Snakemake workflows for metagenomics data

Student Name: Kai Liu
Student ID: 2397780
Supervisor: Umer Zeeshan Ijaz

August 2020

A thesis submitted in partial fulfilment of the requirements for the degree of
MASTER OF SCIENCE IN COMPUTER SYSTEM ENGINEERTING

Abstract

With the continuous development of life sciences and metagenomics, the research on microbial communities is also continuously strengthened. The new generation of genome sequencing technology has brought about new changes in the analysis of microbial communities. These new technologies can quickly and accurately analyze and sequence the genomes of microbial communities. RNA sequencing (also called RNA-Seq) is the one of these technologies, which uses high throughput techniques to transcript the sequences. In the traditional data analysis process, these techniques are divided into multiple steps by the researcher. The researcher wrote a script for each step to connect. There is a lot of repetitive work in this process, which is not only time-consuming, but also may produce wrong results due to errors between the steps. The workflow based on Snakemake can greatly improve the efficiency of data analysis, reduce repetitive labor, and make analysis data easier to manage. The essence of Snakemake workflow is a python script that can be written. Snakemake compiles each analysis step into a corresponding rule, and uses shell commands for input and output. This project used 24 samples of fecal which was provided by Dr. Umer.

Acknowledgment

I would like to express my thanks towards my supervisor Umer Zeeshan Ijaz, for giving me the opportunity to study this incredibly interesting new subject. I would also like to thank him for the many meetings we had and the guidance he provided on numerous occasions.

I would also like to take this opportunity to thank my fellow Masters students, who were always around whenever I needed a helping hand.

Abbreviations and Definitions

NGS = Next Generation Sequencing

PCR = Polymerase chain reaction

OTU = Operational Taxonomic Unit

RNA = Ribonucleic Acid

SD = Shine-Dalgarno sequence

IDE = Integrated Development Environment

dNTP = deoxynucleoside triphosphate

ddNTP = dideoxynucleoside triphosphate

Contents

1. Introduction

Metagenomics is a new method of studying microbial diversity which is developed on the basis of genomics [1]. Metagenomics directly extracts the DNA of all microorganisms from the ecological environment and constructs a metagenomic database to study the genetic information composition and microbial community structure of all microorganisms contained in the sample environment.

Microbial community refers to a structural unit formed by various microorganisms in a specific ecological environment. Microbial communities are widely present in ecosystems, and they are an important part of ecosystems. Different populations in the microbial community can co-exist in an orderly manner [2]. In a sense, it can be said that the development of microbial communities led to the development of organisms.

Most of current research on microbial communities is based on traditional microbiology subjects. These subjects contain biochemistry, molecular biology, and genetics. Some studies have shown that the number of cultured microorganisms may be less than 1% of the total number of microorganisms in nature environment, which means that the study of microbial communities has full prospects.

Traditional microbial research methods are mainly based on selective cultivation or identification and classification based on metabolic characteristics, morphological characteristics and antigen characteristics [3]. The disadvantage of this method is that some bacteria are small in number and difficult to cultivate, or the existing medium and culture technology are not suitable for microbial cultivation, or some bacteria grow extremely slowly. Unknown bacteria cannot be identified also is the disadvantage. These

disadvantages will cause the number and diversity of the normal flora to be greatly underestimated.

First generation sequencing, also called Sanger sequencing, is a sequencing technology that uses DNA polymerase synthesis reactions [4]. Under stable and appropriate situation, DNA polymerase can catalyze the synthesis of DNA strands. This process requires DNA templates, primers and deoxynucleoside triphosphates(dNTP). ddNTP is a dideoxynucleoside triphosphate. In this dideoxynucleoside triphosphate the C3 position is connected to a hydroxyl group after deoxygenation [5]. In general, the -OH on the C3 position is used as the next dNTP connection site. Therefore, the ddNTP that has lost its oxygen atom cannot be connected to the next dNTP, thereby terminating the DNA strand extension. Label the dideoxynucleoside triphosphate with a radioisotope and add the marked dideoxynucleoside triphosphate to the regular PCR reaction. When dideoxynucleoside triphosphate binds, DNA synthesis will be aborted. After dozens of periods, DNA with different lengths and one base difference in length will be obtained [6]. The obtained products are divided into four lanes for polyacrylamide gel electrophoresis. Finally, DNA sequence is deduced backward based on the band positions of the four bases.

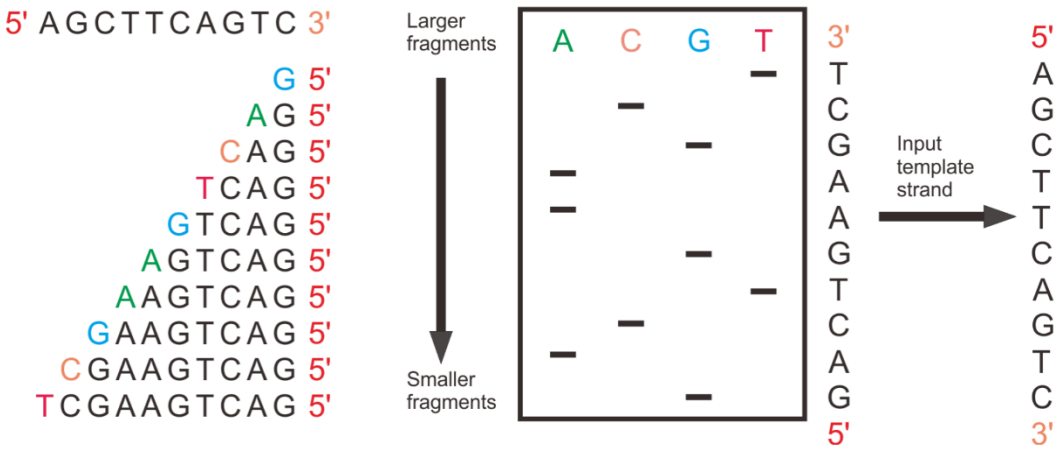

Figure 1: Diagram of Sanger sequencing

With the continuous development of Metagenomics, the first generation sequencing technology gradually cannot meet the huge number of sequencing needs. Second generation sequencing technology, which is also called as high throughput sequencing technology [7]. It solves the defect that the first-generation sequencing can only measure one sequence. With the continuous deepening of scientific research, researchers began to analyze all sequence information in a species or sample. At this time, the method of first-generation sequencing cannot meet the sequencing needs. The second-generation sequencing technology was born under such circumstances. It is called high-throughput sequencing because it can measure many sequences at the same time. Researchers randomly break the DNA into countless small fragments by using some physical or chemical methods. Then they enrich these DNA fragments by constructing a reference base [8]. In the Illumina sequencing platform, this enrichment process is bridge amplification.

Bridge PCR uses the adapter fixed on the surface of the Flowcell as a template for bridge amplification. After continuous cycles of amplification and denaturation, each DNA fragment will eventually be concentrated into a bundle at its own location, and each bundle contains many copies of a single DNA template. Next, put the completed library into a sequencer for sequencing. The sequencing machine has a unique area in which DNA fragments can be attached together. Each fragment has an independent connection space, so the sequencer can simultaneously detect all additional DNA sequence information. Finally, by using bioinformatics analysis technology, small pieces of fragments are spliced into larger pieces [9].

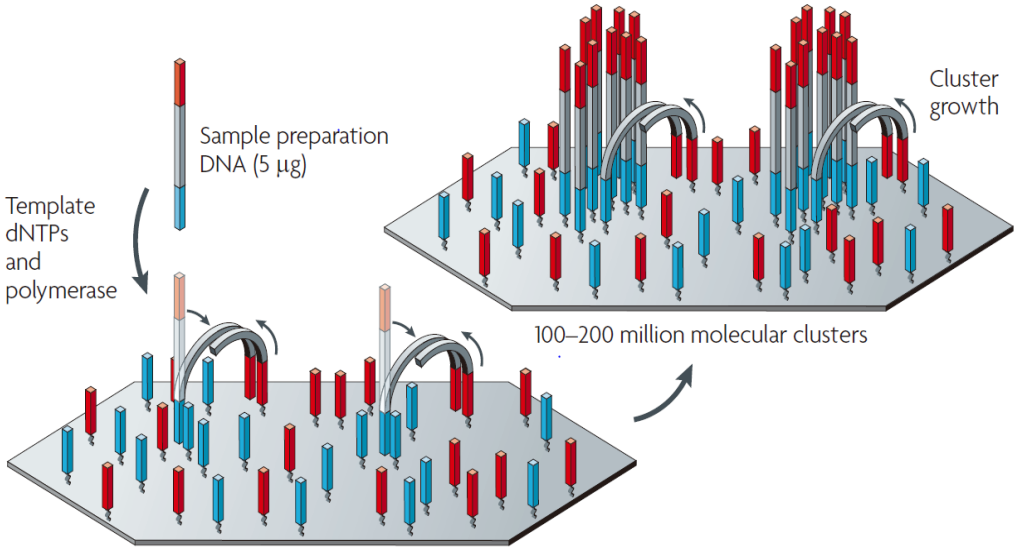

Figure 2: Bridge amplification

This thesis studied how the Snakemake workflow was used for metagenomics, and useed 16S RNA sequences as a sample to illustrate the workflow. 16S ribosomal rRNA, or 16S RNA, is a part of the 30S small subunit of prokaryotic ribosomes and binds to Shine-Dalgarno (SD) sequence, which has approximately 1500 bases [10]. 16S RNA is widely distributed in prokaryotes, which can provide sufficient information and has a relatively slow evolutionary process. The 16S rRNA gene region contains both conserved sequences and variable sequences. While reflecting the genetic relationship of biological species, it also reveals the characteristic accounting sequence of biological species [11]. Therefore, it is used to identify the species of prokaryotes. This thesis useed the sequences supported by Dr.Umer Zeeshan Ijaz from cluster MScBioinf@becker.eng.gla.ac.uk.

In the traditional transcriptome data analysis process, the analysis is generally divided into multiple steps, and each step contains one or more analysis software. Researchers need to connect various steps through programming scripts, and there will be a large number of repetitive data

analysis operations during the analysis process, which not only requires certain computer skills for researchers, but also consumes a lot of time and energy in repetitive. At present, the amount of sequencing data is growing rapidly. In the era of big data, more powerful transcriptome data analysis tools are needed [12]. Therefore, some sequencing tools such as QIIME and VSEARCH workflow have been developed. However, these tools are not without their shortcomings. Most of them require frequent user intervention (manually enter the command line to control the analysis process). In order to reduce repetitive work and improve the efficiency and automation of data analysis, an analysis framework based on Snakemake was developed.

2. Analysis

2.1 Environment

Snakemake is a process management tool written based on Python. Snakemake simplifies each step into a rule. It uses shell commands or Python code to input and output files. It can also use Python language to manage the software running process. Using Snakemake to build transcriptome analysis can improve the efficiency of transcription component data analysis. Snakemake workflow can greatly improve the efficiency of transcriptome data analysis, and is easy to use, providing convenience for researchers with non-computer background [13]. The program of Snakemake workflow in the thesis is based on python version `Python 2.6.6`.

In order to run Snakemake correctly, the first thing is to confirm the operating system. If using Linux or MacOS X operating, there is no need to change operating system. If the operating system is Windows, users need to set up a Linux virtual machine (VM). The programming of this thesis was all done using MobaXterm under Windows environment. In the process of building the Snakemake framework, users needed to install a lot of software related to transcriptome analysis. This framework used Conda to automatically download and install the required software, avoiding the time-consuming manual download and installation of the software, and the software version error. Conda can also configure related dependencies to complete a series of operations such as creating, saving, loading and switching the operating environment in the local computer system. In terms of framework parameters, software parameters, file paths and other settings, the framework is uniformly set through the configuration file `config.yaml`, and there is no need to repeatedly set the same parameters. At the same time, users can also set up and manage the framework through the configuration file, and customize the

transcriptome analysis mode. In terms of software selection, the relevant software for transcriptome analysis was studied and compared, and mainstream software with better performance, wide application, and strong stability was selected to be used in the framework.

2.2 Tools

Vim

The program was coded in Vim on Dr.Umer's cluster. Vim is a lightweight text editor. It has a wealth of functions to meet the needs of developers, such as code completion, compilation and error prompts. It is widely used among programmers. Compared with other common IDEs (such as pycharm, vscode), Vim has a steeper learning curve. However, as a lightweight text editor, Vim has many advantages. Vim can adapt well to various hardware environments, can automate and script tasks, can remotely debug and modify code, can quickly support new languages, and can quickly adopt new development models.

sickle

`Sickle` is a kind of sequence quality trimming tool. Sickle uses a sliding window and quality and length thresholds to determine where to trim the reads. When the sickle starts to get the quality value, the window starts to slide, and the length is 0.1 times the reads. If the length of reads is less than 1, the sliding window is set to the length of reads. Otherwise, this window slides along the quality value until it rises above the threshold and trims at that position. When the quality drops below the threshold, trim again at that position. Finally, the sequence after quality trimming is obtained [14].

`Sickle` can be regarded as a filter, generating high-quality reads for use by other downstream analysis tools. According to `sickle` documentation https://github.com/najoshi/sickle, `sickle` supports three types of quality values: Illumina, Solexa and Sanger.

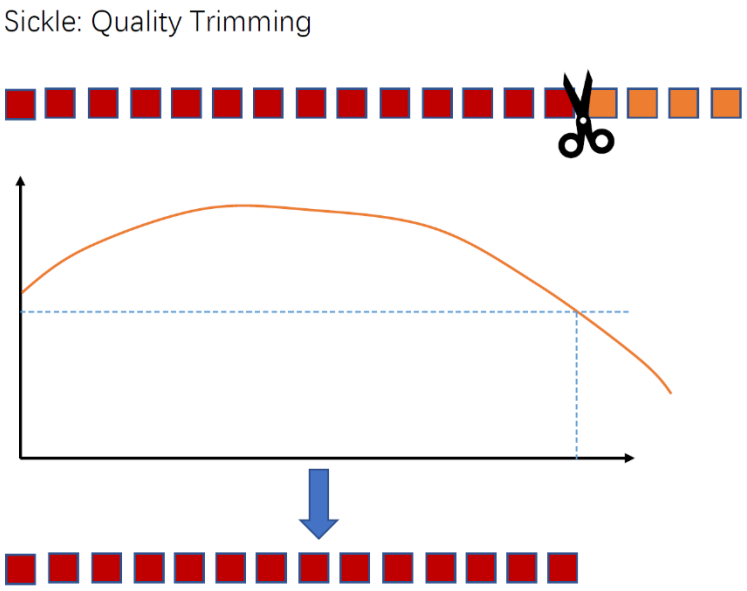

Figure 3: Diagram of Sickle

SPAdes

Due to the complexity of the natural environment and the diversity of biology, a large part of bacteria is difficult to clone and cultivate in the laboratory, making it impossible to sequence these bacteria using existing technologies. `SPAdes` is a useful open-source assembler program, it has several kinds of assembly pipeline toolkits. `SPAdes` can be used for single bacteria data and multiple bacteria data to generate easy-to-operate compilation bacteria data. `SPAdes` can run in a Linux environment and cooperate with the `BayesHammer` module to perform only error correction. After using `SPAdes` assembler program the system will create a corrected folder for each sequence. These folders will contain the error-corrected reads of each paired-

end sequence [15].

VSEARCH

VSEARCH is an open source and free multi-threaded 64-bit assembly tool. VSEARCH is widely used in metagenomics, genomics and population genomics to process nucleotide sequence data [16]. The VSEARCH program adopts a complete dynamic programming strategy and uses multi-threaded parallel computing to improve the speed and accuracy of identifying similar nucleotide sequences. VSEARCH has a variety of methods for processing nucleotide sequences, such as precision-based search, paired alignment, arrangement supplementation, sorting sampling and normalization processing. The VSEARCH version used in this project was `vsearch v2.10.4_linux_x86_64`.

PANDAseq

The simplest method of sequencing is single-end sequencing, that is, there is only one sequencing primer, so that PCR can only be performed along the direction of this primer, and all reads can only be read in one direction. However, this sequencing method has some limitations. Taking Illumina as an example, the quality of sequencing will decrease with the progress of sequencing, so the reads will be less accurate in the future. The solution that researchers came up with was paired-end sequencing. For a 500 bp sequence, the downstream quality of single-end sequencing will be poor [17]. But by measuring 250 bp-300 bp from two directions and then splicing them together, the accuracy of sequencing can be greatly improved.

`PANDseq` can be used to do the paired-end sequencing. The linker sequence of paired-end sequencing is more complicated than single-end sequencing. First, in order to perform sequencing in two directions separately, two sequencing primers in different directions are required. Secondly, in order to distinguish the reads in the two directions, a small index sequence is added in front of one of the sequencing primers to mark. The length of each read in paired-end sequencing is more than half of the entire sequence, so it can be spliced according to the overlapping part of the two Reads.

3. Program Design

This Snakemake workflow project referred to Dr. Umer's tutorial on `VSEARCH` workflow. Figure 4 showed the overview of the Snakemake pipeline. Before designing the program, configure the environment of Snakemake, and entered the command `source activate snakemake` to activate the working environment. Then entered the calling commands of various tools such as `Sickle`, `VSEARCH` and `PANDAseq` to ensure these tools could be used normally on the system. The next step was creating a Snakemake workflow configuration file by entering `touch Snakefile`. The path where this Snakefile located was the directory where the workflow program generated output files. Then created a folder to store the original input files under the current path. The folder created in this project was named samples, which was used to store 24 pairs sequences. Every pair of the sequence contained a forward read sequence and a reverse read sequence.

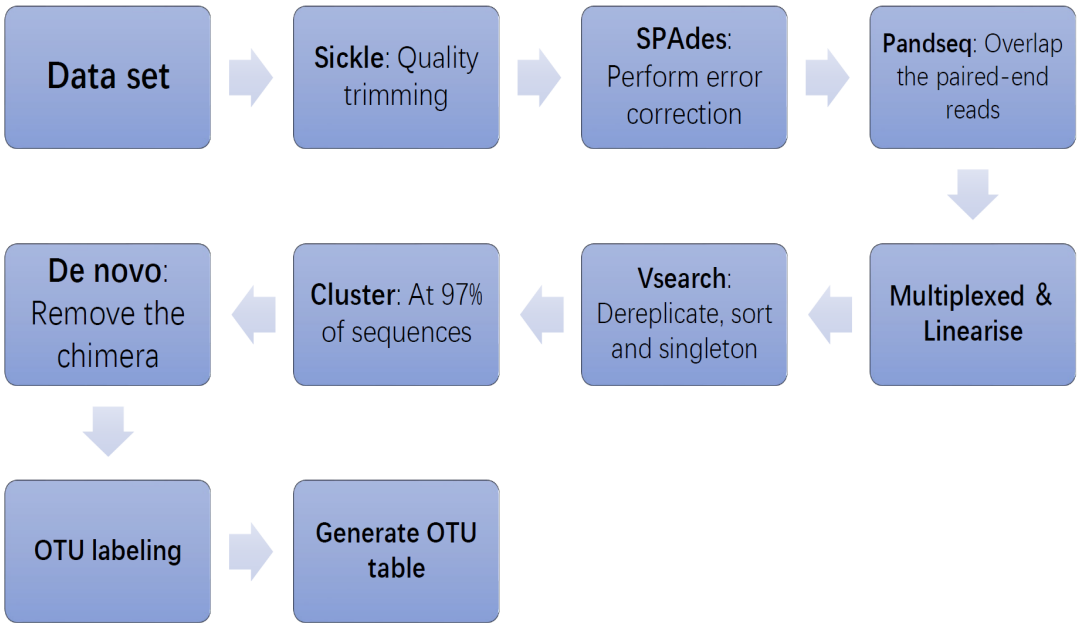

Figure 4: Snakemake workflow overview

First, the input and output of the program needed to be determined. According

to the requirements of the project, the input of the program was the nucleotide sequences in FASTQ format, and the output is the integrated OTU (operational taxonomic unit) FASTQ file and OTU table. In this project all the nucleotide sequences in FASTQ format were support by Dr.Umer's cluster MScBioinf@becker.eng.gla.ac.uk [18]. Since Snakemake has an advantage that it can process multiple data in batches and user do not need to set parameters multiple times, wildcards can be used as input. The most basic and important file in microbial diversity analysis is OTU table. Almost all subsequent analyses, such as alpha diversity analysis, beta diversity analysis, difference analysis, etc., were based on OTU table. The OTU table contains the OTU types and sequence numbers of all samples. The information of all OTUs' species annotation can also be found in OUT table.

```
SAMPLES, = glob_wildcards("samples/{sample}_R1_001.fastq")
NB_SAMPLES = len(SAMPLES)

for smp in SAMPLES:
    message("Sample " + smp + " will be processed")

rule final:
    input:
        "all.otutab.txt"
```

Figure 5: Wildcard and OTU table

The FASTQ file is a kind of file based on text-based format which is used to record nucleotide sequences and corresponding quality scores. The FASTQ file usually consists of 4 lines of ASCII characters to indicate the information of the sequence, of which the fourth line records the quality score of the sequence.

Since the input is a FASTQ file, each sequence has a specific quality score. The quality score can be considered as the correct rate of calling sequences during the sequencing process. The quality score is usually adapted form 0 to 40.

Table 1 indicated the mapping relationship between quality score and probability of incorrect base call [19].

QUALITY SCORE	PROBABILITY OF INCORRECT BASE CALL	BASE CALL ACCURACY
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10000	99.99%
50	1 in 100000	99.999%
60	1 in 1000000	99.9999%

Table 1: mapping relationship between quality score and probability of incorrect base call.

The following formula shows how to determine the quality value of the base:

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

Equation: The formula of calculating quality score

The quality of reads produced by most sequencing programs might drop at the 3'and 5'ends. If these sequences were not trimmed, the bases may be paired in the wrong place. This might lead to errors in downstream analysis.

After inputting all the sequences as FASTQ file, quality trimming should be

performed by using sickle. A `sickle` command called `sickle pe` has been used in this program. This command could take both forward and reverse paired-end FASTQ files as input like "`sample.fastq`", and then outputted two trimmed paired-end FASTQ files like "`sample_trim.fastq`" and a single read file like "`sample_singlet.fastq`". This step was shown as figure 6.

```
rule sickle:
    input:
        forward="samples/{sample}_R1_001.fastq",
        reverse="samples/{sample}_R2_001.fastq"
    output:
        forward_trim="samples/{sample}_R1_001_trim.fastq",
        reverse_trim="samples/{sample}_R2_001_trim.fastq",
        singlet="samples/{sample}_R2_001_singlet.fastq"
    shell:
        """sickle pe -f {input.forward} \
        -r {input.reverse} \
        -t 'sanger' \
        -o {output.forward_trim} \
        -p {output.reverse_trim} \
        -s {output.singlet} \
        -q 20 -l 10
        """
```

Figure 6: Sickle

In this rule, the command `sickle pe` meant using paired-end trimming technology, `-f` meant the one input of this rule is forward reads FASTQ file, `-r` meant the other input is reverse reads FASTQ file. `-o` and `-p` indicated the output of this rule were trimmed paired-end FASTQ file. `-s` represented the other output is singleton FASTQ file. `-q` represented the quality threshold and `-l` represented the length threshold.

After quality trimming by sickle, the trimmed FASTQ files were transferred to `SPAdes`. The rule of `SPAdes` was shown as figure 7. `SPAdes` program cooperated with the `BayesHammer` module to perform only error correction and generate a folder named corrected for each sequence. This folder

contained the error corrected FASTQ files and unpaired FASTQ files.

```
rule spades:
    input:
        r1="samples/{sample}_R1_001_trim.fastq",
        r2="samples/{sample}_R2_001_trim.fastq"
    output:
        o=directory("samples/{sample}_corrected")
    shell:
        """
        spades.py -1 {input.r1} -2 {input.r2} \
        --only-error-correction \
        --careful -o {output.o}
        """
```

Figure 7: Rule SPAdes

The next step was to use `PANDAseq` to compare all the sequences in the corrected folder and reconstruct an overlapping sequence. A schematic diagram of `PANDseq` reconstructing overlapping sequences was shown in Figure 8. First, it aligned forward reads and reverse reads, then identified the best overlap, and finally reconstructed the complete overlap sequence. This part could be design as a rule shown in figure 9.

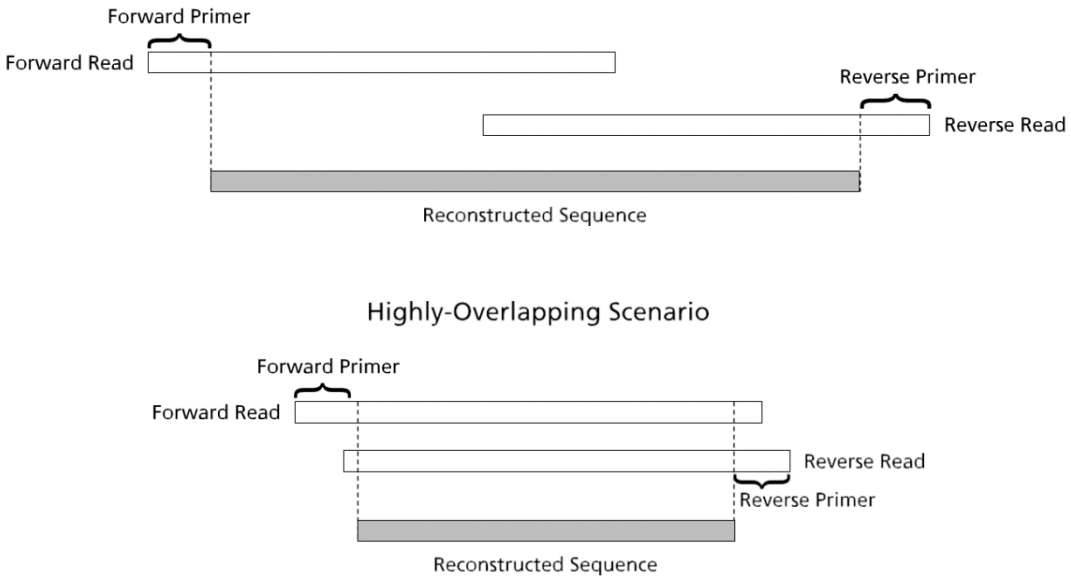

Figure 8: Diagram of PANDseq

```
rule pandaseq:
    input:
        l=directory("samples/{sample}_corrected")
    output:
        o="samples/{sample}_overlap.fasta"
    shell:
        """
        pandaseq -f {input.l}/corrected/*_R1*fastq.gz \
        -r {input.l}/corrected/*_R2*fastq.gz > {output.o}
        """
```

Figure 9: Rule PANDseq

Then users can integrate all the overlapped sequences to generate a multiplexed FASTQ file. Figure 10 shows the rule design of multiplexed.

```
rule merge_all:
    input:
        i=expand("samples/{smp}_overlap.fasta", smp=SAMPLES)
    output:
        m="multiplexed.fasta"
    shell:
        """
        (for i in $(echo {input.i}); \
        do awk -v k=$i '/^>/{{$0=">barcodelabel="k";\
        S"(++i)}}1' $i; done) > multiplexed.fasta
        """
```

Figure 10: Rule Multiplexed

This multiplexed FASTQ file contains all overlap sequences, which makes the sequence information easier to distinguish. For example, figure 11 shows a part of multiplexed file. In this example the sequence comes from the second sequence of "`109-2_S109_L001_overlap.fasta`".

```
>barcodelabel=samples/109-2_S109_L001_overlap.fasta;S2
TACGTAGGGTGCAAGCGTTATCCGGAATTATTGGGCGTAAAGGGCTCGTAGGCGGTTCGTCGCGTCCGGTGTGAAAGTCC
ATCGCTTAACGGTGGATCCGCGCCGGGTACGGGCGGGCTTGAGTGCGGTAGGGGAGACTGGAATTCCCGGTGTAACGGTG
GAATGTGTAGATATCGGGAAGAACACCAATGGCGAAGGCAGGTCTCTGGGCCGTTACTGACGCTGAGGAGCGAAAGCGTG
GGGAGCGAACAG
```

Figure 11: Partial multiplexed sequence

In order to make sure the downstream `VSEARCH` program can run normally,

multiplexed file needs to be linearized, dereplicated and sorted. Then use singleton to remove any sequence that the size is 1. These processes can be designed as figure 12.

```
rule linearize_fasta:
    input:
        i="multiplexed.fasta"
    output:
        o="multiplexed_linearized.fasta"
    shell:
        """
        awk 'NR==1 {{print ; next}} \
        {{printf /^>/ ? "\\n"$0"\\n" : $1}} \
        END {{print}}' {input.i} > {output.o}
        """

rule dereplicate_sort_singleton:
    input:
        i="multiplexed_linearized.fasta"
    output:
        uc="multiplexed_linearized_dereplicated_vsearch_min2.uc",
        fasta="multiplexed_linearized_dereplicated_vsearch_min2.fasta"
    shell:
        """
        vsearch --threads 20 \
        --derep_fulllength {input.i} --minuniquesize 2 \
        --sizein --sizeout --fasta_width 0 --uc {output.uc} \
        --output {output.fasta}
        """
```

Figure 12: Rule VSEARCH

The next step is to cluster the generated sequences at 97% by the VSEARCH program. Clustering at 97% is to facilitate the analysis of the similarity of sequences. Researchers usually set the same signs for various classifications (such as strains, groups, genus, and groups). The sequence is usually divided into different OTUs according to a similarity threshold of 97%, and each OTU is usually considered as coming from one microbial species [20]. If the similarity is less than 97%, it can be regarded as coming from different species, and if the resemblance is less than 95%, it can be regarded as a different genus. The program of this step can be designed as shown in figure 13.

```
rule precluster_97:
    input:
        i="multiplexed_linearized_dereplicated_vsearch_min2.fasta"
    output:
        uc="all.preclustered.uc",
        centroids="all.preclustered.fasta"
    shell:
        """
        vsearch  --cluster_size {input.i} \
        --threads 5 --id 0.97 --strand plus --sizein \
        --sizeout --fasta_width 0 --uc {output.uc} \
        --centroids {output.centroids}
        """
```

Figure 13: Rule Cluster at 97%

The next purpose was using `denovo` to get rid of chimera. `Denovo` assembly is a method of sequencing from the head of the genome. This method can be used for transcription without a reference genome [21]. Since upstream sequences may produce sequences with repetitive fragments when they overlap, `devono` is used to eliminate these chimeras. The sequences processed by `denovo` will get better used in the downstream identification process. Chimera referred to a sequence reconstructed from different sequence shown as figure 14. Then we could use a database supported by Dr.Umer to remove the low bacteria.

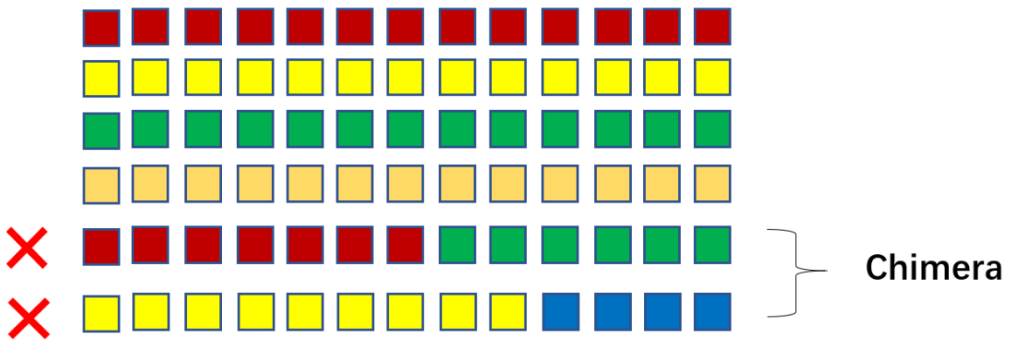

Figure 14: The example of removing chimera

```
rule denovo:
    input:
        i="all.preclustered.fasta"
    output:
        o="all.denovo.nonchimeras.fasta"
    shell:
        """
        vsearch --uchime_denovo {input.i} \
        --sizein --sizeout --fasta_width 0 \
        --nonchimeras {output}
        """

rule ref_chimera_detection:
    input:
        i="all.denovo.nonchimeras.fasta"
    output:
        o="all.ref.nonchimeras.fasta"
    shell:
        """vsearch --uchime_ref {input.i} \
        --threads 5 --db /home/opt/vsearch_GOLD_DATABASE/gold.fasta \
        --sizein --sizeout --fasta_width 0 --nonchimeras {output.o}
        """
```

Figure 15: Rule denovo

The final step was generating the OTU table. First, the user renamed all sequences to be prefixed with "OTU_" and gives an integer identifier. Then the users could use the OTUs as reference to search the original multiplexed sequences. Finally, the OTU table could be created in a tab-delimited format.

```
rule OTU_table:
    input:
        i="all.ref.nonchimeras.fasta"
    output:
        o1="all.clustered.uc",
        o2="all.otus.fasta",
        o3="all.otutab.txt"
    shell:
        """
        vsearch --cluster_size {input.i} \
        --threads 5 --id 0.97 --strand plus --sizein --sizeout \
        --fasta_width 0 --uc {output.o1} --relabel OTU_ \
        --centroids {output.o2} --otutabout {output.o3}
        """
```

Figure 16: Rule OTU table

OTU table can be regarded as the most basic and important document in

microbial diversity analysis. OTU table is a reference table obtained after all OTUs are clustered and annotated. Calculate the distance measurement or similarity between two different sequences through a certain distance measurement method, and then compare with the OTU table to determine the biological characteristics of a specific OTU.

4. Discussion

In order to visualize the process, users could use a command `snakemake --dag | dot -Tsvg> dag.svg` to output the entire process. Due to the excessive number of the input FASTQ files, only some sequences were intercepted and displayed as samples, as shown in Figure 17. It can be seen from this DAG diagram that the tasks in Snakemake workflow can be executed independently of each other.

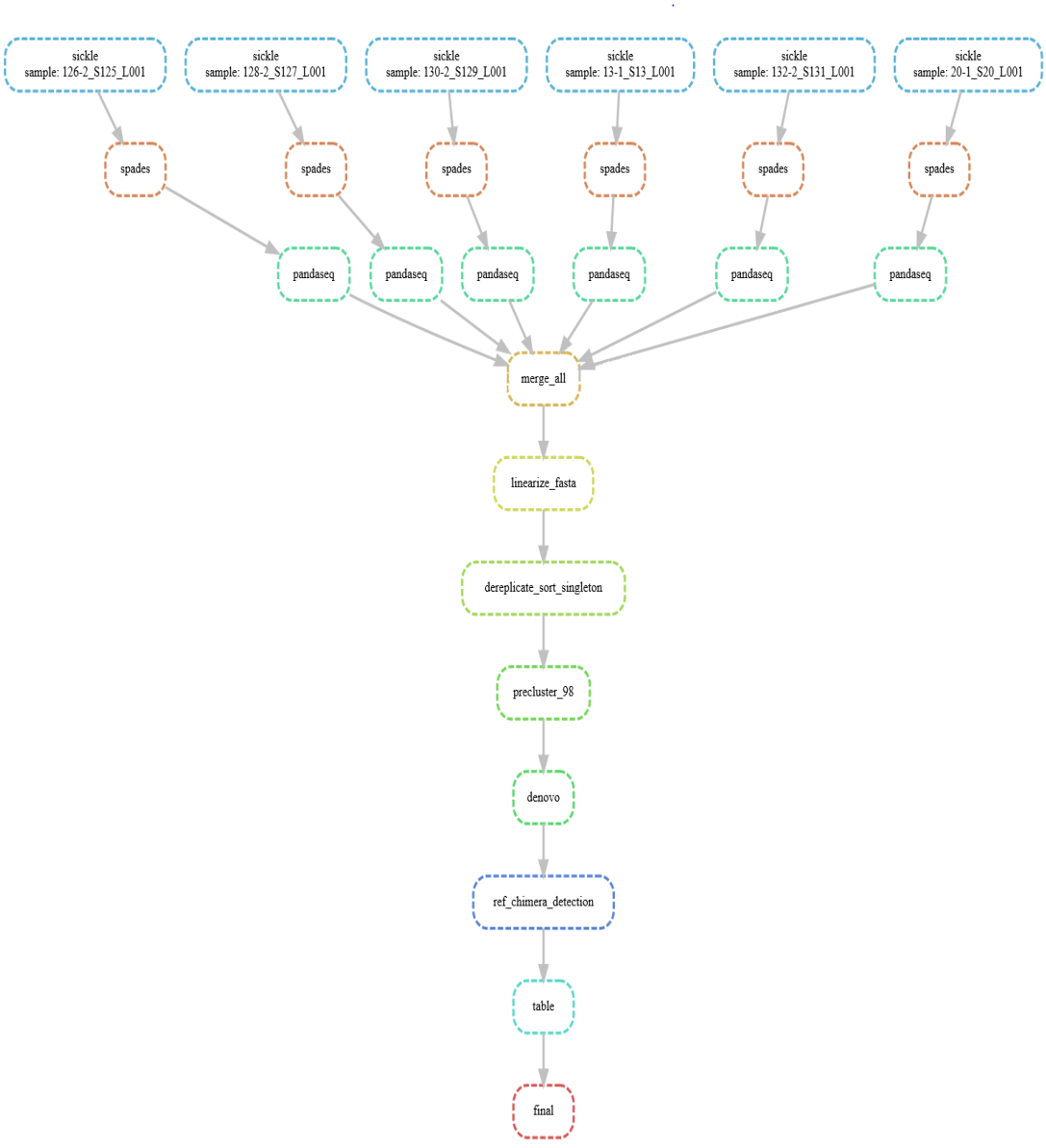

Figure 17: DAG for the example Snakemake Workflow

In order to check whether the sequences are processed correctly in each step, user can check the reads after each step to verify. By using `bioawk -cfastx 'END{print NR}'` command, these reads could be counted. Table 2 recorded the original reads of each sequence, trimmed reads after quality trimming, and overlapped reads after integration.

	Original Reads	Trimmed Reads	Overlapped Reads
109-2	5382	5368	5270
110-2	3745	3731	3659
113-2	1907	1904	1868
114-2	7536	7518	7427
115-2	5585	5564	5480
117-2	206085	205750	203582
119-2	13616	13561	13393
1-1	21165	21085	20723
120-2	9784	9744	9606
126-2	1689	1675	1640
128-2	27086	27023	26760
130-2	13728	13217	13061
13-1	32388	32323	31915
132-2	13130	13100	13013
20-1	55596	55493	54944
27-1	23450	23410	22981
32-1	4586	4566	4468
38-1	10527	10499	10367
45-1	5007	4992	4874
51-1	35597	35463	35066
56-1	33292	33199	32869
62-1	7538	7331	7163
68-1	17025	16986	16672
7-1	1	1	1

Table 2: The sequences reads of every step

	Original Reads	Trimmed Reads	Overlap Reads
109-2	1	0.997398737	0.979189892
110-2	1	0.996261682	0.977036048
113-2	1	0.998426848	0.97954903
114-2	1	0.997611465	0.985536093
115-2	1	0.996239928	0.981199642
117-2	1	0.998374457	0.987854526
119-2	1	0.995960635	0.983622209
1-1	1	0.996220175	0.979116466
120-2	1	0.995911693	0.981807032
126-2	1	0.991711072	0.970988751
128-2	1	0.997674075	0.987964262
130-2	1	0.962776807	0.95141317
13-1	1	0.997993084	0.985395826
132-2	1	0.997715156	0.991089109
20-1	1	0.998147349	0.988272538
27-1	1	0.998294243	0.98
32-1	1	0.995638901	0.974269516
38-1	1	0.997340173	0.984800988
45-1	1	0.997004194	0.973437188
51-1	1	0.996235638	0.985083013
56-1	1	0.997206536	0.987294245
62-1	1	0.972539135	0.950252056
68-1	1	0.997709251	0.979265786
7-7	1	1	1

Table 3: Normalized Reads

Since the length of each sequence is different, in order to show the changes of reads in each step more clearly, users can standardize all reads. Table 3 is the value of each read after normalization processing.

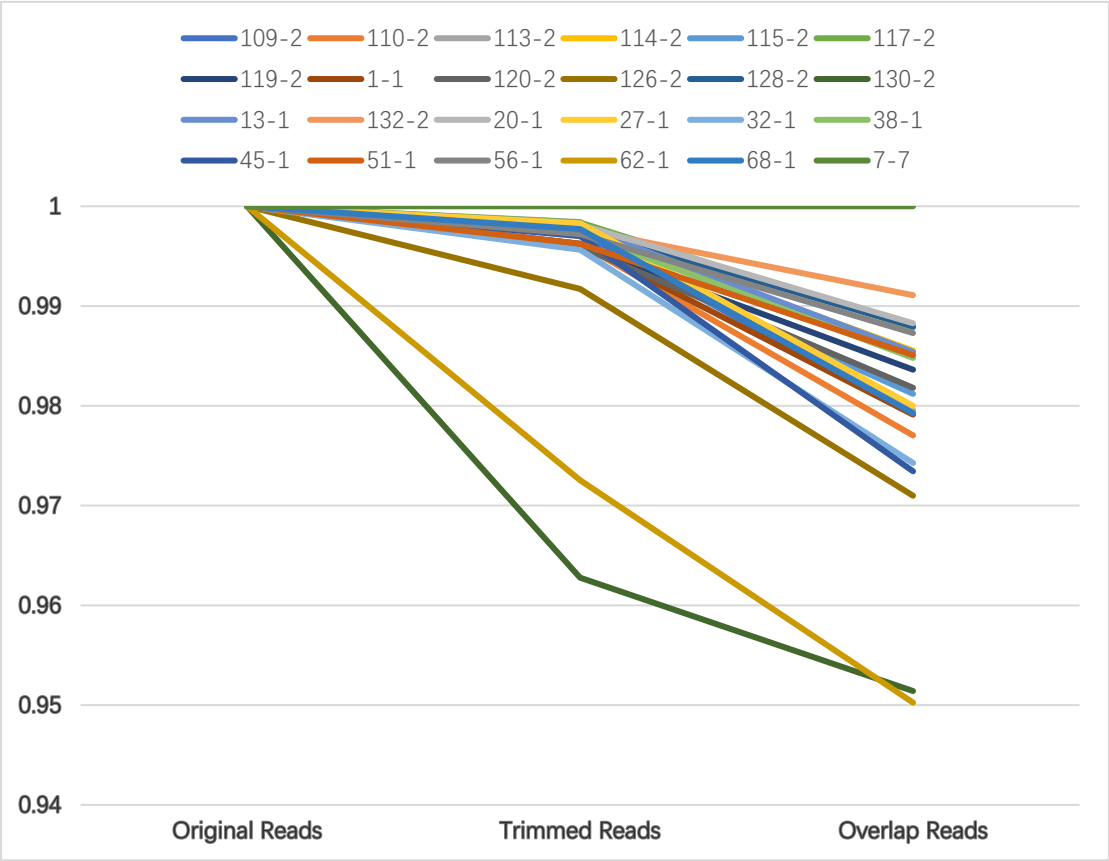

Figure 18: Reads of analysis steps

This graph showed how the reads of every analysis step changed. The x axis meaned steps of analysis and the y axis meaned normalized reads. It could be seen that most of all sequences remain over 97% reads after overlapped except "130-2" and "62-1".

As mentioned in the above section, the Snakemake Workflow finally generated 3 files: `all.clustered.uc`, `all.otus.fasta` and a OTU table.

`all.otus.fasta` file contained all of the OTU sequences and a part of OTU sequences was shown as figure 19.

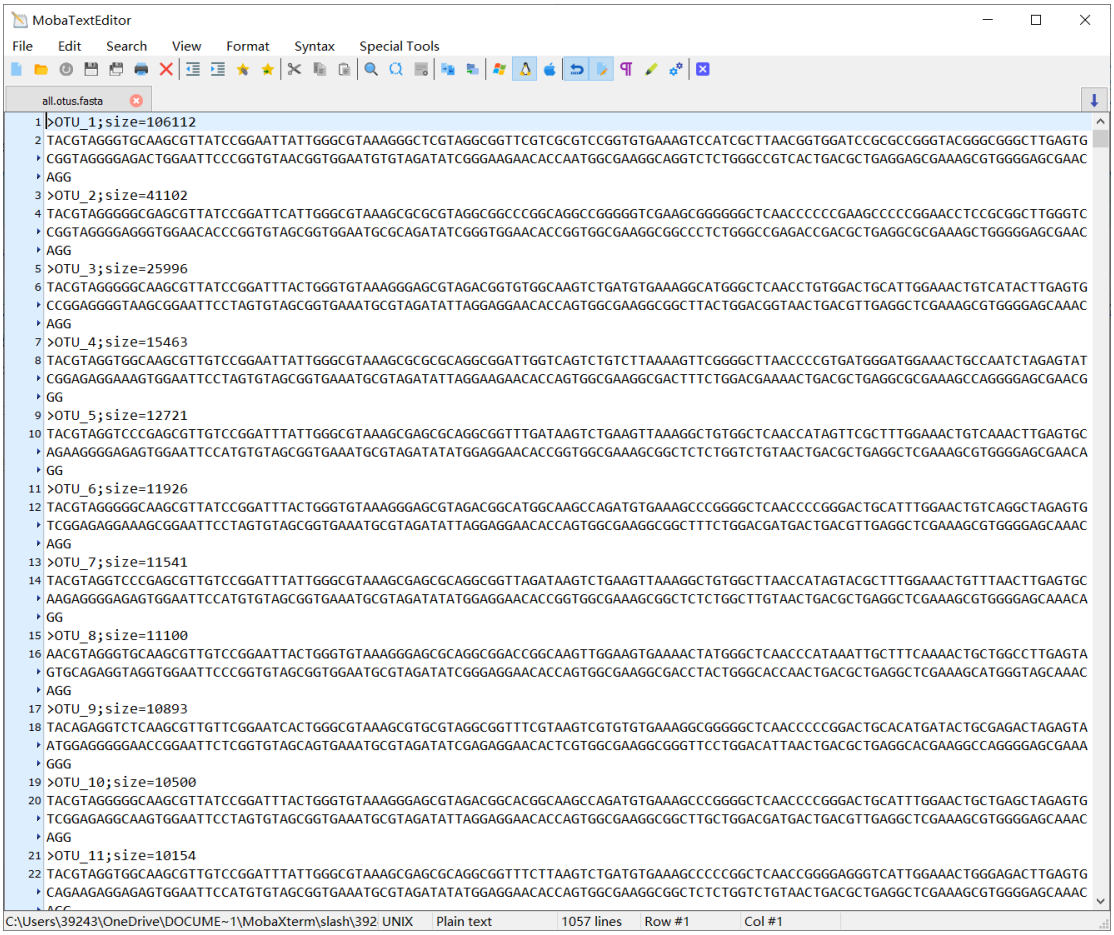

Figure 19: Partial OTU FASTQ file

There were thousands of 16S sequences obtained by high-throughput sequencing. If each sequence is to be annotated with species, it would be a lot of work and time-consuming. Moreover, errors in the process of 16S amplification and sequencing would reduce the accuracy of the results. OTU was introduced into 16S sequencing for analysis. First, users could cluster similar sequences, divided them into a small number of taxa, and annotated species based on taxa. This not only simplified the workload and improved the analysis efficiency, but also removed some sequencing errors during the clustering process, which improves the accuracy of the analysis [22]. If users wanted to check whether the OTU sequence exists or correct, it could be matched on the website https://blast.ncbi.nlm.nih.gov/Blast.cgi. In order to prove the effectiveness of the program, `OTU_1` was chosen as a sample for

matching. The matching result was shown as figure 20, it indicated the `OTU_1` was partial sequence of bifidobacterium faecale strain 2731 16S ribosomal RNA gene.

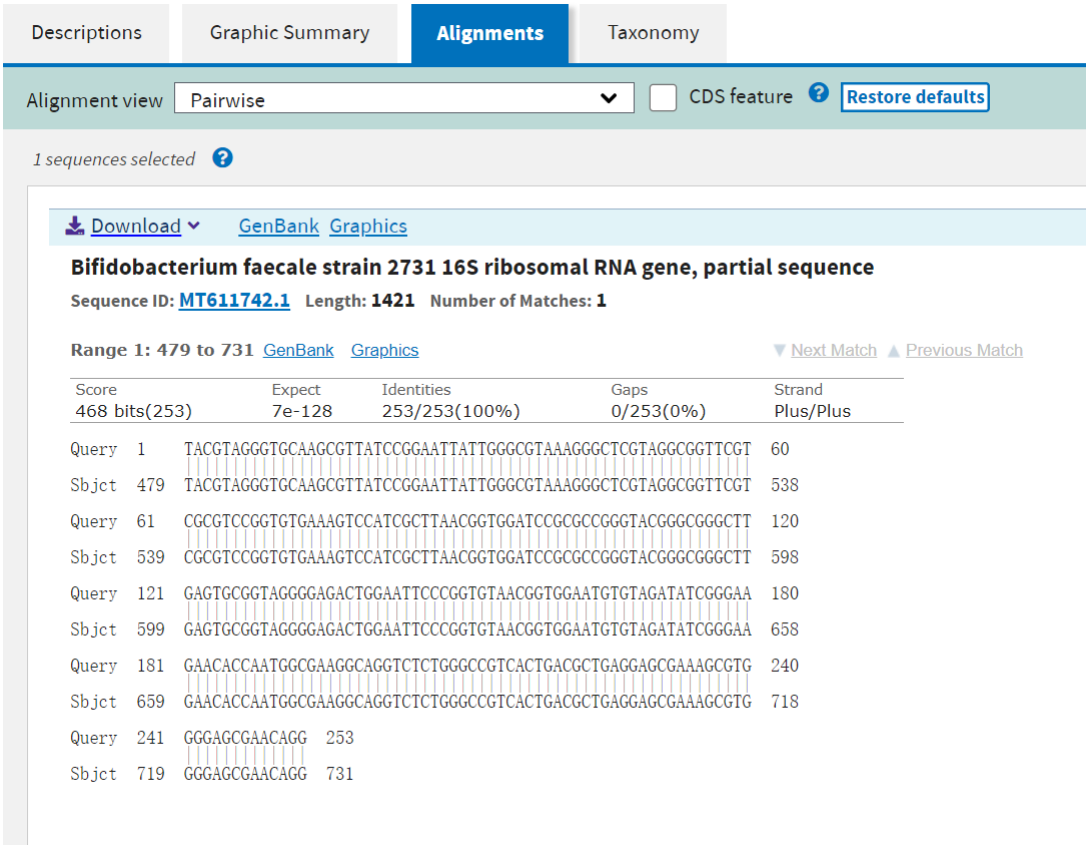

Figure 20: OTU_1 matched result

After clustering the OTUs and classifying species annotations, an OTU table could be generated [23]. This OTU_table generated by Snakemake workflow in the project contained the OTU type and sequence number of each sample. Due to the denovo method of clustering, OTU_table retained the maximum species sequences in the sample. If the amount of sequencing in a sample was larger (the sequencing technology cannot guarantee the absolute consistency of the sequencing amount of each sample), the number of sequences of various microorganisms in the sample would be more than other samples, and the amount of sequences for every OTU increased accordingly. Therefore, the sequence number of OTU could not be directly compared between samples.

Instead, it was necessary to convert the number of sequences into a ratio, which was the relative abundance (the amount of sequences divided by the total amount of sample). Then used this ratio for horizontal comparison. Due to the large size of OTU table file, the partial diagram was shown in Figure 21.

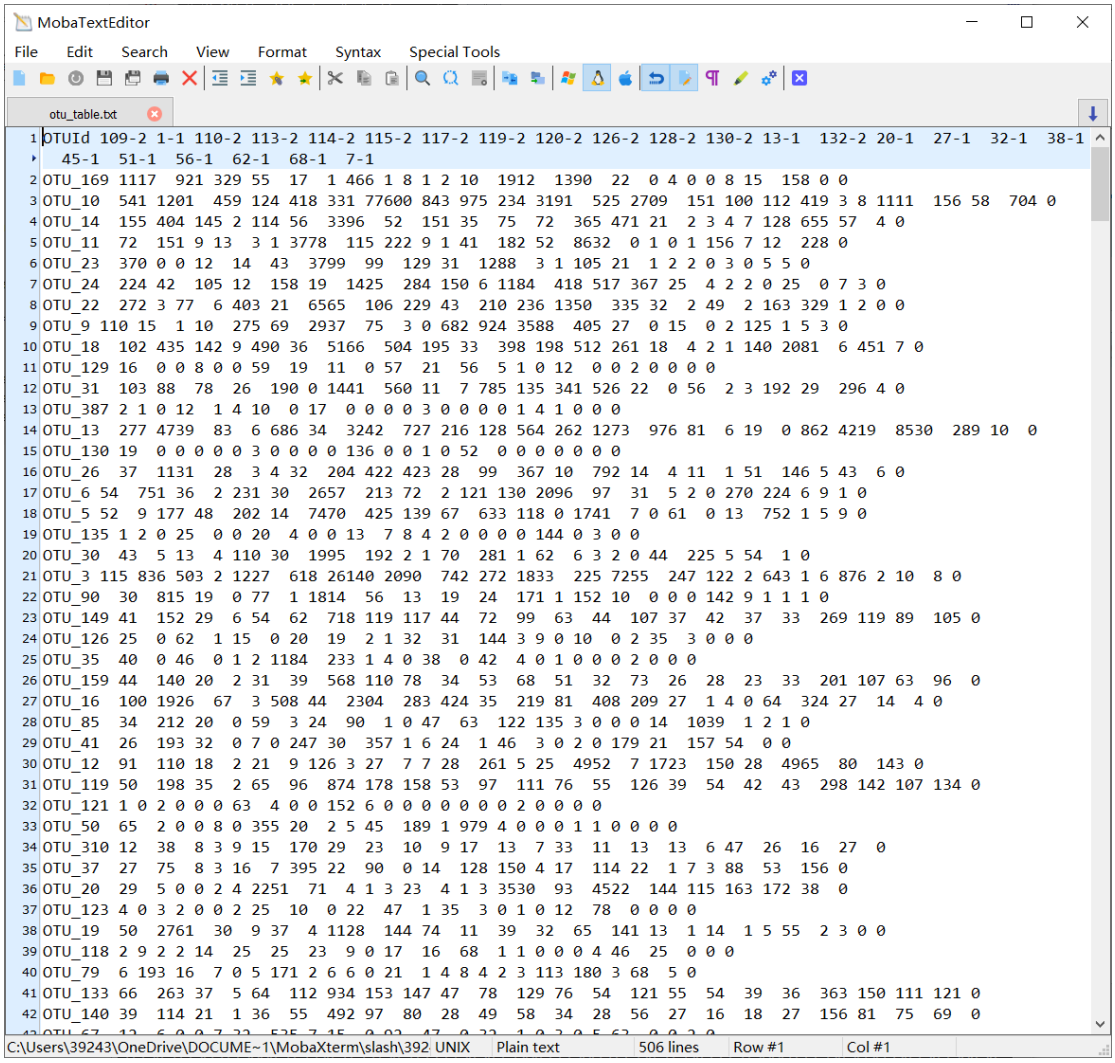

Figure 21: Partial diagram of OTU table

5. Conclusion

The main goal of this project is to build a pipeline for automated processing of metagenomics data. This goal could be accomplished by creating a Snakemake workflow program. By using this pipeline, users could automatically perform sequencing by inputting the specified file at the initial stage when analyzing metagenomics data. This design ensured the stability and repeatability of the program.

The sequencing framework built on Snakemake reduces the requirements of the process for data format conversion and script writing, and could process transcript data simply, efficiently and conveniently. In this thesis, a workflow for metagenomics analysis was initially established. Using this workflow, a set of sequencing data analysis processes could be efficiently completed, and corresponding data and visualization results could be obtained. Based on this Snakemake workflow, sequencing analysis could be made easier to perform, which provided convenience for researchers with a non-computer background to perform transcriptome analysis. The construction of a sequencing data analysis workflow based on Snakemake could provide a reference for the construction of subsequent related analysis frameworks.

The research of microbiome largely depends on the level of technology and methods of microbiome. With the development of microbiome, new algorithms and computing platforms are constantly being developed. However, different omics techniques have their own advantages and disadvantages. Therefore, in addition to developing new technologies and methods to cope with the continuously generated microbial data, it is also necessary to focus on the integration and complementarity of various methods to improve accuracy and complementarity. In terms of data analysis, microbiology research involves the acquisition of large amounts of data,

statistical analysis and modeling. Therefore, the development of bioinformatics is particularly important. Only by further improving analysis technology, developing appropriate algorithms and software, and organically combining analysis technology, data processing, multivariate statistical analysis and visualization, can we better promote microbiology research.

Microbiology data analysis will be transformed into microbiology data science. The collection, storage, function mining and development and utilization of big data of microbiome are the core issues that restrict the development of microbiome. In terms of data collection, a standardized analysis process should be developed and implemented to ensure that the data is reproducible, robust, reproducible and universal. In the process of data analysis, relevant reference databases should be further enriched. In recent years, as the cost of sequencing has decreased, microbiome data has increased dramatically. But most of the data is used once. Therefore, how to fully excavate and use the existing massive sequencing data is another important challenge facing microbiology research. In the process of microbiome research and development, we should strongly support the development of microbiome data science, formulate effective data standards and achieve interoperability, and establish a microbiome data centre that compares data, analyses data, integrates data and achieves data standardization.

In the process of building Snakemake workflow, improving the universality of related functions is a difficult problem, which required continuous debugging. In terms of workflow functions, this project had only carried out basic development, and some specific functions required more profound knowledge in other fields. This had led to the lack of richness of the workflow functions. I hope to enrich its functions in the later stage. On the other hand, the operability of workflow needed to be enhanced, and user steps should be optimized as much as possible. In terms of software selection for workflow,

this project had selected open source projects as much as possible, which was convenient to call at any time. In the follow-up work, I will continue to improve the pipeline content and parameter configuration to adapt to more different RNA sequences.

6. References

[1]. Wooley, J. C., Godzik, A. and Friedberg, I. (2010) 'A primer on metagenomics', PLoS Computational Biology, 6(2).

[2]. Hilty, M. et al. (2010) 'Disordered microbial communities in asthmatic airways', PLoS ONE, 5(1).

[3]. Chen, L., Fu, C. and Wang, G. (2017) 'Microbial diversity associated with ascidians: a review of research methods and application', Symbiosis. Symbiosis, 71(1), pp. 19–26.

[4]. Yang, Y. and Smith, S. A. (2013) 'Optimizing de novo assembly of short-read RNA-seq data for phylogenomics.', BMC genomics, 14.

[5]. Sikkema-Raddatz, B. et al. (2013) 'Targeted Next-Generation Sequencing can Replace Sanger Sequencing in Clinical Diagnostics', Human Mutation, 34(7), pp. 1035–1042.

[6]. Marchant, J. et al. (2014) 'Comparative evaluation of the new FDA approved THxIDTM-BRAF test with high resolution melting and sanger sequencing', BMC Cancer, 14(1), pp. 1–9.

[7]. Metzker, M. L. (2010) 'Sequencing technologies the next generation', Nature Reviews Genetics. Nature Publishing Group, 11(1), pp. 31–46.

[8]. Franzosa, E. A. et al. (2015) 'Sequencing and beyond: Integrating molecular "omics" for microbial community profiling', Nature Reviews Microbiology. Nature Publishing Group, 13(6), pp. 360–372.

[9]. Illumina Inc. (2017) 'Illumina sequencing introduction', Illumina sequencing introduction, (October), pp. 1–8. doi: http://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[10]. Yang, Y. et al. (2018) 'Application Progress of 16s rRNA High-Throughput Sequencing Technology on Human Medicine', pp. 16–18.

[11]. Klindworth, A. et al. (2013) 'Evaluation of general 16S ribosomal RNA gene PCR primers for classical and next-generation sequencing-based diversity studies', Nucleic Acids Research, 41(1), pp. 1–11.

[12]. Sharma, C. M. and Konrad, U. F. (2014) 'BIOINFORMATICS APPLICATIONS NOTE READemption — a tool for the computational analysis of deep-sequencing – based transcriptome data', 30(23), pp. 3421–3423.

[13]. Köster, J. and Rahmann, S. (2012) 'Snakemake-a scalable bioinformatics workflow engine', Bioinformatics, 28(19), pp. 2520–2522.

[14]. D'Amore, R. et al. (2016) 'A comprehensive benchmarking study of protocols and sequencing platforms for 16S rRNA community profiling', BMC Genomics. BMC Genomics, 17(1).

[15]. Bankevich, A. et al. (2012) 'SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing', Journal of Computational Biology, 19(5), pp. 455–477.

[16]. Rognes, T. et al. (2016) 'VSEARCH: A versatile open source tool for metagenomics', PeerJ, 2016(10), pp. 1–22.

[17]. Masella, A. P. et al. (2012) 'PANDAseq: Paired-end assembler for illumina sequences', BMC Bioinformatics, 13(1), pp. 1–7.

[18]. Quince, C. et al. (2015) 'Extensive modulation of the fecal metagenome in children with Crohn's disease during exclusive enteral nutrition', American Journal of Gastroenterology, 110(12), pp. 1718–1729.

[19]. George, B. et al. (2017) 'Transcriptome Sequencing for Precise and Accurate Measurement of Transcripts and Accessibility of TCGA for Cancer Datasets and Analysis', Applications of RNA-Seq and Omics Strategies - From Microorganisms to Human Health.

[20]. Wu, C. et al. (2019) 'A Markov-based model for predicting the development trend of soil microbial communities in saline-alkali land in Wudi County', Concurrency Computation , 31(10), pp. 1–8.

[21]. Wang, Z. et al. (2012) 'The genome of flax (Linum usitatissimum) assembled de novo from short shotgun sequence reads', Plant Journal, 72(3), pp. 461–473.

[22]. Edgar, R. C. (2013) 'UPARSE: Highly accurate OTU sequences from microbial amplicon reads', Nature Methods, 10(10), pp. 996–998.

[23]. Olesen, S. W., Duvallet, C. and Alm, E. J. (2017) 'DbOTU3: A new implementation of distribution-based OTU calling', PLoS ONE, 12(5), pp. 1–13.

Appendices

Source code:

```
#This is adopted from
#http://pedagogix-tagc.univ-
mrs.fr/courses/ABD/practical/snakemake/snake_intro.html

#-----------------------------------------------##
## A set of functions
##-----------------------------------------------##
import sys
def message(mes):
    sys.stderr.write("|--- " + mes + "\n")

##-------------------------------------------------------------------------
----------------##
## The list of samples to be processed
##-------------------------------------------------------------------------
----------------##
SAMPLES, = glob_wildcards("samples/{sample}_R1_001.fastq")
NB_SAMPLES = len(SAMPLES)

for smp in SAMPLES:
    message("Sample " + smp + " will be processed")

rule final:
    input:
         "all.otutab.txt"

rule sickle:
    input:
        forward="samples/{sample}_R1_001.fastq",
        reverse="samples/{sample}_R2_001.fastq"
    output:
        forward_trim="samples/{sample}_R1_001_trim.fastq",
        reverse_trim="samples/{sample}_R2_001_trim.fastq",
        singlet="samples/{sample}_R2_001_singlet.fastq"
    shell:
        """sickle pe -f {input.forward} \
        -r {input.reverse} \
        -t 'sanger' \
        -o {output.forward_trim} \
```

```
        -p {output.reverse_trim} \
        -s {output.singlet} \
        -q 20 -l 10
        """

rule spades:
    input:
        r1="samples/{sample}_R1_001_trim.fastq",
        r2="samples/{sample}_R2_001_trim.fastq"
    output:
        o=directory("samples/{sample}_corrected")
    shell:
        """
        spades.py -1 {input.r1} -2 {input.r2} \
        --only-error-correction \
        --careful -o {output.o}
        """

rule pandaseq:
    input:
        l=directory("samples/{sample}_corrected")
    output:
        o="samples/{sample}_overlap.fasta"
    shell:
        """
        pandaseq -f {input.l}/corrected/*_R1*fastq.gz \
        -r {input.l}/corrected/*_R2*fastq.gz > {output.o}
        """

#Help for including shell commands in snakemake:
#https://stackoverflow.com/questions/50965417/awk-command-fails-in-
snakemake-use-singularity

rule merge_all:
    input:
        i=expand("samples/{smp}_overlap.fasta", smp=SAMPLES)
    output:
        m="multiplexed.fasta"
    shell:
        """
        (for i in $(echo {input.i}); \
        do awk -v k=$i '/^>/{{$0=">barcodelabel="k";\
        S"(++i)}}1' $i; done) > multiplexed.fasta
        """
```

```
rule linearize_fasta:
    input:
        i="multiplexed.fasta"
    output:
        o="multiplexed_linearized.fasta"
    shell:
        """
        awk 'NR==1 {{print ; next}} \
        {{printf /^>/ ? "\\n"$0"\\n" : $1}} \
        END {{print}}' {input.i} > {output.o}
        """

rule dereplicate_sort_singleton:
    input:
        i="multiplexed_linearized.fasta"
    output:
        uc="multiplexed_linearized_dereplicated_vsearch_min2.uc",
        fasta="multiplexed_linearized_dereplicated_vsearch_min2.fasta"
    shell:
        """
        vsearch --threads 20 \
        --derep_fulllength {input.i} --minuniquesize 2 \
        --sizein --sizeout --fasta_width 0 --uc {output.uc} \
        --output {output.fasta}
        """

rule precluster_97:
    input:
        i="multiplexed_linearized_dereplicated_vsearch_min2.fasta"
    output:
        uc="all.preclustered.uc",
        centroids="all.preclustered.fasta"
    shell:
        """
        vsearch  --cluster_size {input.i} \
        --threads 5 --id 0.97 --strand plus --sizein \
        --sizeout --fasta_width 0 --uc {output.uc} \
        --centroids {output.centroids}
        """

rule denovo:
    input:
        i="all.preclustered.fasta"
```

```
    output:
        o="all.denovo.nonchimeras.fasta"
    shell:
        """
        vsearch --uchime_denovo {input.i} \
        --sizein --sizeout --fasta_width 0 \
        --nonchimeras {output}
        """

rule ref_chimera_detection:
    input:
        i="all.denovo.nonchimeras.fasta"
    output:
        o="all.ref.nonchimeras.fasta"
    shell:
        """vsearch --uchime_ref {input.i} \
        --threads 5 --db /home/opt/vsearch_GOLD_DATABASE/gold.fasta \
        --sizein --sizeout --fasta_width 0 --nonchimeras {output.o}
        """

rule OTU_table:
    input:
        i="all.ref.nonchimeras.fasta"
    output:
        o1="all.clustered.uc",
        o2="all.otus.fasta",
        o3="all.otutab.txt"
    shell:
        """
        vsearch --cluster_size {input.i} \
        --threads 5 --id 0.97 --strand plus --sizein --sizeout \
        --fasta_width 0 --uc {output.o1} --relabel OTU_ \
        --centroids {output.o2} --otutabout {output.o3}
        """
```