AMPLIpyth: A Python Pipeline for Amplicon Processing

David Jacob Meltzer

0803837m

MSc Bioinformatics, Polyomics and Systems Biology

Supervisor: Dr. Umer Zeeshan Ijaz

A report submitted in partial fulfillment of the requirements for the MSc Bioinformatics, Polyomics and Systems Biology Degree at the University of Glasgow

August 2015

Summary

Next generation sequencing technologies have revolutionized the way in which microbial communities are analyzed. The massively parallel nature of these technologies allows for rapid and accurate sequencing of the contents of these communities. This combined with the culture-independent 16S and 18S rRNA sequence amplification method had dramatically increased the ability to analyze these types of data. Analysis of these metagenomic datasets is complicated by the variety of platforms that generate them and the variety of tools optimized to analyze the resulting data. Currently, there exist analysis pipelines such as `QIIME` and `mothur` to process these data. These pipelines, while powerful, have certain limitations that leave room for improvement. In this project we sought to develop a software workflow which could analyze amplicon based metagenomic datasets and give detailed information about the microbial community profile. In addition this program would be optimized for the Illumina sequencing platform and would require minimal user input and upon execution, no user input at all. This was accomplished with the creation of the `AMPLIpyth` software. `AMPLIpyth` uses a python wrapper to call server sides programs using subprocesses and generates an HTML output with the pertinent results of the analysis. The program was heavily tested using a fecal sample dataset for Crohn's disease and through the comparison of the results to an existing bash based workflow.

Acknowledgments

I would like to thank my supervisor Dr. Umer Zeeshan Ijaz for his mentorship and all of his assistance throughout this process.

I would like to thank Gianna Hayner for her assistance in editing this document.

Contents

Abbreviations and Definitions

NGS = Next Generation Sequencing

BP = Base Pair

EEN = Exclusive Enteral Nutrition

CD = Crohn's Disease

CSV = Comma Separated Value

OTU = Operational Taxonomic Unit

Amplicon = The product of a PCR Amplification

Raw Reads = The sequencing data directly from the sequencer before any processing or analysis

Introduction

Microbial communities are as numerous as they are widespread. These communities are either directly or indirectly involved with every living organism on Earth[1]. In the past, the study of microbial communities was limited by the ability to culture these microbes within a laboratory environment. With less than 1% of prokaryotes able to be laboratory cultured, community studies were incredibly difficult[2]. Next-generation sequencing technologies, also called second-generation technologies, have revolutionized the investigation of these communities and because of this, metagenomics has exploded as a research topic in recent years[3].

Metagenomic investigations enable the profiling of microbial communities without the need to culture them in laboratory environment. This is made possible by next generation sequencing (NGS) that allows for the massively-parallel sequencing of millions of DNA fragments[4]. One of the predominant techniques is the amplification of ribosomal RNA genes such as 16S/18S rRNA. These genes are fairly conserved at species level and can be used as a DNA barcoding strategy[5,6]. There are two main approaches for amplicon processing: a reference-based approach and a *de novo* approach. A reference-based approach uses a well curated database such as SILVA, BERGEYS taxonomy for RDP as a reference point from which sequencing data is assembled against. A *de novo* approach involves the clustering of reads by aligning them against each other and then binning the reads based on a similarity threshold [7]. Regardless of the approach used, the process is computationally intensive and may require the use of a cohort of software tools, each with their own arguments and complications, and each tailored to a specific

part of the analysis [8]. Therefore, there exist several analysis pipelines for metagenomic datasets. Of these, two of the most widely used are `QIIME` and `mothur`. Both `QIIME` and `mothur` are capable of handling large amounts of data in a variety of NGS formats and produce highly accurate analyses [8,9]. These programs are not without their downsides. `QIIME` can be very difficult to install, requiring a significant amount of dependencies, and both `QIIME` and `mothur` require user intervention (command-line interactivity) during their function [10,11]. With this in mind, we sought to create an analysis pipeline that lacked these downsides as well as an optimized workflow informed by recently published work on error profiling for the Illumina platform[12]. Hence, the aim of this project was to construct an automated pipeline for metagenomic analysis that requires minimal human interaction outside the initial set up and has the optimal subprocesses.

Analysis

Approach

The program was coded entirely in python version 2.6.6 as it is a flexible programming language that interacts easily with the underlying unix based operating systems which most bioinformatic servers will have. The python program is a wrapper around multiple other programs to perform the analysis tasks (described below). These components would need to be called by the program and run from within the code. To do this the python module `subprocess` was used to run shell commands in the operating system. A sample dataset (described below) was provided and used to test functionality at every step of development. The python package `MatPlotLib` was used to generate all graphical outputs.

Dr. Ijaz's command line tutorial "Illumina Amplicons OTU Construction with Noise Removal" found at www.tinyurl.com/JCBioinformatics was used to develop the framework for the pipeline. It was also used as part of the testing process as the sample data used in the tutorial is the same as used by the pipeline.

This pipeline is optimized for Illumina sequencing. Illumina sequencing is the dominant sequencing technology in the research world and with the recent announcement by Roche discontinuing the 454 platform, the market share of Illumina is set to increase[13]. Unpublished research by D'Amore et al determined through benchmarking that the Illumina platform has the highest accuracy and lowest error rates over the other commercially available platforms (D'Amore et al, A comprehensive benchmarking study of protocols and sequencing platforms for 16S rRNA community profiling, Unpublished). With this in mind, work done previously

by Schirmer and Ijaz et al was used to inform on the software choices used by this pipeline. Schirmer and Ijaz et al investigated which combination of tools resulted in the best error correction on Illumina generated dat. They determined that using *sickle, BayesHammer* and *PANDAseq* for error correction resulted in a 93% decrease in substitution error rates when compared to alternate choices[12].

Programs Used

Sickle

`Sickle` is a sequence-trimming program that uses a sliding window, length and phred quality of the sliding window to determine where to trim the sequences. In practice, the quality usually falls off at the end of the reads. `Sickle` has settings to accommodate both single and paired-end reads and a variety of quality scoring formats[14]. `Sickle` was used to quality filter the sample reads to prepare them for downstream applications.

SPAdes

`SPAdes` is an open-source assembler software that can be used for single and multicellular assemblies and produces excellent assemblies for uncharacterized bacteria. Many environmental bacteria cannot be easily cloned and thus their amplification and subsequent sequencing using current technologies is difficult if not impossible. The `SPAdes` assembler uses a novel approach to deal with these difficulties and is capable of quickly producing accurate assemblies [15]. The assembly function of the `SPAdes` assembler was not used in the `AMPLIpyth`

program. SPAdes was used only to perform error-correction as it is bundled with BayesHammer which is used for pre-correction of reads before performing assembly [16].

PANDAseq

PANDAseq is assembly software used to assemble Illumina paired-end reads. The program takes advantage of the overlap between paired-end reads for short DNA fragments (i.e., 16S/18S variable regions that are small enough for the reads to overlap). As the overlap increases there is a corresponding increase in the ease of correcting incorrectly called bases by using the sequence mate's overlap. PANDAseq algorithmically determines the necessary overlap and performs correction on that region, systematically going through the entire provided sequence [17]. PANDAseq was used here in its assembler function.

UPARSE

The UPARSE pipeline functions similar to the BLAST algorithm however the UPARSE not only allows for alignment against a reference database similar to BLAST, but also allows clustering of reads, and the removal of chimeras (*de novo* as well as reference based approach). It is substantially faster than contemporary alignment tools as the search algorithm is heuristic in nature [7]. UPARSE pipeline was used here to perform several analyses in the pipeline (UPARSE or USEARCH is also used in QIIME and mothur). These are described in the "Product" section below.

Mafft

The `mafft` program performs multiple sequence alignments[18,19]. In `AMPLIpyth`, `mafft` takes the sequences in FASTA format and then generates a multiple sequence alignment of these sequences. This alignment was used later by *`FastTree`* to generate a phylogenetic tree.

FastTree

`FastTree` program is a program for generating phylogenies. Many phylogeny generating programs use a distance matrix to store relationships however as the size of the matrix increases so does the computational requirements. The `FastTree` program "stores sequence profiles of internal nodes in the tree" [20]. This allows for a decrease in the computational requirements in large trees. `FastTree` was used here to generate a phylogeny of the identified OTUs.

Ribosomal Database Project (RDP) Classifier

The `RDP Classifier` is "a naïve Bayesian classifier" that is well suited to give the taxonomy of the sequences at different levels (Phylum, Class, Order, Family, and Genus)[21]. In the `AMPLIpyth program` the `RDP Classifier` was used to identify the taxonomies that each of the identified OTUs belong to.

Python Program Development

All python programming was performed using the PyCharm Professional Edition Integrated Development Environment (IDE). This IDE is specifically designed for python coding and provides a variety of tools and plugins to analyze and optimize code[22]. Code was locally produced in PyCharm and tested on Dr. Ijaz's bioinformatics server.

Sample Data

A dataset of twenty-four fecal samples was provided by Dr. Ijaz**.** The fecal samples had been processed as part of a separate study in which Dr. Ijaz was an investigator. In this study Quince et al sought to investigate the differences in gut flora between healthy children and children with CD. The children with CD were put on the EEN program and their feces sampled at five time points referred to by the letters A through E: A) before or within 6 days of starting EEN, B) 16 days after staring EEN, C) 32 days after starting EEN, D) 54 days after starting EEN and E) 63 days after the end the EEN program. The "E" group was to ascertain if the CD children's flora had returned to their pre-EEN levels. These samples were then compare to the healthy children, referenced as group "H" (Quince, Loman and Ijaz et al. Extensive modulation of the fecal metagenome in children with Crohn's disease during exclusive enteral nutrition. Submitted for Publishing). The samples from group E were those used in testing *AMPLIpyth*. The files are available on the attached CD.

Sample Processing

The fecal samples had been processed per standard protocols and bacterial DNA was isolated and purified using the chaotropic method. 16S rRNA sequencing of the V4 region was performed on the MiSeq (Illumina) platform using 2 × 250 bp paired-end reads. The V4 region was amplified using fusion Golay adaptors barcoded on the reverse strand. The forward 16S rRNA primer sequence 515f (GTGNCAGCMGCCGCGGTAA) was used. The reverse primers, barcodes and adaptors were identical to those described previously. Amplicons were purified with AMPure XP DNA purification beads (Beckman Coulter, Danvers, MA, USA) according to the manufacturer's instructions, and eluted in 25 µl of Elution Buffer (Qiagen, 19086, UK). Subsequently, amplicons were quantified with use of KAPA SYBR® FAST qPCR Kit (Kapa biosystems, KK4824, UK), diluted to 40 pM and spiked-in with 40 pM of genomic DNA to avoid base-calling issues due to low base diversity (Method barrowed with permission of Dr. Ijaz from Quince, Loman and Ijaz et al. Extensive modulation of the fecal metagenome in children with Crohn's disease during exclusive enteral nutrition. Submitted for Publishing).

Product

Program overview

An amplicon-processing pipeline was successfully produced using python. This pipeline named, `AMPLIpyth`, is presented in figure 1.

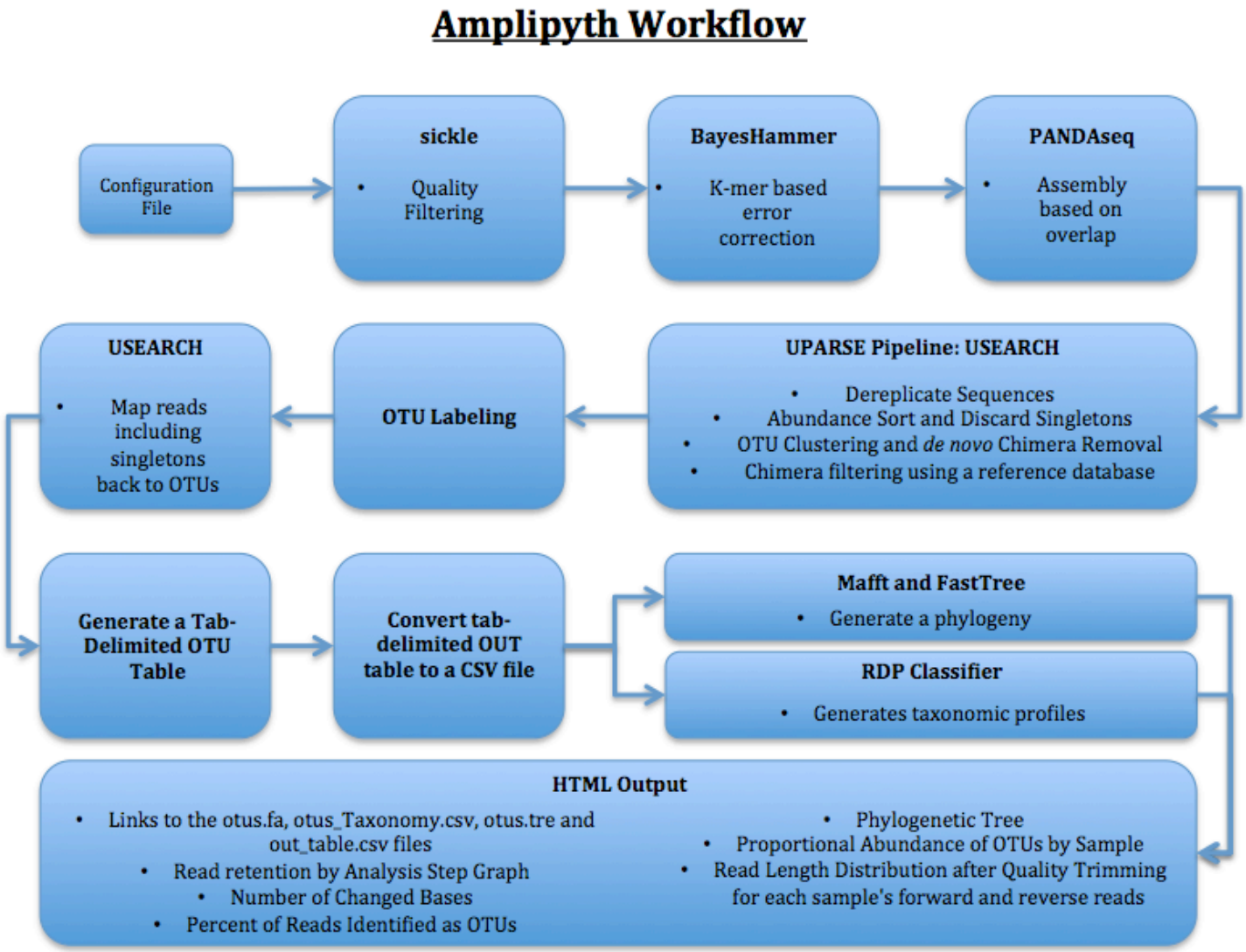

Figure 1: Amplipyth Workflow. The program takes a configuration file which contains an alias for each sample and the absolute filepath to their location. The files are then processed through the workflow as described. An HTML output is generated at the end of the workflow.

Program Design

Running *AMPLIpyth* requires three arguments: python [*AMPLIpyth Version*] –c [Configuration File] –o [Analysis Output Directory] –t [OTU Output Directory]. The program requires the user to produce a configuration file which includes all of the files to be analyzed and the required arguments for the programs involved in the analysis. This configuration file has a required format shown in figure 2. The pipeline requires no user input after it has been initiated.

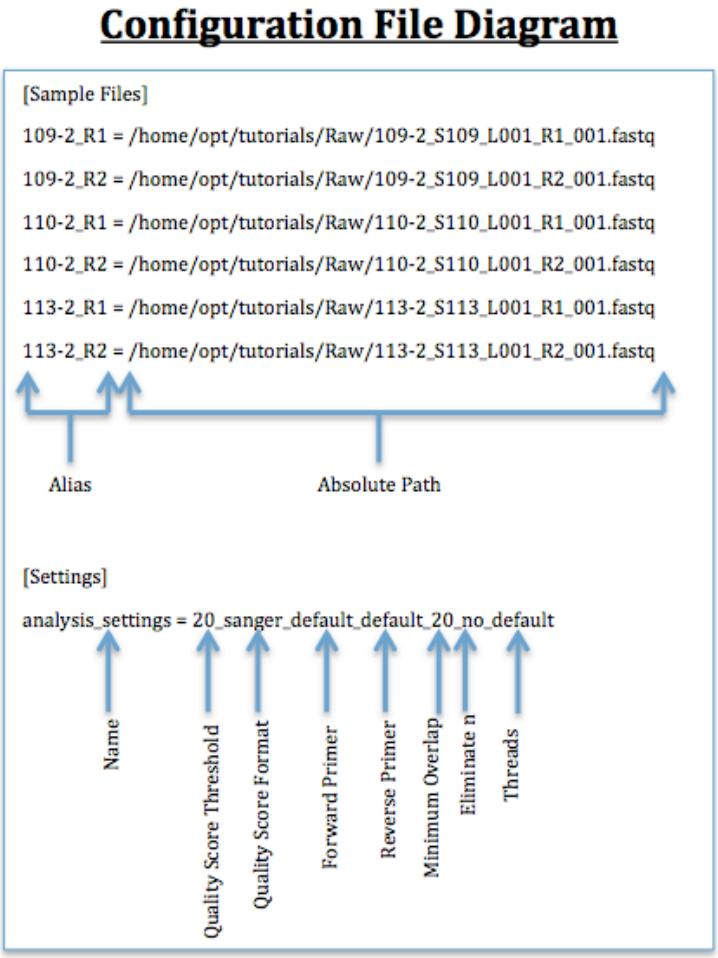

Figure 2: Configuration File Diagram. The program takes a configuration file which contains an alias for each sample and the absolute path to their location. The user must also specify settings within the setting section. These settings are given as a string that *Amplipyth* parses and uses in the appropriate functions.

The pipeline begins by checking to ensure that the programs `sickle`, `USEARCH`, `PANDAseq`, `fastqc`, `SPAdes`, `mafft` and `FastTree` are found on the system. The program generates the analysis output directory provided by user in the arguments. The python module `ConfigParser` then parses in the specified configuration file. This configuration file contains an alias for each file found in the "name" value. This name value is then used to make a single directory for each of the sample pairs. These directories are referred to as the "main directory" for each sample henceforth. Within each directory a directory named "Raw" is created. This raw directory will contain the raw reads. The raw reads are then copied from their location, given as the absolute path, found in the "value" value in the configuration file, to the "Raw" directory of the appropriate directory.

Quality filtering is then performed using `sickle`. Briefly, each read has a corresponding quality score. This quality score indicates the probability that the base was called correctly during sequencing. The equation below describes this (Equation 1) [23]

$$Quality\ (q) = -10 \log 10(p)$$

Equation 1: The formula for determining quality score.

The quality score used to determine trimmed reads is provided in the "settings" section of the configuration file. Likewise the quality score system is specified here as well. The quality score value is not variable and it is advised that the user examine the quality information of the reads before analysis to select the appropriate score. `Sickle` quality filtering generates files with the suffix

"_trim.fastq" as well as the singlet file suffixed "singlet.fastq". These files are then used to produce a read length distribution graph for the forward and reverse trimmed reads. These files are originally placed into the main directory where the program itself is found but are moved into a collected outputs folder at a later step.

The trimmed forward and reverse FASTQ files are then passed to SPAdes which calls the BayesHammer program and performs error correction. The SPAdes program takes no input from the configuration file and performs no assembly. The SPAdes assembly function is not required for this program to function. The SPAdes program, through BayesHammer, generates an output directory named "corrected" within each main directory. Within this "corrected" directory are the error-corrected forward, reverse and unpaired FASTQ files. SPAdes produces a log, spades.log, which is deposited in the main directory. This log contains information as to the error correction performed by BayesHammer. This information is later used in downstream statistics.

Due to an issue with the BayesHammer program, identifying information concerning the directionality of the reads is stripped from the reads (i.e., tags to tell whether the read is a forward or a reverse reads) during BayesHammer error correction. Without identifying information, downstream analysis with PANDAseq is impossible. To correct for this a function was written to write a new file with the correct information and overwrite the incorrect file.

The corrected forward and reverse reads are then processed with PANDAseq which overlaps the paired end reads. The arguments used by PANDAseq

are found in the settings section of the configuration file. PANDAseq outputs an overlap file located in the main directory for each sample.

Before the program can run UPARSE the overlap.fasta files for each sample need to have a UPARSE formatted barcode labels for each read. This is accomplished through a function that adds this required barcode to each read and outputs the appropriate FASTA file.

At this point in the program, analysis within the Analysis Output Directory is complete. The program next generates the OTU Output Directory. All further analysis takes place within this directory.

In preparation for UPARSE analysis the overlap.fasta files from each sample are then multiplexed into a single multiplexed.fasta file, linearized and dereplicated. These three steps are the most time consuming and computationally intensive of the entire program taking an average of six minutes.

The UPARSE program is then used to perform a series of functions within the AMPLIpyth pipeline: abundance sorting, the discarding of singletons, OTU clustering and *de novo* chimera removal. UPARSE is then used to remove chimeras using a reference database. The sequences are then labeled with the fasta_number.py python script created by Robert Edgar http://drive5.com/python/python_scripts.tar.gz. This generates the otus.fa file. UPARSE is then used to map the singletons back to the OTUs. Another of Robert Edgar's python scripts, uc2otutab.py, is then used to generate a tab-delimited OTU table. This table is then converted to a CSV file for further analysis. Mafft is then used to perform multisequence alignment on the representative OTU

sequences. `FastTree` is then used to generate a phylogenetic tree. The program will then display to the user the statistics involved in the OTU construction step: total reads, total reads dereplicated, total reads dereplicated and singletons removed, OTUs after *de novo* chimera removal, OTUs after database assisted chimera removal and the final OTUs after matching. This concludes the analytical portion of the pipeline

`AMPLIpyth` produces a minimum of seven outputs: a read retention by step graph, a changed bases graph, the percent of reads in OTUs graph, a stacked bar graph showing the proportion of the total OTUs comprising the twenty most common and all others, the phylogeny and a length distribution graphs for the forward and reverse reads. These output are provided in an HTML output with download links to the `otus.fa`, `otus_Taxonomy.csv`, `otus.tre` and `otu_table.csv`. The number of outputs will vary according to the samples as a length distribution graph is produced for each samples forward and reverse reads.

Design Considerations

The use of a configuration file is ideal because it allows for the user to prepare multiple different analysis sets and process them without making any changes to `AMPLIpyth` itself. This also allows for jobs to run concurrently assuming that there is sufficient space and processing power. The configuration file also allows for an additional degree of reproducibility as it can be provided, run with the pipeline and yield the same result.

The separation of the output directory and the OTU output is necessitated by the program design. By keeping the analysis directory populated only by sample data ensures that the code needed to process the data remains relatively small. Additional functions and coding would have been required to accommodate the additional contents resulting in negative impacts on the computational requirements and time requirements.

The program is strictly divided between an analysis section and a visualization section, the exception to this being the read length distribution graphs. It was advantageous programmatically to have the read length distribution graphs generated after quality filtering but before `SPAdes`. This prevented the need for complicated functions to go back and access each of the folders individually to generate the outputs.

Certain design choices were used to ensure that the graphical outputs conveyed the most information possible with the least amount of clutter. The "Read Retention by Analysis Step" graph does not provide the specific numbers of the reads lost at each step but as a proportion of the whole. This was done to allow the user to visualize all samples at once while still being able to gather meaningful information from it. This choice was not carried over to the "Changed Bases" graph. It was decided that understanding the actual number of changed bases was important to the interpretation of the data. Visualizing the data in this method does not detract from the readability of the graph as it would have done in the "Read Retention by Analysis Step" graph.

The “Abundance Chart” which displays the twenty most abundant OTUs with the rest grouped into an “Other_Otus” category was designed to provide a substantial amount of detail and is tied for the most important output with the phylogeny above it. Personal communication with Dr. Ijaz led to the decision to display the twenty most abundant OTUs as opposed to another amount. This amount is arbitrary and is not readily editable for the user however as this is open-source software a knowledgeable user would be able to edit the code to what ever they would like. Future iterations of the program will add this as a user specifiable option in the configuration file.

Evaluation

Testing

The sample dataset, described previously, was used to test the program at all steps of development as well as its computational and time requirements. This sample set, representing a modest amount of data takes between sixteen and sixty minutes to run depending on server load. Program stability was determined through thorough testing. Unexpected termination of the program was found to occur when the user profile exceeded its memory allotment. It is therefore recommended that the user ensure they have sufficient storage space before running `AMPLIpyth`. Computational walls were never encountered during testing.

Two separate configuration files were used for the testing process. As noted above, the absolute paths for each of the files was given as well as an alias. This alias was the sample number followed by the abbreviation `_R1` or `_R2`. These corresponded to whether the read was the forward (`R1`) or the reverse (`R2`). The first configuration file contained all of the sample files while the second contained only five sample files. The results of this second configuration file could not be compared as described below and as such it was only used to ensure that changes to the pipeline did not result in unexpected program failure.

As described previously, the program requires certain parameters to be provided within the configuration file. For both the full and abridged sample files the configuration parameters used for `sickle` were a quality score of twenty and a quality score format of “Sanger”. For the `PANDAseq` configuration inputs the, arguments passed to the subprocess were `pandaseq –f`

[forward_fastq_file] –r [reverse_fastq_file] –B -F –d [bfsrk] –o [20] > [overlap_fastq_file]. Briefly, the –f and –r arguments are the forward and reverse FASTQ files that PANDAseq requires. The –B, -F and –d arguments allow PANDAseq to ignore missing barcodes or tags, ensure that the output is in FASTQ format and limiting the output to the user respectively. The bfsrk value details what kinds of information are displayed to the user. The –o argument takes the overlap threshold. Finally, the last item is the name of the output file. These values are passed to the function in which PANDAseq is called.

The specification "default", was set for the forward, reverse and threads arguments. Default needs to be specified as part of the settings string. It cannot be left blank or the program will not function. The program searches for the value "default" and if detected, removes the argument that it is attached to from the PANDAseq call in the subprocess. The program is capable of accommodating all arguments for PANDAseq. For the remaining PANDAseq options an overlap of 20 bases was specified and for n bases, the "eliminate n" option, was specified as no. Setting the "eliminate n" option to no is also a value that the program searches for. The "no" value causes the program to remove the –N argument from the PANDAseq call in the subprocess.

To ensure that the program was functioning correctly the results of the sample dataset analysis were compared to the results of the tutorial analysis described previously. This tutorial allowed for the direct comparison of the results of the pipeline to the results of an established pipeline. Practically, as each part of

the `AMPLIpyth` pipeline was coded, its corresponding tutorial section was run on the command line and the results compared. The completed pipeline had a 1 to 1 result when compared to the tutorial outputs indicating the pipeline was functioning correctly.

The tutorial however only allowed for the testing of the analytical section of the pipeline. Testing the graphical outputs required separate verification steps that consisted of adding code (now removed) to ensure that the contents of the graphs were the correct translations of the results of the pipeline.

Testing Results

To determine the stability of the `AMPLIpyth` program and the reproducibility of its results, the program was run 50 times. The sample dataset of 24 samples (48 FASTQ files) was successfully processed by `AMPLIpyth` each time. The average run length of these tests was 17 minutes. The metadata was mined for the initial reads, trimmed paired-end reads, paired-end reads with changed bases, changed bases, failed bases, total bases, final paired-end reads and the number of overlap reads using a script written by Dr. Ijaz and available at www.tinyurl.com/JCBioinformatics. This data mining resulted in identical results between tests and when compared to the tutorial data. Table 1 is an example comparison of the tutorial result with a pipeline output.

Comparison of Reference and *Amplipyth* Results

	Initial Paired-End Reads		Trimmed Paired-End Reads		Paired-End Reads with Changed Bases		Changed Bases		Failed Bases		Total Bases		Final Paired-End Reads		Overlap Reads	
Sample	Ref	Amp	Ref	Amp	Ref	Amp	Ref	Amp	Ref	Amp	Ref	Amp	Ref	Amp	Ref	Amp
109-2	5382	5382	5368	5368	5950	5950	6553	6553	976091	976091	2633465	2633465	5352	5352	5262	5262
1-1	21165	21165	21085	21085	16464	16464	18584	18584	2332952	2332952	10373755	10373755	21007	21007	20689	20689
110-2	3745	3745	3731	3731	4139	4139	4820	4820	632337	632337	1824064	1824064	3718	3718	3643	3643
113-2	1907	1907	1904	1904	2181	2181	2400	2400	342141	342141	928343	928343	1897	1897	1856	1856
114-2	7536	7536	7518	7518	8118	8118	9774	9774	1247564	1247564	3691854	3691854	7503	7503	7402	7402
115-2	5585	5585	5564	5564	6777	6777	7292	7292	970011	970011	2688154	2688154	5548	5548	5460	5460
117-2	206085	206085	205750	205750	94371	94371	103187	103187	10505265	10505265	100681940	100681940	205399	205399	203092	203092
119-2	13616	13616	13561	13561	15282	15282	19103	19103	2464707	2464707	6636196	6636196	13530	13530	13341	13341
120-2	9784	9784	9744	9744	10552	10552	11232	11232	1541370	1541370	4779674	4779674	9713	9713	9589	9589
126-2	1689	1689	1675	1675	1978	1978	2068	2068	310653	310653	812844	812844	1667	1667	1634	1634
128-2	27086	27086	27023	27023	25396	25396	26922	26922	3462404	3462404	13273476	13273476	26964	26964	26737	26737
130-2	13278	13278	13217	13217	13510	13510	14394	14394	2157529	2157529	6507004	6507004	13176	13176	13044	13044
13-1	32388	32388	32323	32323	26874	26874	28086	28086	3204975	3204975	15785006	15785006	32261	32261	31868	31868
132-2	13130	13130	13110	13110	11183	11183	11663	11663	1587045	1587045	6479125	6479125	13078	13078	12979	12979
20-1	55596	55596	55493	55493	41505	41505	43973	43973	5003510	5003510	26993274	26993274	55402	55402	54734	54734
27-1	23450	23450	23410	23410	19390	19390	20335	20335	2644967	2644967	11498857	11498857	23334	23334	22938	22938
32-1	4586	4586	4566	4566	5366	5366	5754	5754	754967	754967	2226446	2226446	4544	4544	4465	4465
38-1	10527	10527	10499	10499	13782	13782	14076	14076	2098349	2098349	5188696	5188696	10467	10467	10346	10346
45-1	5007	5007	4992	4992	5479	5479	6124	6124	773263	773263	2387260	2387260	4973	4973	4834	4834
51-1	35597	35597	35463	35463	30262	30262	32146	32146	4194525	4194525	17422574	17422574	35362	35362	35027	35027
56-1	33292	33292	33199	33199	24805	24805	27963	27963	3351048	3351048	16372931	16372931	33118	33118	32840	32840
62-1	7358	7358	7331	7331	7736	7736	8995	8995	1181410	1181410	3589508	3589508	7299	7299	7155	7155
68-1	17025	17025	16986	16986	14167	14167	14632	14632	1836506	1836506	8354853	8354853	16919	16919	16633	16633
7-1	1	1	1	1	0	0	0	0	156	156	502	502	1	1	1	1

Table 1: **Comparison of established results with a representative result of the *Amplipyth* pipeline.** The results of the reference tutorial results are shown next to their *Amplipyth* counterparts.

During each test, *AMPLIpyth* correctly identified the 498 distinct OTUs identified in the reference tutorial. To ensure that the results were identical, the otu_table.csv file was examined after each test. The otu_table.csv contains a list of the identified OTUs along with the number of reads from each sample that were identified as parts of that OTU. Each otu_table.csv file was compared to the reference tutorial otu_table.csv file to ensure a match. The `otu_table.csv` file is available in the HTML output folder provided with this report. Upon the completion of the pipeline an HTML output was successfully generated containing a read retention by step graph, a changed bases graph, the percent of reads in OTUs graph, a stacked bar graph showing the proportion of the total OTUs comprising the twenty most common and all others, the phylogeny and the length distribution graphs for the forward and reverse reads. A HTML sample output is available on the attached CD. Below is a walkthrough of a set of representative results.

The HTML output begins with the links to pertinent files (Figure 3). It then displays the "Read Retention by Analysis Step" graph. During each step of the pipeline a number of reads are lost. Based on the sample analysis, the largest loss in reads occurs in the overlapping step. A representative sample of these types of graph is displayed in figure 4.

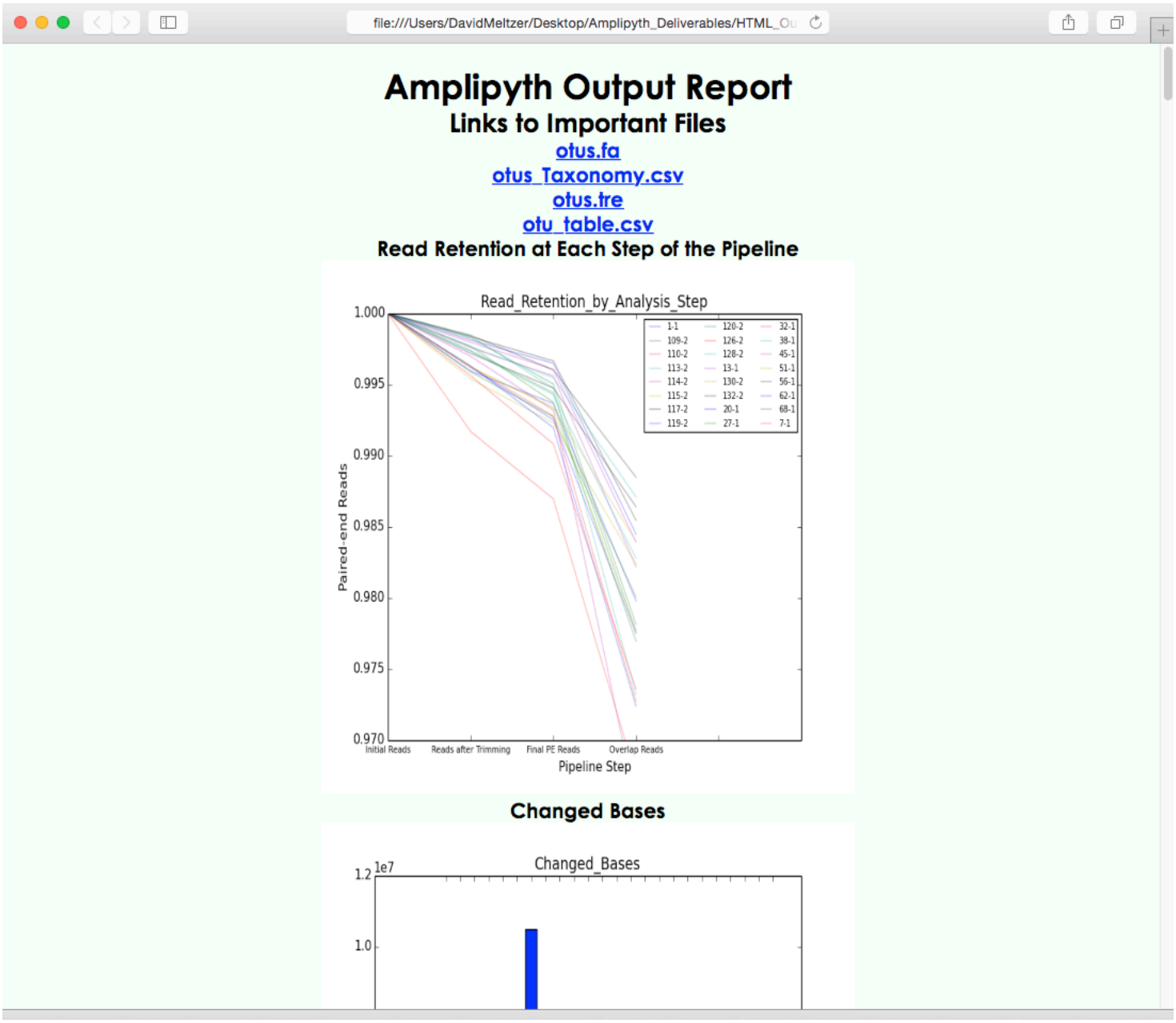

Figure 3: The `AMPLIpyth` HTML output. This output is generated upon the completion of the program and contains links to important files and outputs useful for analysis.

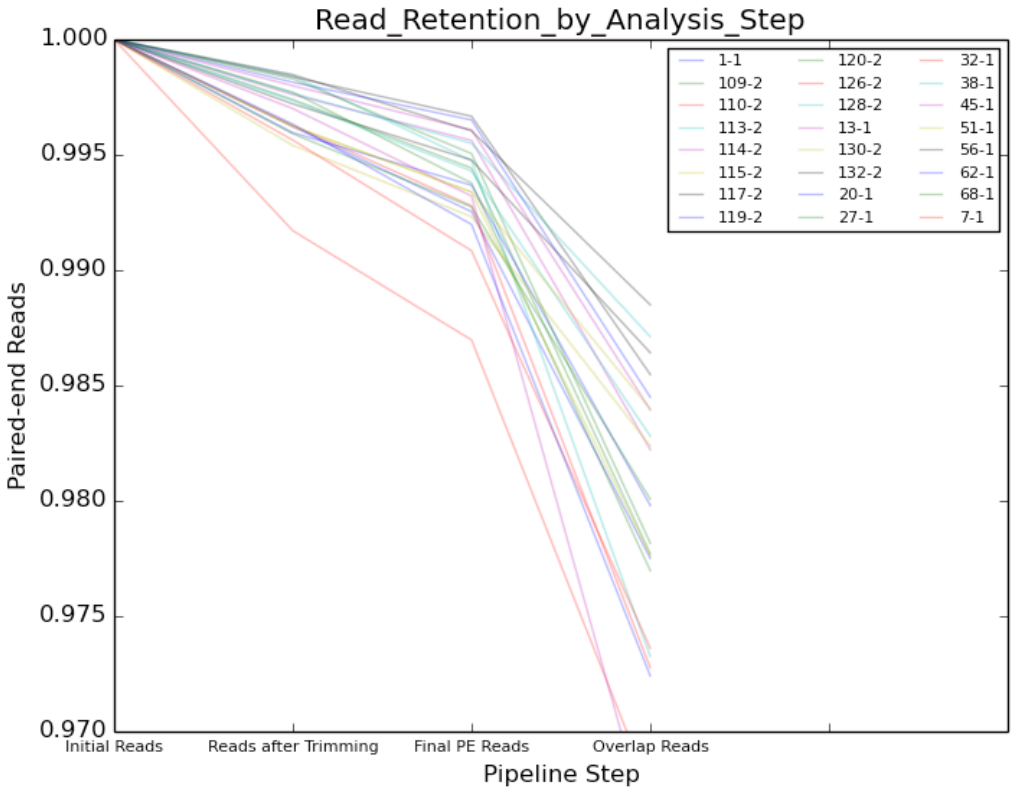

Figure 4: Read Retention by Analysis Step Graph. This graph displays the proportional read retention by analysis step of the course of the pipeline. The sample alias is displayed in the key on the upper right corner. The x-axis gives the pipeline step and the y-axis gives the proportion of the paired-end reads lost.

This graph is followed by a changed bases graph. This graph gives information on the number of bases changed during the data analysis for each of the samples. As stated previously, this graph is not proportional (Figure 5).

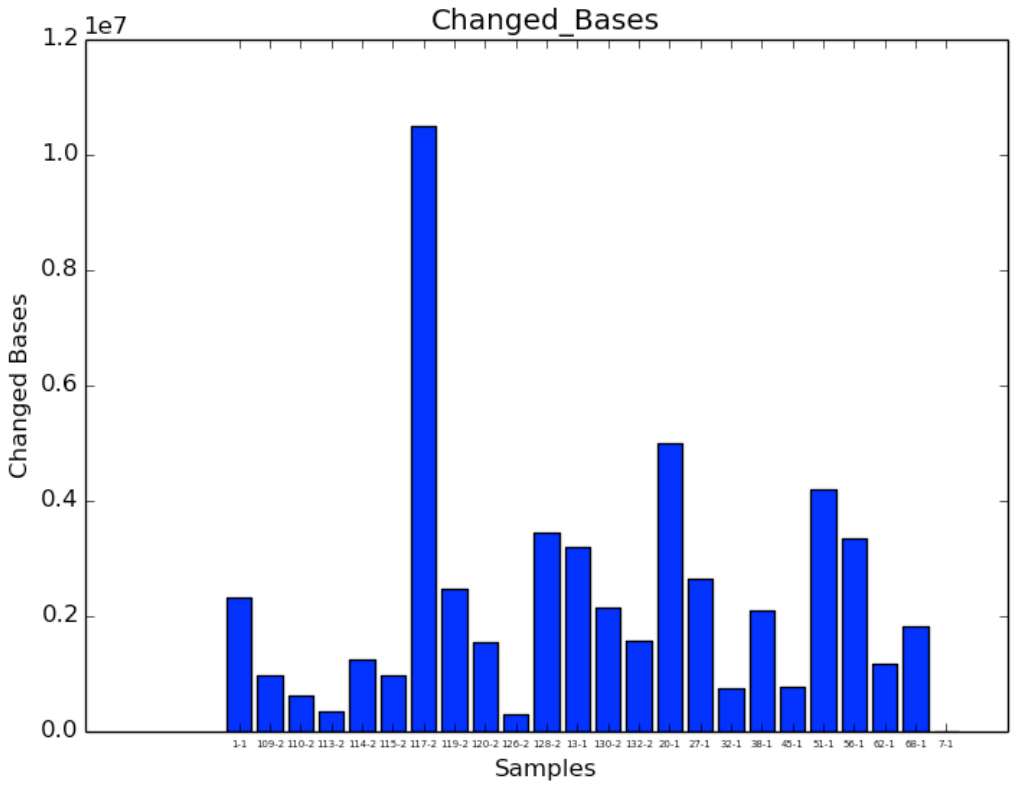

Figure 5: The Changed Bases Graph. This graph provides the user with information how many changed bases occurred during the analysis.

Following this is a graph that gives the total amount of reads from each sample that were identified as belonging to one of the identified OTUs (Figure 6). This graph's data is presented as percentages. As with the other graphs it is possible to go to the source file, in this case out_tabe.csv, and determine the actual numbers.

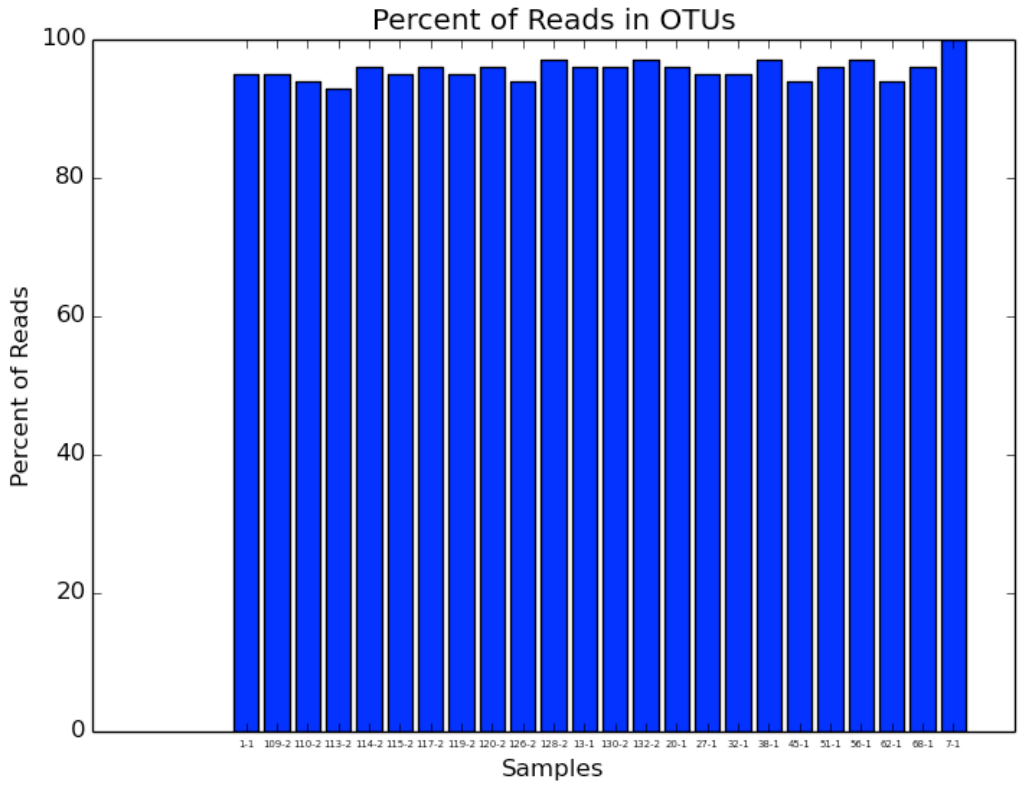

Figure 6: The Percent of Reads in OTUs Graph. This graph provides the user with information on what percentage of the reads were identified as part of an OTU.

Well over 90% of reads were identified as mapping to OTUs. This appeared to be an error at first but discussion with Dr. Ijaz determined that this was not an uncommon result. The next figure is the phylogenetic tree generated from the identified OTU data (figure 7).

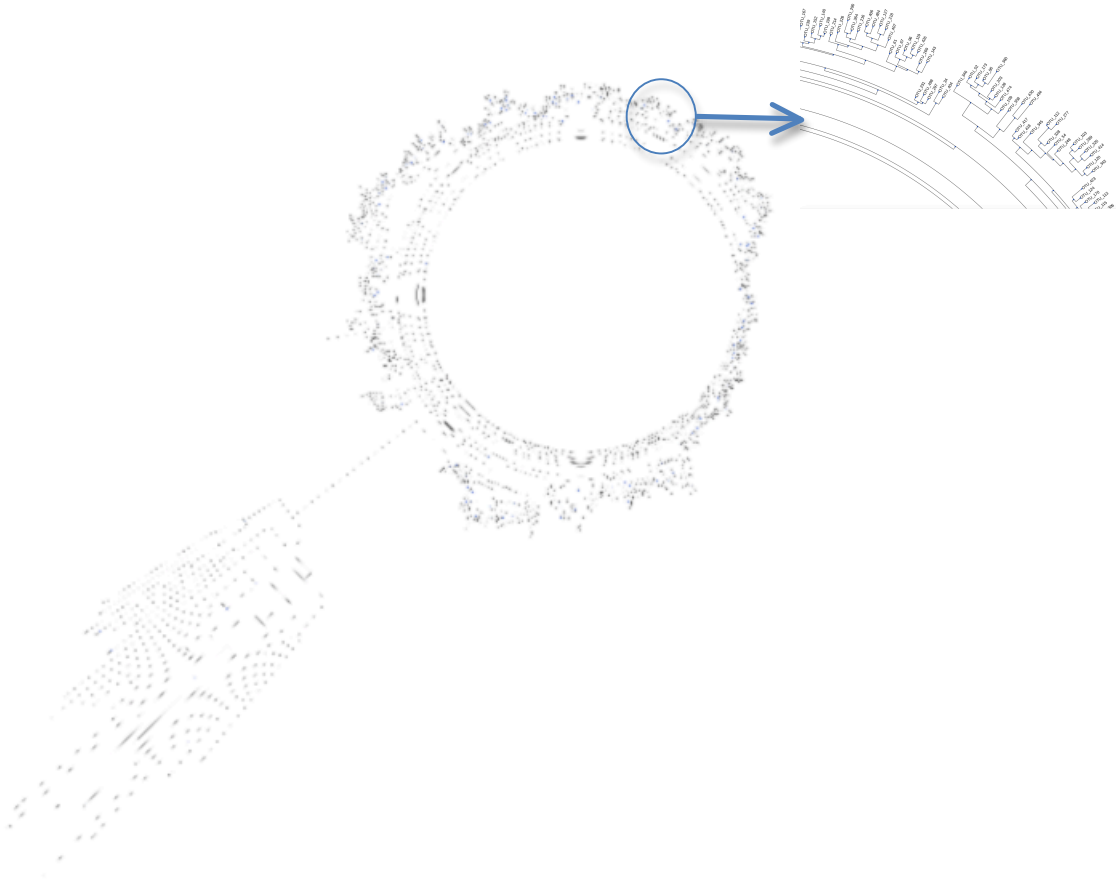

Figure 7: Phylogenetic Tree. The phylogenetic tree is generated using the OTU data. It shows the relationships between the identified OTUs.

The phylogenetic tree was by far the largest file to be output by the `AMPLIpyth` program. The output image has the resolution required to examine the phylogenetic relationships however this is not readily observable on the HTML output.

The next output displays the proportional abundance of the twenty most common OTUs identified within the entire study, for each of the samples. As previously stated, any OTU not specified in this list was lumped into an Other_Otus category (figure 8).

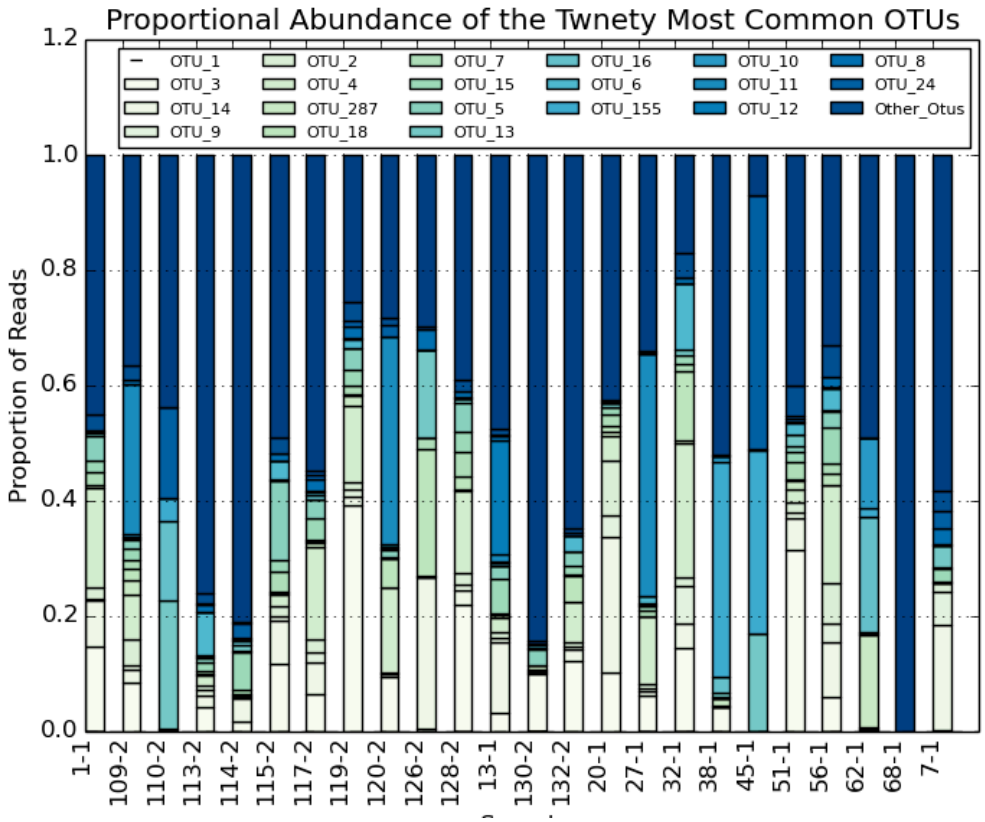

Figure 8: Proportional Abundance of the Twenty Most Common OTUs. The twenty most common OTUs identified by the study were graphed against the remained of the OTUs clustered into a single Other_Otus category.

The final output or outputs depending on the number of samples are the read distribution by length graphs. The graphs give quantitative information on the lengths of the reads after quality trimming. Two graphs are generated for pair-end reads: one for the forward read and one for the reverse read. For the sample data set of 24 samples in the test data, 48 read length distribution graphs were produced. A representative pair is shown in figure 9.

Representative Read Length Distribution Graphs

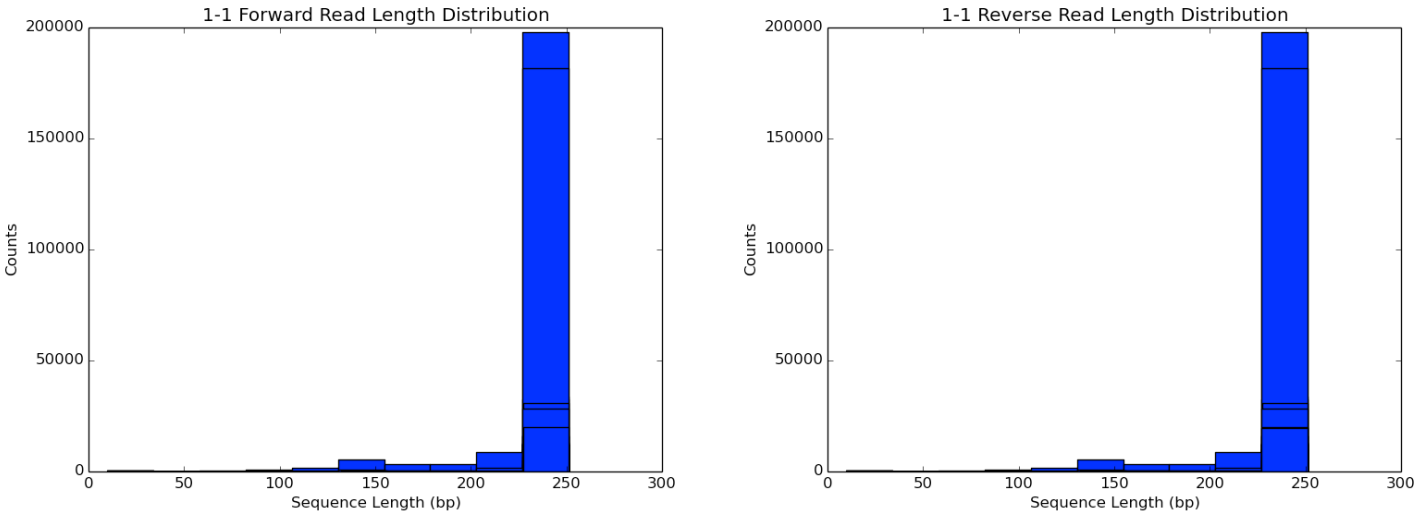

Figure 9. Representative Read Length Distribution Graphs. These graphs show the distribution of lengths observed after quality trimming.

The graphs provide valuable information on what effect quality trimming has had on the overall length of the reads. In this sample data, the majority of the reads were of high enough quality to not require trimming. This is evidenced by the majority of the reads being 250 bp.

Discussion

The goal of this project was to construct an automated pipeline for metagenomic analysis that required minimal human interaction outside the initial setup. This was successfully accomplished with the creation of the `AMPLIpyth` program. This python based program uses a configuration file to specify the files for analysis and settings, desired by the user, for the dependent tools. The configuration file is given as part of three required arguments passed to the program by the user. Once the program is initiated, no further input is required from the user. Extensive testing was done to ensure program stability and reproducibility of results. In all testing program failure was only observed twice, both a function of hitting available space walls. Results were absolutely consistent both between tests and compared to an external reference. The program successfully output he desired HTML page.

Sample data testing was covered previously, however it is important to note that what was observed here may not be observed with other data sets. The largest percentage of read loss occurred during the overlapping step but it is entirely possible that this could occur at any other step. Likewise the percentage of reads mapped to OTUs could be lower or higher. When using the sample data to evaluate this program it is vital to remember that it was used to show the program's functionality not to display the results of an experiment.

The use of the python module `ConfigParser` and a configuration file meant that the settings for the pipeline tools would need to be contained within the configuration file. Initially a loop was implemented that went through each of the name:value pairs in the settings sections and passed those arguments to the

appropriate functions. It became evident early on that there was an issue which resulted in the order of the settings not being observed by the program and the wrong settings were being called at the wrong time. To circumvent this, a single name “settings” had a value given as a string that contained all of the settings separated by underscores. This solution, while not ideal, worked and was kept during the development of the program.

A cursory examination of the source code will reveal hardcoding in several areas. As with any program, user input had to be balanced with ease of use. Many aspects of this program could have had user input but did not necessarily require it. In these areas hard coding was deferred to as the overall goal of the project was to develop a program that required minimal intervention.

The read length distribution graphs incorrectly display some of their data through the overlap of the bars on the histogram. To clarify, the data they display is not incorrect; rather the incrementation on the x-axis is not providing sufficient space to allow for all of the bars to be displayed where they should be resulting in overlap. This issue was unable to be resolved in the current iteration of the program however it will be fixed in future releases.

Conclusions/Future Work

The `AMPLIpyth` program is a ready-to-use solution for metagenomic analysis. It uses tools that are widely available and well documented. It can be easily incorporated into existing workflows without the need to substantially change other procedures. The program design was the benefit of substantial research performed previously by Dr. Ijaz and his collaborators. This led to a "good value for money" situation by which the efficacy of design could be assured. This however is a double-edged sword. The program design while backed by empirical research, is optimized for Illumina sequencing technology. For the foreseeable future this will suffice however with recent advances in third generation sequencing technology, it is very likely that the pipeline will have to be expanded to include these new technologies and the tools for analyzing their data.

The current program is only designed to use a single core which, while not a problem for the sample dataset, would result in a problem for much larger datasets. As such the next iteration of the program will include the capability to take advantage of multiple cores.

The configuration file is an integral part of the program functionality. Currently the settings are passed in the form of a string for reasons discussed previously. This is not ideal since programmatically it is more appropriate for each setting to be passed as a name:value pair. Future iterations of the program will have this implemented.

Future iterations of `AMPLIpyth` will contain changes to the HTML outputs. A proportionality graph for the changed bases data will be added. It is possible that

with some datasets the differences in read count will be so great that the resolution on a purely quantitative graph would suffer. A second proportional graph will provide additional context.

The current version of AMPLIpyth outputs a significant amount of text to the user. This is a result of the normal course of function for each of the tools being called. Future iterations of the program will have a verbose function added so that the user may suppress these outputs.

References

1 Deutschbauer, A. M., Chivian, D. and Arkin, A. P. (2006) Genomics for environmental microbiology. Curr. Opin. Biotechnol. **17**, 229–235.

2 Schloss, P. D. and Handelsman, J. (2005) Metagenomics for studying unculturable microorganisms: cutting the Gordian knot. Genome Biol. **6**, 229.

3 Simon, C. and Daniel, R. (2011) Metagenomic analyses: Past and future trends. Appl. Environ. Microbiol. **77**, 1153–1161.

4 Mardis, E. R. (2008) Next-generation DNA sequencing methods. Annu. Rev. Genomics Hum. Genet. **9**, 387–402.

5 Handelsman, J. (2004) Metagenomics : Application of Genomics to Uncultured Microorganisms Metagenomics : Application of Genomics to Uncultured Microorganisms. Microbiol. Mol. Biol. Rev. **68**, 669–685.

6 Bik, H. M., Porazinska, D. L., Creer, S., Caporaso, J. G., Knight, R. and Thomas, W. K. (2012) Sequencing our way towards understanding global eukaryotic biodiversity. Trends Ecol. Evol., Elsevier Ltd **27**, 233–243.

7 Edgar, R. C. (2010) Search and clustering orders of magnitude faster than BLAST. Bioinformatics **26**, 2460–2461.

8 Schloss, P. D., Westcott, S. L., Ryabin, T., Hall, J. R., Hartmann, M., Hollister, E. B., Lesniewski, R. a., Oakley, B. B., Parks, D. H., Robinson, C. J., et al. (2009) Introducing mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities. Appl. Environ. Microbiol. **75**, 7537–7541.

9 Caporaso, J. G., Kuczynski, J., Stombaugh, J., Bittinger, K., Bushman, F. D., Costello, E. K., Fierer, N., Peña, A. G., Goodrich, J. K., Gordon, J. I., et al. (2010) QIIME allows analysis of high- throughput community sequencing data Intensity normalization improves color calling in SOLiD sequencing. Nat. Publ. Gr., Nature Publishing Group **7**, 335–336.

10 QIIME Development Team. (2015) QIIME.org.

11 Schloss, P. and Deparment of Microbiology & Immunology at The University of Michigan. (2015) mothur.org.

12 Schirmer, M., Ijaz, U. Z., D'Amore, R., Hall, N., Sloan, W. T. and Quince, C. (2015) Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform. Nucleic Acids Res 1–16.

13 Thayer, A. M. (2014, August) Next-Gen Sequencing Is A Numbers Game. Chem. Eng. News.

14 Joshi, N. and Fass, J. (2011) Sickle: A sliding-window, adaptive, quality-based trimming tool for FastQ files.

15 Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. a., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., et al. (2012) SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. J. Comput. Biol. **19**, 455–477.

16 Nikolenko, S. I., Korobeynikov, A. I. and Alekseyev, M. a. (2013) BayesHammer: Bayesian clustering for error correction in single-cell sequencing. BMC Genomics, BioMed Central Ltd **14 Suppl 1**, S7.

17 Masella, A. P., Bartram, A. K., Truszkowski, J. M., Brown, D. G. and Neufeld, J. D. (2012) PANDAseq: paired-end assembler for illumina sequences. BMC Bioinformatics, BioMed Central Ltd **13**, 31.

18 Katoh, K., Misawa, K., Kuma, K. and Miyata, T. (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. Nucleic Acids Res. **30**, 3059–3066.

19 Katoh, K. and Standley, D. M. (2013) MAFFT multiple sequence alignment software version 7: Improvements in performance and usability. Mol. Biol. Evol. **30**, 772–780.

20 Price, M. N., Dehal, P. S. and Arkin, A. P. (2009) Fasttree: Computing large minimum evolution trees with profiles instead of a distance matrix. Mol. Biol. Evol. **26**, 1641–1650.

21 Wang, Q., Garrity, G. M., Tiedje, J. M. and Cole, J. R. (2007) Naïve Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. Appl. Environ. Microbiol. **73**, 5261–5267.

22 JetBrains s.r.o. (2015) PyCharm Professional Edition, JetBrains s.r.o.

23 Ewing, B. and Green, P. (1998) Base-calling of automated sequencer traces using phred. II. Error probabilities. Genome Res. **8**, 186–94.