS
        os.makedirs(gr2,exist_ok=True)
        os.chdir(gr2)
        key = g3[gr3]
        family = group_three[key[0]]
        d8 = open(family+'.csv','a',newline='')
        pr = csv.writer(d8)
        pr.writerow(row)
        d8.close()
        os.chdir('../../..')

    elif lineage_trace[2] == 'Archaea':
        c6.writerow(row)
        os.makedirs('Archaea',exist_ok=True)
        os.chdir('Archaea')
```

```
            key = g1[gr1]
            family = group_one[key[0]]
            d8 = open(family+'.csv','a',newline='')
            pr = csv.writer(d8)
            pr.writerow(row)
            d8.close()
            ####BIN GROUP 2 aRCHEA
            os.makedirs(gr1,exist_ok=True)
            os.chdir(gr1)
            key = g2[gr2]
            family = group_two[key[0]]
            d8 = open(family+'.csv','a',newline='')
            pr = csv.writer(d8)
            pr.writerow(row)
            d8.close()
            ####BIN GROUP 3 ARCHEA
            os.makedirs(gr2,exist_ok=True)
            os.chdir(gr2)
            key = g3[gr3]
            family = group_three[key[0]]
            d8 = open(family+'.csv','a',newline='')
            pr = csv.writer(d8)
            os.chdir('../../..')


            pr.writerow(row)
            d8.close()
        else:
                c9.writerow(row)


cs2.close()
f3.close()
f4.close()
f5.close()
f6.close()
f9.close()


################################################################################
#Begin Host_Counts
################################################################################
print('Counting host assignments....')
super_kingdoms = ['Eukaryota','Bacteria','Virus','Archaea','NA']

num_lines = ['Blank']*5
num_lines[0] = sum(1 for line in open('Eukaryota.csv'))
num_lines[1] = sum(1 for line in open('Bacteria.csv'))
num_lines[2] = sum(1 for line in open('Virus.csv'))
num_lines[3] = sum(1 for line in open('Archaea.csv'))
num_lines[4] = sum(1 for line in open('NAf.csv'))

with open ('Counts.csv','w') as counts:
     for i in zip(super_kingdoms,num_lines):
            co = csv.writer(counts)
            co.writerow(i)


##This will go through and do all the rest of counting

for fname in os.listdir():
    path = fname
    if os.path.isdir(path):
        os.chdir(path)
        for g in group_one:
            try:
                d8 = open('COUNTS.csv','a',newline='')
                pr = csv.writer(d8)
                num_lines = sum(1 for line in open(g+'.csv'))
                i = [g]+ [str(num_lines)]
                pr.writerow(i)
                d8.close()
                ###group 2 counts
                os.chdir(g)

                for g2 in group_two:
                    try:
```

```
                            d8 = open('COUNTS.csv','a',newline='')
                            pr = csv.writer(d8)
                            num_lines = sum(1 for line in open(g2+'.csv'))
                            i = [g2]+ [str(num_lines)]
                            pr.writerow(i)
                            d8.close()
                            os.chdir(g2)

                            for g3 in group_three:
                                try:
                                    d8 = open('COUNTS.csv','a',newline='')
                                    pr = csv.writer(d8)
                                    num_lines = sum(1 for line in open(g3+'.csv'))
                                    i = [g3]+ [str(num_lines)]
                                    pr.writerow(i)
                                    d8.close()
                                except FileNotFoundError:
                                        pass
                            os.chdir('..')
                        except FileNotFoundError:
                            pass
                    os.chdir('..')
                except FileNotFoundError:
                        pass

        os.chdir('..')

#############################################################################################
#
###BIN UNASSIGNED HOSTS
#############################################################################################
#
print('Classifying unassigned viruses....')
z2 = open('NAf.csv','r')
sourceids = csv.reader(z2)

os.chdir('..')
#VSource-Classifier
#Take all the unassigned hits and run through IMG ER database to assign to environment.
#Create a dictionary function like for the vhost-db to define the groups


os.makedirs('Host Unassigned')
os.chdir('Host Unassigned')


count = range(0,len(incbiids))

for results_row in sourceids:
    tids = results_row[0]
    name = results_row[2]
    found = False
    ##first use the imger database
    for c,incbi in zip(count,incbiids):
        if tids == incbi:
             found = True
             ecos = iecos[c]
             ecoc = iecoc[c]
             ecot = iecot[c]
             ecost = iecost[c]
             if not ecos:
                ecos = 'NA'
             if not ecoc:
                ecoc = 'NA'
             if not ecot:
                ecot = 'NA'
             if not ecost:
                ecost = 'NA'
             break
    ##if that doesn't work predict the environment

    if found == False:
        (ecoc,ecos,ecot,ecost) = environ_locate(name)


    if ecos == 'Host-associated':
                            d8 = open(ecos+'.csv','a',newline='')
```

```
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ##BIN GROUP 1
                                os.makedirs('Host-associated',exist_ok=True)
                                os.chdir('Host-associated')
                                d8 = open(ecoc+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 2
                                os.makedirs(ecoc,exist_ok=True)
                                os.chdir(ecoc)
                                d8 = open(ecot+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 3
                                os.makedirs(ecot,exist_ok=True)
                                os.chdir(ecot)
                                d8 = open(ecost+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                os.chdir('../../..')
                                #break
        if ecos == 'Environmental':
                                d8 = open(ecos+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ##BIN GROUP 1
                                os.makedirs('Environmental',exist_ok=True)
                                os.chdir('Environmental')
                                d8 = open(ecoc+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 2
                                os.makedirs(ecoc,exist_ok=True)
                                os.chdir(ecoc)
                                d8 = open(ecot+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 3
                                os.makedirs(ecot,exist_ok=True)
                                os.chdir(ecot)
                                d8 = open(ecost+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                os.chdir('../../..')
                                #break

        if ecos == 'Engineered':
                                d8 = open(ecos+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ##BIN GROUP 1
                                os.makedirs('Engineered',exist_ok=True)
                                os.chdir('Engineered')
                                d8 = open(ecoc+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 2
                                os.makedirs(ecoc,exist_ok=True)
                                os.chdir(ecoc)
                                d8 = open(ecot+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 3
                                os.makedirs(ecot,exist_ok=True)
                                os.chdir(ecot)
                                d8 = open(ecost+'.csv','a',newline='')
                                pr = csv.writer(d8)
```

```
                                pr.writerow(results_row)
                                os.chdir('../../..')
                                #break
    if ecos == 'NA':
                                d8 = open(ecos+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ##BIN GROUP 1
                                os.makedirs('NA',exist_ok=True)
                                os.chdir('NA')
                                d8 = open(ecoc+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 2
                                os.makedirs(ecoc,exist_ok=True)
                                os.chdir(ecoc)
                                d8 = open(ecot+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                d8.close()
                                ####BIN GROUP 3
                                os.makedirs(ecot,exist_ok=True)
                                os.chdir(ecot)
                                d8 = open(ecost+'.csv','a',newline='')
                                pr = csv.writer(d8)
                                pr.writerow(results_row)
                                os.chdir('../../..')
                                #break


z1.close()
z2.close()

################################################
## BEGIN ENVIRONMENTAL COUNTS
################################################
print('Counting environmental groups....')
for g in ecosys:
            try:
                d8 = open('COUNTS1.csv','a',newline='')
                pr = csv.writer(d8)
                num_lines = sum(1 for line in open(g+'.csv'))
                i = [g]+ [str(num_lines)]
                pr.writerow(i)
                d8.close()
                os.chdir(g)
                for g2 in ecocat:
                    try:
                            d8 = open('COUNTS2.csv','a',newline='')
                            pr = csv.writer(d8)
                            num_lines = sum(1 for line in open(g2+'.csv'))
                            i = [g2]+ [str(num_lines)]
                            pr.writerow(i)
                            d8.close()
                            os.chdir(g2)
                            for g3 in ecotyp:
                                try:
                                    d8 = open('COUNTS3.csv','a',newline='')
                                    pr = csv.writer(d8)
                                    num_lines = sum(1 for line in open(g3+'.csv'))
                                    i = [g3]+ [str(num_lines)]
                                    pr.writerow(i)
                                    d8.close()
                                    os.chdir(g3)
                                    for g4 in ecosub:
                                        try:
                                            d8 = open('COUNTS4.csv','a',newline='')
                                            pr = csv.writer(d8)
                                            num_lines = sum(1 for line in open(g4+'.csv'))
                                            i = [g4]+ [str(num_lines)]
                                            pr.writerow(i)
                                            d8.close()
                                        except FileNotFoundError:
                                            pass
                                    os.chdir('..')
```

```
                                except FileNotFoundError:
                                    pass
                            os.chdir('..')
                        except FileNotFoundError:
                            pass
                    os.chdir('..')
                except FileNotFoundError:
                    pass


os.chdir('..')

#Finally do the host-assigned/unassigned counts (ugly code)

d8 = open('COUNTS.csv','a',newline='')
pr = csv.writer(d8)
os.chdir('Host Assigned')
bum_lines = sum(1 for line in open('NAf'+'.csv'))
i = ['Host Unassigned']+ [str(bum_lines)]
os.chdir('..')
pr.writerow(i)
d8.close()
d8 = open('COUNTS.csv','a',newline='')
pr = csv.writer(d8)
os.chdir('..')
num_lines = sum(1 for line in open('Virus'+'.csv'))
os.chdir('Virus')
i = ['Host Assigned']+ [str(num_lines-bum_lines)]
pr.writerow(i)
d8.close()


f1.close()
f2.close()
wh.close()
```

C2: Optimal-Translate

```
#!usr/bin/env python3

def transcribe(sequence,rev): #turns sequence into a list, cycles through letters and produces
reverse complement, rejoins list
 if sequence.find("U") == -1:  #if you have dna
    if rev == 1:
     seq = list(sequence)
     rna_seq = ['Blank']*len(seq)
     rc = range(0,len(seq))
     for c,dna in zip(rc,seq):
        if dna == 'A':
           rna_seq[c] = 'U'
        if dna == 'T':
           rna_seq[c] = 'A'
        if dna == 'G':
           rna_seq[c] = 'C'
        if dna == 'C':
           rna_seq[c] = 'G'
     rna_seq = ("".join(rna_seq))
    else:
     rna_seq = sequence.replace('T','U')
 else:
    if rev == 1: #also the reverse complement for RNA
     seq = list(sequence)
     rna_seq = ['Blank']*len(seq)
     rc = range(0,len(seq))
     for c,rna in zip(rc,seq):
        if rna == 'A':
           rna_seq[c] = 'U'
        if rna == 'U':
           rna_seq[c] = 'A'
        if rna == 'G':
           rna_seq[c] = 'C'
        if rna == 'C':
           rna_seq[c] = 'G'
     rna_seq = ("".join(rna_seq))
    else:
        rna_seq = sequence
 return rna_seq

def translate(sequence): #this creates a dictionary {} which is essentially a hash table
linking codon to amino acid.

    codon2aa = {"AAA":"K", "AAC":"N", "AAG":"K", "AAU":"N",
                "ACA":"T", "ACC":"T", "ACG":"T", "ACU":"T",
                "AGA":"R", "AGC":"S", "AGG":"R", "AGU":"S",
                "AUA":"I", "AUC":"I", "AUG":"M", "AUU":"I",

                "CAA":"Q", "CAC":"H", "CAG":"Q", "CAU":"H",
                "CCA":"P", "CCC":"P", "CCG":"P", "CCU":"P",
                "CGA":"R", "CGC":"R", "CGG":"R", "CGU":"R",
                "CUA":"L", "CUC":"L", "CUG":"L", "CUU":"L",

                "GAA":"E", "GAC":"D", "GAG":"E", "GAU":"D",
                "GCA":"A", "GCC":"A", "GCG":"A", "GCU":"A",
                "GGA":"G", "GGC":"G", "GGG":"G", "GGU":"G",
                "GUA":"V", "GUC":"V", "GUG":"V", "GUU":"V",

                "UAA":"*", "UAC":"Y", "UAG":"*", "UAU":"T",
                "UCA":"S", "UCC":"S", "UCG":"S", "UCU":"S",
                "UGA":"*", "UGC":"C", "UGG":"W", "UGU":"C",
                "UUA":"L", "UUC":"F", "UUG":"L", "UUU":"F"}

    r = int(len(sequence)/3)
    codon = ['None']*r
    prim_seq = ['None']*r
    codon[0] = sequence[0:3]

    for i in range(1, r): #here range generates integers from 1 up to r (but not including r)
        n = i*3
        codon[i] = sequence[n:n+3] #in python between (:) excludes the last integer
```

```
    for i in range(0, r):
        prim_seq[i] = codon2aa[codon[i]]


    translation = ("".join(prim_seq))
    return translation

##This function transcribes DNA and then counts the number of stopcodons in each reading frame
(1 - 6) and finally translates the reading frame that produces the  least number of stop
codons.
def count_stop_codons(sequence):

    nsequence = transcribe(sequence,0)
    rsequence = transcribe(sequence,1)


    frame_1 = nsequence
    frame_2 = nsequence[1:len(sequence)]
    frame_3 = nsequence[2:len(sequence)]

    reverse_sequence = rsequence[::-1] #this is a function of [] indexing called 'slicing'


    rframe_1 =reverse_sequence
    rframe_2 =reverse_sequence[1:len(reverse_sequence)]
    rframe_3 =reverse_sequence[2:len(reverse_sequence)]
    frames = [frame_1,frame_2,frame_3,rframe_1,rframe_2,rframe_3]
    reading_frame =['Frame_1','Frame_2','Frame_3','rFrame_1','rFrame_2','rFrame_3']

    stop_count = ['None']*6

    count = [0,1,2,3,4,5]

    for f,a in zip(frames, count): #Zip is a nice command that allows simultaneous iteration
of two same sized variables
        stopcodon = ['UAA','UGA','UAG']
        b = int(len(f)/3)
        codon = ['None']*b
        codon[0]=f[0:3]

        for i in range(1, b):
            c =i*3
            codon[i] = f[c:c+3]


        stopcodon[2]=  codon.count('UAG')
        stopcodon[1] = codon.count('UGA')
        stopcodon[0] = codon.count('UAA')

        stop_count[a] = sum(stopcodon)


    min_value = min(stop_count)
    min_index = ([i for i, x in enumerate(stop_count) if x == min_value])


    optimal_translation = ['Blank']*len(min_index)
    optimal_frame = ['Blank']*len(min_index)
    tc = range(0, len(min_index))

    for v,index in zip(tc,min_index):
        optimal_translation[v] = translate(frames[index])
        optimal_frame[v] = reading_frame[index]
    return optimal_translation, optimal_frame


import argparse
parser = argparse.ArgumentParser()
parser.add_argument("DNA_fasta_file", help="[Input]: list of DNA sequences to be translated")
parser.add_argument("output_fasta_file", help ="[Output]: list of translated sequences")
args = parser.parse_args()
ln = 1*10**7
with open(args.DNA_fasta_file, "rt") as data: #with open automatically calls file close
    text = data.read()
    sp = ([i for i, x in enumerate(text) if x == ">"]) #> demarks new sequences
    sequence_name = ['Blank']*len(sp)
    sequence = ['Blank']*len(sp)
```

```
    if sp == [0]:
        query = text
        #print(query)
        sequence_name = query.splitlines()[0]
        #print(sequence_name)
        sequence = query.splitlines()[1:ln] #obviously there are fewer lines than the ln but
this works
        #sequence = sequence[0] #removes double list
        #print(sequence)

   ##if there is more than one
    else:
        for ind in range(0,len(sp)-1):
            query = (text[sp[ind]:sp[ind+1]])
            #print(query)
            sequence_name[ind] = query.splitlines()[0]
            #print(sequence_name)
            sequence[ind] = query.splitlines()[1:ln] #obviously there are fewer lines than the
len(sequence) but this works
            #print(sequence)
            sequence[ind] = sequence[ind][0] #removes double list


        query = (text[sp[len(sp)-1]:len(text)]) #deal with the last sequence outside of the
for loop
        sequence name[len(sp)-1] = query.splitlines()[0]
        sequence[len(sp)-1] = query.splitlines()[1:len(sequence)]
        sequence[len(sp)-1] = sequence[len(sp)-1][0]
optimal_translation = ['Blank']*len(sp)
reading_frame = ['Blank']*len(sp)
count = range(0,len(optimal_translation))

for ct,sq in zip(count,sequence):
    optimal_translation[ct] = count_stop_codons(sq)[0]
    reading_frame[ct] = count_stop_codons(sq)[1]

with open(args.output_fasta_file,'w') as output:

     for name,frame,trans in zip(sequence_name,reading_frame,optimal_translation):
         for f in range(0, len(trans)):
             temp = [name,frame[f]]
             temp = "_".join(temp)
             output.write(temp)
             output.write('\n')
             output.write(trans[f])
             output.write('\n')
         output.write('\n')
```

C3: Simple - Circularise

```
#!usr/bin/env python3
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("fasta_file", help="[Input]: list of  sequences to be circularised")
parser.add_argument("output_fasta_file", help ="[Output]: circularised sequences")
parser.add_argument("-p","--probability",type=float, help="custom probability (default =
0.005)")
parser.add_argument("-max","--maximum_size",type=int,help="maximum size of output sequence")
parser.add_argument("-min","--minimum_size",type=int,help="minimum size of output sequence")
parser.add_argument("-r","--repeat",type=int,help="change behaviour to optimise repeat size
(default: optimise genome size)")
args = parser.parse_args()
if args.probability:
    print ("custom probability turned on")
if args.maximum_size:
    print("maxium_size turned on")
if args.minimum_size:
    print("minimum_size turned on")
if args.repeat:
    print("behaviour set to optimise size of repeat")
print("Circularising...")
import math

##calculates the probability of a string of length L occuring in a DNA seq.
def seq_prob (L):
    P = 0.25 ** L
    return P

##determine the number of events in a sequence of size S with a sub string size of L
def events (S,L):
    P = S - (L - 1)
    return P

#calculate probability of exactly b successes
# ** raises number to a power.
##where: a = number of events, b = number of successes, c = probability of success

def binomial_prob(a,b,c):
    d = 1 - c
    P = math.factorial(a)/ (math.factorial(a-b)* math.factorial(b)) * c**b * d**(a-b)
    return P

# Use the poisson distribution to approximate
def poisson(lmbda,b):
    e = 2.71828
    P = (lmbda ** b * e ** -lmbda) / math.factorial(b)
    return P

def lambd_a (eventn,sprob):
    lmbda = (eventn * sprob)
    return lmbda

#calculate probability of at least two occurances of sequence
#  where a = number of events and c = probability of success
def cooccur_prob(a,c,lmbda,SeqL):
    d = 1 - c
    noccur = (d ** a)
    occur = 1 - noccur
    if SeqL > 9999:
        once = poisson(lmbda,1)
    else:
        once = binomial_prob(a,1,c)
    P = occur - once
    return P

##returns the probability that the linear contig is circular. That is the probability of a n
base region being repeated
# twice or more
def pcircular(SeqL,SLength):
    sprob = seq_prob(SLength)
    eventn = events(SeqL,SLength)
    lmbda = lambd_a(eventn,sprob)
    #print(sprob)
```

```
    #print(eventn)
    co_prob = cooccur_prob(eventn,sprob,lmbda,SeqL)
    #print(co_prob)
    return co_prob


##for a sequence of size S how big does a sub string L have to be for it to be >99.5% certain
that duplication wasn't due to chance
##This is inefficient and only works for sequence lengths up to 10000. Above that values are
too large to compute
# in factorial functions
# Above 10KB the possion distribution is used as an approximation of the binomial

def framesize(SeqL):
 y = 0
 co_prob = 1

 if args.probability:
    cusprob = args.probability
    while co_prob > cusprob:
          y = y+1
          co_prob = pcircular(SeqL,y)
 else:
    while co_prob > 0.005:
           y = y+1
           co prob = pcircular(SeqL,y)
 print('Probability of repeat ',co_prob)
 return y

 ##Script will return index values of duplicated elements in a list, modified from stack
overflow user 'PaulMcG'
##Two methods:
#def list_duplicates_of(seq,item):
    #start at -1 nice way to include a +1 term in a loop
    #while True will loop indefinitely, in this case untill the seq.index returns an error
    #start_at = loc means that when a duplicate is found, next time the seq.index runs it will
consider all the elements after that element
    # but it will know their true index position
    #start_at = -1
    #locs = []
    #while True:
        #try:
            #loc = seq.index(item,start_at+1)
        #except ValueError:
            #break
        #else:
            #locs.append(loc)
            #start_at = loc
    #return locs

#numpy useful scientific package, esp. for linear algebra etc.
#nonzero - find all instances of conditions that are true
#flat nonzero - give terms in 'flattened form' ie it levels out a nested list ([a,b],[c,d]) =
(a,b,c,d)
#this function will find all elements > than limit and set them to 0
def limit_size(size,limit,method):
 import numpy as np
 x = np.array(size)
 if method == 1:
    a = list(np.flatnonzero(x>limit))
 if method == 0:
    a = list(np.flatnonzero(x<limit))
 #print(a)
 x[a] = 0
 x = np.array(x).tolist()
 return x

#source = "AKTGRGRGEDSJAADSJSJWFAFDFAFK"
#print(list_duplicates_of(source, 'B'))

##Using defaultdict this will find all duplicated elements in a single stroke, and return them
and their index positions.
#Default dict makes a dict in the form (repeated element,[index_1, index_2]). Default dict is
used in place of dict, as default dict
#will not give an error if it encounters an element not in the dictionary, instead it saves it
as a new key.
```

```
#Using list as the default_factory, it is easy to group a sequence of key-value pairs into a
dictionary of lists.
#enumerate is being used here to give both the count and the current item being enumerated,
this will give the index for each item.
#tally is a default dict, so when an item is repeated it is saved to the same key and updates
the (i) element.
# the return statement returns key locs, produced by the for command. the for command only
rpduces key,locs if len(locs) >1 i.e. if key is repeated.

#I've added lines to deduce the size of the contig

from collections import defaultdict

def list_duplicates(seq):
    #size = []
    tally = defaultdict(list)
    for i,item in enumerate(seq):
        tally[item].append(i)
    return ((key,locs) for key,locs in tally.items()
                            if len(locs)>1)

def largest_repeat(seq):
    size = []
    #dup = []
    for key,locs in list_duplicates(seq):
        r = locs[1] - locs[0]
        size.append(r)
    if args.maximum_size:
        size = limit_size(size,args.maximum_size,1)
    if args.minimum_size:
        size = limit_size(size,args.minimum_size,0)
    max value = max(size)
    if max_value == 0:
       max_value = []
    max_index = size.index(max_value)
    return max_index



# We don't need to worry about reading in every frame
#list_duplicates will have duplicates ordered by index pos of first apperance, if two
sequences are same size, always take the lowest
#this will happen autmatically with this script

##To add - can pass a sequence or a list of sequences
# can pass a minimum size of genome to circularise [optional - chance co-prob criteria untill
a size of this genome is made]
# can pass a mandatory probability
# ln means it can deal with sequences of any length
#we need sp-1 because the length command gives true length i.e. two elements = 2 not 1(0,1)


ln = 1*10**7
#fasta_file = input("File name?")
with open(args.fasta_file, "rt") as data: #with open automatically calls file close
    #The enumerate command gives an index for where the > is found
    text = data.read()
    sp = ([i for i, x in enumerate(text) if x == ">"]) #> demarks new sequences
    sequence_name = ['Blank']*len(sp)
    sequence = ['Blank']*len(sp)
    #print(sp)
    ##if there is only one sequence
    if sp == [0]:
        query = text
        #print(query)
        sequence_name = query.splitlines()[0]
        #print(sequence_name)
        sequence = query.splitlines()[1:ln] #obviously there are fewer lines than the ln but
this works
        #sequence = sequence[0] #removes double list
        #print(sequence)

   ##if there is more than one
    else:
        for ind in range(0,len(sp)-1):
            query = (text[sp[ind]:sp[ind+1]])
            #print(query)
```

```
            sequence_name[ind] = query.splitlines()[0]
            #print(sequence_name)
            sequence[ind] = query.splitlines()[1:ln] #obviously there are fewer lines than the
len(sequence) but this works
            #print(sequence)
            sequence[ind] = sequence[ind][0] #removes double list


        query = (text[sp[len(sp)-1]:len(text)]) #deal with the last sequence outside of the
for loop
        sequence_name[len(sp)-1] = query.splitlines()[0]
        sequence[len(sp)-1] = query.splitlines()[1:len(sequence)]
        sequence[len(sp)-1] = sequence[len(sp)-1][0]
        #print(query)
        #print(sequence_name)
        #print(sequence)

#if sp == 0 sequence must be contained in a list otherwise DNA read as first integer
circgenome = ['Blank']*len(sp)
count = range(0,len(sp))

for c,DNA in zip(count,sequence):
   print(sequence_name[c])
    #print(DNA)
        # sindex and gindex consider only the first and second element of the repeat
   if args.repeat:
       f = args.repeat-1;
       error = 0
       while error == 0:
         SeqL = len(DNA)
         f = f+1
         eventn = events(SeqL,f)
         repeats = ['Blank'] * eventn
         dupe = []
         for i in range(0,eventn):
               repeats[i] = DNA[i:f+i]

         for dup in list_duplicates(repeats):
               dupe.append(dup)

         try:
               index = largest_repeat(repeats)
               #print(index)
         except ValueError:
               print ("Maximum repeat size exceeded")
               #circgenome[c] = ' '
               error = 1


         except TypeError:
               print ("Maximum repeat size exceeded")
               #circgenome[c] = ' '
               error = 1


         else:
               sindex = (dupe[index][1][0])
               gindex = (dupe[index][1][1])
               #print(sindex)
               #print(gindex)
               circgenome[c] = (DNA[sindex:gindex])

               print('Searching for repeats of minimum length',f)
               print('Circularising at',(dupe[index]))
               print('Circularised genome size is',(gindex - sindex))

   else:
       SeqL = len(DNA)
       f = framesize(SeqL)
       print('Searching for repeats of minimum length',f)
       eventn = events(SeqL,f)
       repeats = ['Blank'] * eventn
       dupe = []
       for i in range(0,eventn):
             repeats[i] = DNA[i:f+i]

       for dup in list_duplicates(repeats):
```

```
                dupe.append(dup)

        try:
                index = largest_repeat(repeats)
                #print(index)
        except ValueError:
                print ("Oops!  This genome couldn't be circularised [Try increasing the
probability]")
                circgenome[c] = ' '

        except TypeError:
                print ("Oops!  This genome couldn't be circularised [Try changing the min max
boundaries]")
                circgenome[c] = ' '

        else:
                sindex = (dupe[index][1][0])
                gindex = (dupe[index][1][1])
                #print(sindex)
                #print(gindex)
                print('Circularising at',(dupe[index]))
                print('Circularised genome size is',(gindex - sindex))
                circgenome[c] = (DNA[sindex:gindex])

#print(circgenome)
with open(args.output_fasta_file,'w') as output:

     for name,genome in zip(sequence_name,circgenome):
         #for f in range(0, len(circgenome)):
                output.write(name)
                output.write('\n')
                output.write(genome)
                output.write('\n')
                output.write('\n')
```

C4: Bootstrap – Jplace

```
#!/usr/bin/bash
#This script was written using git bash for windows. On another OS '\).{3}' might be needed
with grep to escape the parenthesis.
#Where the reference tree, RAxML_bipartitions.ref_tree.out and the jplace tree,
RaxML_portableTree.EPA_tree.out.jplace have identical topology.
#Two output dataset files will be produced that can label a jplace tree with text or branch
symbols on iTOL.
#nodeID2 is the nodeID + 1, this ensures the bootstrap labels are consistent with those
produced from iTOL automatically on the reference tree. To use the true nodeiDs replace
nodeID2 with nodeID the the paste command.
grep -o -E ').{3}' $1  | sed 's/)//' | sed 's/:0//' | sed 's/://' >BS
grep -o -E '.{4}).{0}' $2 | sed 's/^.*{//' | sed 's/})//' >nodeID
while read p;
do echo "$(($p + 1 ))" >> nodeID2
done <nodeID
a=$(cat BS | wc -l)
yes "1" | head -$a >  Symbol
yes "#B2B8D8" | head -$a > Colour
yes "0" | head -$a > Fill
yes "0.5" | head -$a > Position
cat > Bootstrap_Symbol.txt <<EOF
DATASET_SYMBOL
SEPARATOR COMMA
DATASET_LABEL,Bootstrap_Symbol
COLOR,#B2B8D8
MAXIMUM_SIZE,10
DATA
EOF
paste -d , nodeID2 Symbol BS Colour Fill Position >> Bootstrap_Symbol.txt
yes "normal" | head -$a>Style
yes "#000000" | head -$a >Text_Colour
yes "1" | head -$a >Size
yes "0" | head -$a >Rotation
cat > Bootstrap_Label.txt <<EOF
DATASET_TEXT
SEPARATOR COMMA
DATASET_LABEL,Bootstrap_Label
COLOR,#000000
DATA
EOF
paste -d , nodeID2 BS Position Text_Colour Style Size Rotation>> Bootstrap_Label.txt
rm -f nodeID2 BS Position Text_Colour Style Size Rotation Colour Fill Position Symbol-Node
```