16S RNA Sequencing Data Management using SQLite

Xu Junjie, Kevin

June 2014

Abstract

The 16S Ribosomal RNA Sequencing is used extensively in analzying bacterial phylogeny and taxonomy. This project attempts to streamline the 16S sequencing pipeline using local file databases to replace multiple flat sequence files used in the pipeline, to ease logistical burdens on the researcher and enable greater metadata analysis and accountability of experiments.

Contents

1 Introduction

The project utilized computational methods such as 16S Ribosomal Profiling as a proxy for species identity. The 16S small subunit of bacterial ribosomes are highly conserved, which means that the differences within the 16S RNA profiles can be used as an analogue for species identity. Since the 16S gene contains both highly conserved and highly variable regions all interspersed together, The highly conserved regions can be amplified using PCR (Polymerase chain reaction), while the variable regions are used to classify the organism. HTS (High Throughput Sequencing) using Illumina sequencers are then used to sequence the PCR products from the prior step.

The output from HTS forms the start of the computational pipeline. In the preprocessing stage, poor quality reads are filtered and trimmed. Chimeras and non-bacterial 16S reads are then removed. The pipeline then attempts to produce phylogenetic classification through the use of RDP classifiers and sequence similarity (i.e. OTU Analysis). By comparing the filtered output from HTS against the already existing taxonomies of over 2 million species, the genus, family, and order of the sample can be determined.

The RDP classifier can fail for various reasons: the majority of bacterial species have not been identified or sequenced, the 16S reads are usually too short for accurate classification. In those cases, groups of highly similar sequences are grouped into OTUs (Operational Taxonomic Unit) that can be analyzed for quantitative differences in communities between the sequenced samples.

The project utilizes a computational approach due to the large amount of raw sequencing data that is created from the PCR amplification process. The variability of results is further affected by the OTU Analysis stage, where prior parameters can affect the analysis outcome. Using a computational approach would allow the change of various parameters, in order to quantitate the effects of those parameter changes.

The majority of the pipeline is executed on the University of Oregons ACISS High-Performance Supercomputer Cluster, and utilizes PBS scripts (normal shell scripts with extra variables defined to manage job resources) to execute the different stages of the pipeline.

The FASTQ output produced by the Illumina sequencer is run through the preprocessing stage of filtering, trimming, and demultiplexing. The PBS script for the preprocessing stage calls on the Demultiplexer, a python script written by Rodger Voelker, which removes the primer attached to the sequences during the amplification process, and attaches barcodes (signifying sample origin) to both ends of the paired-end reads in the FASTQ file. After demultiplexing, the pipeline proceeds to use Trimmomatic v0.32, an open

source tool to trim poor quality feeds from the reads. It uses a sliding window trimming, that cuts out sequences when the average quality within a window falls below a certain threshold. Finally, Bowtie 2.1.0 is used to align the reads to existing mitochondrial and phiX sequences and to remove them.

2 Data Management

This Talk about the db schema, rationale. The database is initialized with 4 tables:

TODO - NEED TO CONSULT JC for clarification on db structure * primer - holds all the primers used in experiment * offset - randomized offsets to aid 16S sequencing * barcode - experiment identifier * log - holds logging data to aid accountability in experiments

The log table supports the logging of metadata and a record of all operations done on the database. How to describe metadata logging? What sort of metadata are we planning to capture?

General logging can be described in accountability terms - e.g. time of operation, tables operated on, number of rows operated, who (email? username?) operated. Purpose of this table is to automatically record operations so experiments can be reproduced with the appropriate settings/parameters easily.

During the process of importing FASTQ data into the database, another table is produced:

* reads1,reads2 - holds paired-end FASTQ data * reads - holds single-end FASTQ data

2.1 Compression

Short description of the compression scheme (combining sequence and quality lines) and file size savings

Majority of space usage within sequence/quality lines:

```
@HWI-ST0747:277:D1M96ACXX:6:1101:1232:2090 1:N:0:
GGATAGTACTAGGGTATCTAATCCTGTTTGCTCCCCACGCTTTCGCACCTCAGCGTCAGTATCGAGCCAGTGAGCCGCCTTC
+
ACCCFFDDFH#FHHIGIJJFJJJJJJJJJJJIIJJJGIJJGIJJJIGIJJJIIIIJJJJHHHHHFFFDCEACCDDCDDD@BD

ATCGN #$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJ
```

* 8-bit char can hold 256 (28) values * 5 bases require 4 (22) to 8 (23) values * 42 qualities require 32 (25) to 64 (26) values

A: 0-49 T: 50-99 C: 100-149 G: 150-199 N: 200-255

* 50* Fast compression/decompression * Lossless

3 Data Flow using UNIX Pipes

A central part of managing the sequence data in the 16S pipeline is the manipulation and transferring of sequences between the various tools. There

Tool	Expected input format
Trimmomatic	FASTQ/Gzipped FASTQ
Bowtie2	FASTQ/FASTA
QIIME	454-FASTA/454-Quality Scores

Table 1: Expected input formats for various tools

is often a need to modify the structure or format of the data to fit the varying input requirements of those tools. Table 1 shows the various input formats that tools in the pipeline expect.

Traditionally, the way to manage the various formats was to transform and store various copies of the data in multiple files. For example, running a sequence through just the first three tools in the pipeline will produce the following files:

* Streaming Data with Named Pipes * Tool-specific adapters * Demultiplexer, Trimmomatic, Bowtie * No intermediate files Virtual files on filesystem

1. Request data 2. Transforms data into streams of FASTQ data 3. Feed data into tool using pipes 4. Receive feedback from tool 5. Transform feedback into deltas 6. Store data into SQLite

* No intermediate files * No disk I/O

4 Benchmarks

Show insert speed, normalization speed, streaming speed (to trimmomatic) Insert speeds: 500K Paired-end: Pip: 500000 sequences imported in 5.19 sec-

onds Pip: 500000 sequences imported in 5.06 seconds Merged 2 files in 10.35 seconds ¡ actual inserts Pip: normalizing the database... Pip: normalization complete in 3.06718 seconds Pip: vacuuming the database to recover free space... Pip: vacuum complete in 3.07635 seconds Pip: Normalized database in 6.14 seconds ¡ time taken for database normalization and space vacuum (recovery)

1M paired-end: ip: 1000000 sequences imported in 9.74 seconds Pip: 1000000 sequences imported in 9.51 seconds Merged 2 files in 19.57 seconds

2M paired-end: Pip: 2000000 sequences imported in 19.78 seconds Pip: 2000000 sequences imported in 19.00 seconds Merged 2 files in 39.22 seconds

4M paired-end: Pip: 4000000 sequences imported in 39.10 seconds Pip: 4000000 sequences imported in 38.82 seconds Merged 2 files in 78.67 seconds

8M paired-end: Pip: 8000000 sequences imported in 81.01 seconds Pip: 8000000 sequences imported in 78.75 seconds Merged 2 files in 160.84 seconds

16M paired-end: Pip: 16000000 sequences imported in 159.81 seconds Pip: 16000000 sequences imported in 157.41 seconds Merged 2 files in 319.38 seconds

32M paired-end: Pip: 32000000 sequences imported in 321.50 seconds Pip: 32000000 sequences imported in 320.16 seconds Merged 2 files in 644.90 seconds

64M paired-end: Pip: 64000000 sequences imported in 644.42 seconds Pip: 64000000 sequences imported in 637.76 seconds Merged 2 files in 1286.26 seconds

5 Conclusion