UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Optimizing 16S Sequencing Analysis Pipelines

Mestrado em Bioinformática e Biologia Computacional
Especialização em Bioinformática

Samuel Dias Rosa Viana

Dissertação orientada por:
Cátia Luísa Santana Calisto Pesquita
Daniel Pedro de Jesus Faria

2016

To my friends and family.

Abstract

The 16S rRNA gene is a widely used target for phylogenetic analysis of prokaryote communities. This analysis starts with the sequencing of the 16S rRNA gene of a microbial sample, and includes several steps such as paired-end merging (when the sequencing technique produces paired-end reads), chimera removal, clustering, and sequence database search. The end-product is the phylogeny of the prokaryote taxa in the sample and an estimation of their abundance.

The problem is that there are multiple tools available to carry out this analysis, and it is unclear which is the most effective. Namely, there are three analysis pipelines in wide use by the community: mothur, QIIME and USEARCH. These use different paired-end merging algorithms, different clustering algorithms, and different sequence reference databases (Silva, Greengenes, and RDP respectively). Additionally, there are a number of other paired-end mergers available and again, it is unclear which performs better in the context of this analysis.

In this study, we start by evaluating each of the seven publicly available paired-end merging algorithms: BBmerge, FastqJoin (QIIME's merger), FLASH, mothur's merger, PANDAseq, PEAR and USEARCH's merger. Then, we assess the effectiveness of each the three analysis pipelines in conjunction with each of the three reference databases, and each of the most promising paired-end mergers.

To do this evaluation, we use two sequencing datasets from mock communities, one publicly available and the other produced in-house. We evaluated the paired-end mergers by using BLAST against the known references to compare the number of mismatches before and after merging, and thereby calculate their precision and recall. We

evaluated the analysis pipelines by implementing the UniFrac metric (a community standard) in order to measure the similarity between the predicted phylogeny and the real one. We implemented both a qualitative and a quantitative variant of UniFrac.

We found that the best mergers were PEAR, FastqJoin and FLASH in terms of balance between precision and recall, whereas mothur was the best in terms of recall, and USEARCH the most correct in terms of the quality scores of the merged sequences. Regarding the analysis pipelines, in terms of qualitative UniFrac, QIIME with Silva as the reference and mothur's merger was the best on the first dataset, and mothur with either Greengenes or RDP and its own merger was the best in the second dataset. In terms of quantative unifrac, mothur with Greengenes and its own merger was the best for the first dataset, and USEARCH with SILVA and mothur's merger was the best on the second dataset.

We concluded that having a high recall in the merging step is more important than having a high precision for the downstream phylogenetic analysis, as mothur's merger was either the best or tied for the best in all settings.

Keywords: bioinformatics, phylogeny, 16s gene, pipelines, paired-end mergers

Resumo

O gene de rRNA 16S é amplamente usado para a análise filogenética de comunidades de procariotas. Esta análise inicia-se com a sequenciação deste de uma comunidade microbiana, e inclui diversos passos tais como a junção ("merging") de sequências emparelhadas ("paired-end reads") - quando a sequenciação produz este formato - , remoção de quimeras, agrupamento ("clustering") e comparação com sequências conhecidas guardadas em base de dados. O produto final é a obtenção dos taxa de procariotas presentes na amostra e uma estimativa da sua respectiva abundância.

O problema que se coloca é que existem várias ferramentas informáticas ("pipelines") disponíveis para realizar esta análise, e não é claro qual delas a que produz melhor desempenho. Nomeadamente, existem três "pipelines" a serem bastante usadas pela comunidade: mothur, QIIME e USEARCH. Cada uma usa diferentes algoritmos de "merging", "clustering" e bases de dados de 16S (Silva,Greengenes e RDP, respectivamente). Para além disso, existem também vários mergers independentes disponíveis e, uma vez mais, não é claro qual deles apresenta um melhor desempenho no contexto desta análise.

Neste trabalho, começámos por avaliar cada um de sete "mergers" disponíveis gratuitamente: BBmerge, FastqJoin (usado pelo QIIME), FLASH, "merger" do mothur, PANDAseq, PEAR e "merger" do USEARCH. Após este passo, é feita uma avaliação a cada uma das três pipelines de análise conjuntamente com cada uma das três bases de dados, e com cada um dos mergers que conseguiram melhor desempenho no passo anterior.

Para proceder a esta avaliação, usámos dois conjuntos de dados ("datasets") provenientes de comunidades simuladas ("mock"), um disponível ao

público e outro criado internamente. Os mergers foram avaliados efectuando alinhamento contra as referências conhecidas e comparando o número de discordâncias ("mismatches") antes e depois do "merge", usando-os para calcular a sua precisão e sensibilidade ("recall"). Por sua vez, as pipelines de análise foram avaliadas implementando a métrica Unifrac (um "standard" estabelecido) de forma a medir a semelhança entre a filogenia prevista e a real. O Unifrac foi implementado nas suas versões quantitativa e qualitativa.

Estabelecemos como melhores mergers o PEAR, FastqJoin e o FLASH devido ao equilíbrio entre a precisão e o "recall", enquanto que o mothur como merger foi o melhor em termos de "recall", sendo o USEARCH o merger que age de forma mais correcta no cálculo dos valores de qualidade das sequências resultantes do merge.

Relativamente à pipelines de análise, e em termos de Unifrac qualitativo, o QIIME usando o Silva como referência foi o melhor usando o merger do mothur para o primeiro dataset, sendo o mothur usando Greengenes ou RDP com o seu próprio merger o melhor no segundo dataset. Para o Unifrac quantitativo, o mothur com o Greengenes e o seu próprio merger foi o melhor no primeiro dataset e o USEARCH usando SILVA com o merger do mothur novamente a ser o melhor no segundo dataset.

Concluímos que possuir um recall elevado no passo de merge é mais importante do que possuir uma precisão elevada para a análise filogenética subsequente, uma vez que o merger do mothur foi o melhor e esteve sempre associado com os melhores desempenhos em todas as combinações testadas de merger, pipeline e base de dados.

Palavras Chave: bioinformática, gene 16S, filogenia, pipelines, paired-end mergers

Resumo Alargado

O gene de rRNA 16S que codifica para a componente menor do ribossoma é um gene amplamente usado para a análise filogenética de comunidades de procariotas. Este gene, por ser responsável pela síntese de um componente fundamental na sobrevivência da célula procariota, é omnipresente em todos os procariotas, apresentando, ao longo dos seus 1500 pares de bases de comprimento, zonas chamadas hipervariáveis ("HV"), que existem em número de nove, e que permitem identificar os "taxa" em comunidades bacterianas. De entre todas estas zonas hipervariáveis, é a zona HV4 que se assume como de especial importância, visto ser a que apresenta uma maior variabilidade que permite estabelecer a filogenia em estudos de comunidades bacterianas.

Esta análise inicia-se com a sequenciação do gene 16S de uma amostra microbiana, e inclui diversos passos tais como o (1) limpeza de porções de menor qualidade ("trimming"), (2) junção ("merging") de pares de sequências numa única sequência, (3) remoção de quimeras, (4) agrupamento ("clustering") em "clusters", e (5) sua respectiva identificação taxonómica, usando como medida de comparação sequências conhecidas guardada numa base de dados de 16S. O produto final é a obtenção dos "taxa" de procariotas presentes na comunidade amostrada e uma estimativa da sua respectiva abundância.

Relativamente ao "clustering", podemos ter diferentes níveis de semelhança a serem usados na construção dos "clusters": os valores mais usados são de 97 e 99

O problema que se coloca é que existem várias ferramentas informáticas disponíveis ("pipelines") para realizar esta análise, e não é claro qual delas é a que obtém melhor desempenho. Nomeadamente, existem três "pipelines" a serem bastante usadas pela comunidade:

mothur, QIIME e USEARCH. Cada uma usa diferentes algoritmos de "merging", "clustering" e bases de dados de 16S (Silva, Greengenes e RDP, respectivamente). Para além disso, existem também mergers independentes da "pipeline" e, uma vez mais, não é claro qual deles apresenta um melhor desempenho no contexto desta análise.

Estas ferramentas trabalham sobre conjuntos de dados ("data sets") que são o produto da sequenciação de amostras de DNA. O sequenciador cria dados sob a forma de pares de sequências ("paired-end reads") que resultam da leitura do mesmo fragmento vindo de direcções opostas. Consoante o tamanho do fragmento, poderá haver lugar ao "merge" destas duas sequências numa sequência única se as mesmas apresentarem uma zona comum ("overlap") que o permita. Tal apresenta diversas vantagens que são: melhoria dos valores de qualidade associados à posição de cada nucleótido na sequência resultante do "merge", para além de simplificar os passos de montagem ("assembly") ou análise posteriores, uma vez que ficamos com aproximadamente metade do número de sequências original.

Algumas dificuldades adicionais poderão surgir durante o processo da "pipeline" como por exemplo "quimeras" que são artefactos resultantes do processo de amplificação do DNA original e que poderão levar ao desvio dos valores esperados com as quimeras a poderem ser atribuídas erroneamente a "taxa" não esperados.

Neste trabalho, começámos por avaliar cada um de sete "mergers" disponíveis gratuitamente : BBmerge, FastqJoin (usado pelo QIIME), FLASH, "merger" do mothur, PANDAseq, PEAR e "merger" do USEARCH. Após este passo, é feita uma avaliação a cada uma das três pipelines de análise conjuntamente com cada uma das três bases de dados, e com cada um dos mergers que conseguiram melhor desempenho no passo anterior.

Para proceder a esta avaliação, usámos dois conjuntos de dados ("datasets") provenientes de comunidades simuladas ("mock") de bactérias, um disponível ao público e outro criado internamente. O primeiro dataset

("Mock1") possui 21 espécies de bactérias em abundâncias idênticas e é constituído por quase meio milhão de pares de sequências, enquanto que o segundo dataset ("Mock2") possui apenas 8 espécies, com abundâncias dissemelhantes dispostas em "escada", sendo a espécie mais abundante a um nível de um milhão de vezes da espécie menos abundante.

Os mergers foram avaliados seguindo o esquema: (1) remoção de replicados e limpeza de extremidades de menor qualidade ("trimming"), (2) alinhamento contra as referências conhecidas, obtendo-se o número de sequências que não são passíveis de serem juntas numa só . Após o (3) "merge" propriamente dito, (4) procede-se a um novo alinhamento desta vez com as sequências geradas pelo merge, após o que (5) se procede à comparação do número de discordâncias antes e depois do merge, que servem de base (6) para o cálculo da matriz de confusão, precisão e "recall".

Após concluído este aspecto da análise, é efectuada uma sondagem dos valores de qualidade associados com as posições individuais nas sequências para perceber a distribuição dos valores de qualidade inerente a cada dataset, para entender em que medida cada "merger" procede à estimação das qualidades após o "merge". Ainda é testado o efeito que o "trimming" poderá ter no desempenho dos mergers. Para esse efeito usámos a ferramenta seq-tk a dois níveis de "trimming": 95 e 99

Por sua vez, as pipelines de análise foram avaliadas implementando a métrica Unifrac (um "standard" estabelecido) de forma a medir a semelhança entre a filogenia prevista e a real. O Unifrac foi implementado nas suas versões quantitativa e qualitativa.

As pipelines são constituídas basicamente pelos seguintes passos: (1) alinhamento contra uma referência de alinhamento múltiplo, (2) remoção de replicados, (3) remoção de quimeras, (4) "clustering", seguido por (5)identificação dos OTUs (ou "clusters") encontrados. A estes passos segue-se a remoção de "singletons" (que são "clusters" com

apenas uma sequência) e, finalmente, o cálculo propriamente dito da Unifrac nas suas variantes qualitativa e quantitativa.

Seleccionámos como sendo os melhores "mergers" o PEAR, FastqJoin e o FLASH devido ao equilíbrio entre a precisão e o "recall" que revelaram, enquanto que o mothur como "merger" foi o melhor em termos de "recall", sendo o USEARCH o "merger" que age de forma mais correcta no cálculo dos valores de qualidade das sequências resultantes do "merge". Relativamente ao efeito do "trimming" no desempenho dos "mergers", verificámos que um trimming muito elevado tem efeitos drásticos na precisão, mas com vantagens evidentes no "recall". Para além de tudo isto, o segundo dataset ("Mock2") alcançou melhores resultados de precisão e "recall" em virtude de apresentar uma qualidade em média mais elevada do que o primeiro dataset ("Mock1").

Relativamente às "pipelines" cinco "datasets" resultado da acção dos mergers foram submetidos: os que foram gerados pelo fastq-join, FLASH, PEAR, mothur e USEARCH. Juntando os dois passos de avaliação, ao todo para cada dataset foram testadas 45 combinações diferentes de merger/pipeline de análise 16S/base de dados de 16S.

Relativamente às "pipelines" de análise, e em termos de Unifrac qualitativo, o QIIME usando o Silva como referência foi o melhor usando o "merger" do mothur para o primeiro "dataset", tendo o mothur usando como referências Greengenes ou RDP, com o merging efectuado pelo "merger" do próprio mothur como sendo a melhor combinação no segundo dataset. Para o Unifrac quantitativo, o mothur com o Greengenes e o seu próprio merger foi o melhor no primeiro dataset e o USEARCH usando SILVA com o merger do mothur novamente a ser o melhor no segundo dataset.

Concluímos que possuir um "recall" elevado no passo de "merge" é mais importante do que possuir uma precisão elevada para a análise filogenética subsequente, uma vez que o "merger" do mothur foi o melhor e esteve sempre associado com os melhores desempenhos em todas as combinações testadas de "merger", pipeline e base de dados.

A decisão final do merger e pipeline a escolher deve-se revestir de um merger com melhor "recall" e em que o "trimming" deve ser bem ajustado de forma a que não ocorra um grande impacto na precisão.

Acknowledgements

First of all, I would like to thank profs. Cátia Pesquita from Faculdade de Ciencias da Universidade Lisboa (FCUL), and Daniel Faria for accepting to be my thesis co-supervisors in this a work of a nine-months stay at the installations of the Instituto Gulbenkian de Ciência (IGC).

I thank also prof. Daniel Sobral, head of the Bioinformatics Unit at the IGC for accepting my application as trainee for at the Unit and for using the unit resources . Thanks also to João Sobral from the Genetic Expression Unit for providing one of the datasets that I used in my work.

Also thanks for Tiago Macedo, Paulo Almeida, João Costa, Pedro Fernandes, Patrícia Santos and Isabel Marques, for their fellowship at my time of stay on IGC.

Contents

Table of contents xvii

List of figures xix

List of tables xxi

1 Introduction 1

1.1 Motivation . . . 1
1.2 Objectives . . . 1
1.3 Contributions . . . 2
1.4 Overview . . . 2

2 Basic concepts 5

2.1 16S gene as marker . . . 5
2.2 Next-Generation Sequencing (NGS) . . . 5
2.3 The merging process . . . 6
2.4 Clustering into Operational Taxonomic Units (OTUs) . . . 8

3 State of the art 11

3.1 Trimming . . . 11
3.1.1 The algorithm used by `seq-tk` . . . 11
3.2 Mergers . . . 12
3.2.1 BBMerge . . . 12
3.2.2 Fastq-join . . . 12
3.2.3 FLASH . . . 13
3.2.4 Mothur . . . 14

3.2.5 PandaSeq . . . 15
3.2.6 PEAR . . . 15
3.2.7 USEARCH . . . 16
3.3 16S Analysis Pipelines . . . 16
3.3.1 mothur . . . 17
3.3.2 QIIME . . . 17
3.3.3 USEARCH . . . 17
3.4 16S Databases . . . 18
3.4.1 Silva . . . 18
3.4.2 Greengenes . . . 18
3.4.3 RDP . . . 18
3.5 Unifrac . . . 19

4 Evaluating Paired-end mergers 21
4.1 Experimental design . . . 21
4.1.1 Datasets used . . . 21
4.1.1.1 Dataset Mock1 . . . 21
4.1.1.2 Dataset Mock2 . . . 22
4.1.2 The Mergers Evaluation pipeline . . . 22
4.1.2.1 The 'before-merge' steps in detail . . . 25
4.1.2.2 Difficulties that arise from the 'before-merge' BLAST 27
4.1.2.3 Mergers Execution . . . 27
4.1.2.4 Parameter configurations . . . 28
4.1.2.5 After-merge evaluation steps . . . 28
4.1.3 The evaluation metrics: precision and recall . . . 29
4.2 Results and discussion . . . 30
4.2.1 Mock1 . . . 30
4.2.2 Mock2 . . . 32
4.2.3 The quality scores after the merge . . . 34
4.3 The effects of trimming before the merge . . . 36
4.4 Source code availability . . . 39

5 Evaluating 16S Analysis pipelines **41**
5.1 Experimental design 41
5.1.1 The merged reads' datasets chosen 41
5.1.2 Unifrac similarity 42
5.1.3 The 16S databases 42
5.1.4 Some difficulties posed by each analysis pipeline 44
5.1.5 The pos-pipelines processing 44
5.2 Results 45
5.2.1 Mock1 45
5.2.2 Mock2 47
5.2.3 Discussion 47
5.3 Source code availability 49

6 Conclusion **51**

Appendices **53**
A Materials and Methods 55
A.1 Technical details regarding the pipelines 55
A.1.1 QIIME 55
A.1.2 Mothur 56
A.1.3 Usearch 57
B Supplementary figures and tables 58

References **67**

List of Figures

1.1 Typical steps in a 16S analysis pipeline. 2

2.1 16S gene Hypervariable regions (from (Yang *et al.*, 2016)). . . . 6
2.2 Paired-end reads overlap merging (from (Masella *et al.*, 2012)) . . 7
2.3 Three different scenarios for the overlap on paired-end reads (from (Zhang *et al.*, 2014)) 8
2.4 An example of quality scores boxplots 9

4.1 Mergers evaluation pipeline 23
4.2 Calculation of the putative overlap 26
4.3 How the reads which were excluded from the pipeline are related . 28
4.4 Precision-recall scatterplot for Mock1 31
4.5 Precision-recall scatterplot for Mock2 33
4.6 Mock1 quality scores distribution boxplot before and after merge . 35
4.7 Mock2 quality scores distribution boxplot before and after merge . 35
4.8 The trimming effects on Mock1 37
4.9 The trimming effects on Mock2 38

5.1 Main steps implemented by the 16s Analysis Frameworks 43

1 Mock1 phylogenetic tree 59
2 Mock2 phylogenetic tree 60
3 Mock1 putative overlaps distribution (logarithmic scale) 61
4 Mock2 putative overlaps distribution (logarithmic scale) 62

List of Tables

4.1 `Mock2` *a priori* known concentrations 22

5.1 The configurations that achieved the best Unifrac values per Unifrac type and dataset 46
5.2 Unweighted Unifrac similarity values calculated for dataset Mock1 46
5.3 Unweighted Unifrac similarity values calculated for dataset Mock2 48

1 Parameter configurations tested for the mergers 58
2 Precision, Recall and F-Measure on Mock1 60
3 Precision, Recall and F-Measure on Mock2 61
4 Mock1 pipelines evaluation results 63
5 Mock2 pipelines evaluation results 64
6 Weighted Unifrac similarity values calculated for dataset Mock1 . 65
7 Weighted Unifrac similarity values calculated for dataset Mock2 . 66

Chapter 1

Introduction

1.1 Motivation

The 16S rRNA gene is a widely used target for phylogenetic analysis of prokaryote communities. The entire process follows usually four steps: paired-end reads merging (1), clustering (2), taxonomic identification (3), and finally beta diversity evaluation (4). Step (1) is performed by tools called paired-end mergers, and steps (2), (3) and (4) are carried out by 16S analysis pipelines with the help of 16S database sequences (fig. 1.1). There are multiple mergers and pipelines available and there is no consensus which performs better (Kopylova *et al.*, 2016, D'Amore *et al.* (2016)), and there is a lack of comparison studies on the performance on these two tool categories.

1.2 Objectives

In this study, and with the purpose of filling the gap in comparison studies, we identified the following objectives:

- evaluate the mergers; and test how trimming can affects their performance;
- evaluate the 16S analysis pipelines themselves.

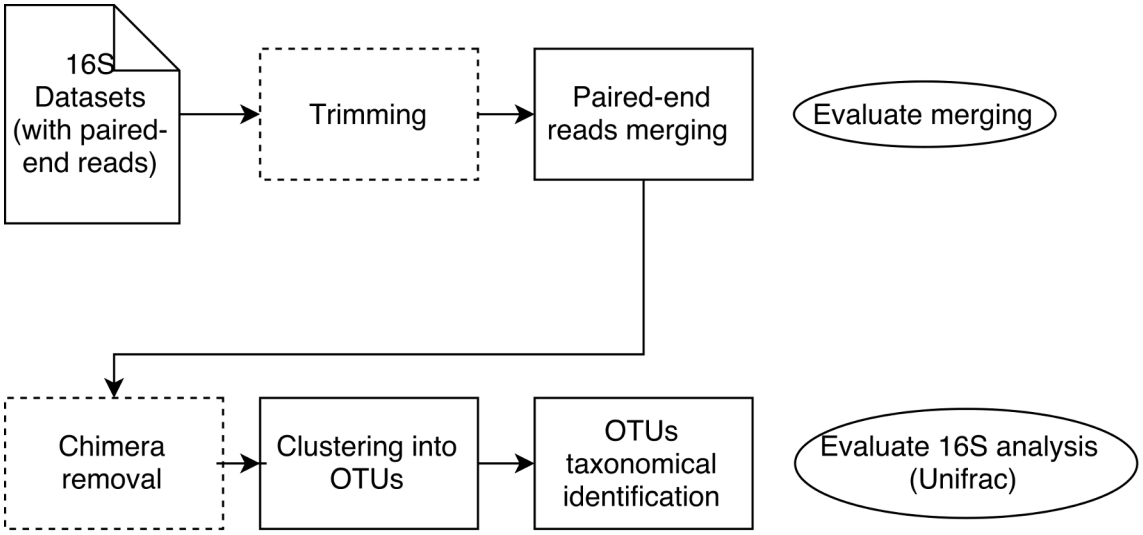

Figure 1.1: Typical steps in a 16S analysis pipeline.
Main steps in a 16S analysis pipeline (squared boxes), optional ones (dashed boxes) and points under evaluation in the present study (elliptical boxes).

1.3 Contributions

The contributions of this study are the following:

- evaluated seven paired-end mergers, measuring their effectiveness using precision and recall;
- evaluated three 16S pipelines along with the 16S sequence databases they use, having implemented Unifrac similarity to measure the divergence between the taxa revealed by the pipelines from the expected ones.

1.4 Overview

The rest of the document are organized in the following way:

- On chapter 2 the main biological concepts behind this work are described in detail.
- Chapter 3 presents the computational tools relevant for this study, namely: paired-end mergers, 16S analysis pipelines and 16S reference databases, along with the Unifrac metric.

- Chapter 4 is devoted to the evaluation of the paired-end mergers: including the methods used to evaluate them, the results obtained, and their interpretation and discussion.
- Chapter 5, in which the 16S pipelines are put to the test, describing the methods used for their evaluation, along with the results and discussion.
- Finally chapter 6 presents the main conclusions of the study as a whole.

Two appendixes are provided: one with Materials and methods and other Supplementary figures and tables.

Chapter 2

Basic concepts

In this section we describe the basic concepts relevant for our work: the role of 16S gene as marker, Next-Generation Sequencing, the merging process, and finally the clustering into operational taxonomic units (OTUs).

2.1 16S gene as marker

The 16S rRNA gene is the gene that holds the information for the sequence of the RNA that constitutes the prokaryote ribosome small subunit (SSU)(Woese *et al.*, 1990). Being vital for the protein synthesis, it is present among all prokaryotes, and being its length of about 1500 base pairs (bp), it presents some more conserved regions and others with more variation. The other regions are called *hyper-variable* regions, and it is from the differences found among these regions that is possible to use it as the basis for phylogeny alongside the prokaryotes.

There were identified until 9 hyper-variable (HV) regions (see figure 2.1) and some have more variation than the others (Chakravorty *et al.*, 2007). It is reported that the region HV4 (Soergel *et al.*, 2012) is the one that present more information suited for phylogenetic studies.

2.2 Next-Generation Sequencing (NGS)

There is a variety of DNA sequencing read lengths in the NGS platforms: from small segments, typically from 100 on the HiSeq up to 250 bp on MiSeq, both

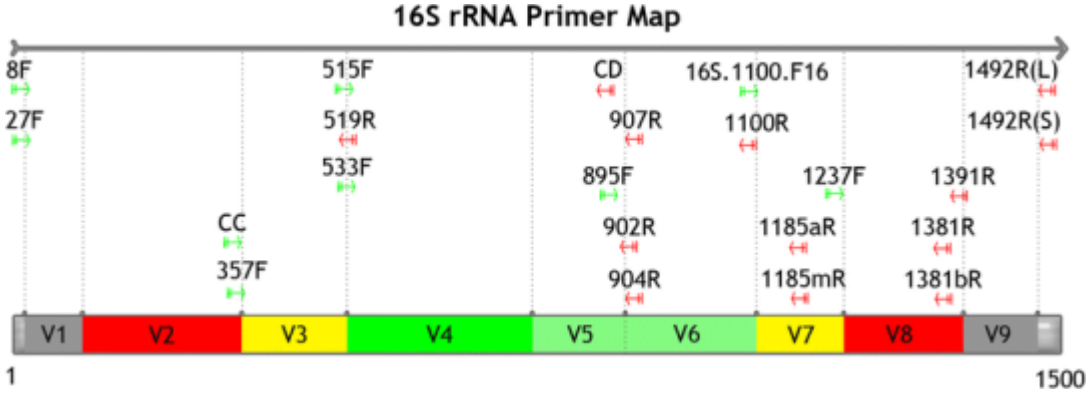

Figure 2.1: 16S gene Hypervariable regions (from (Yang *et al.*, 2016)).

being Illumina machines (Caporaso *et al.*, 2011). Usually, the smaller the reads, the cheaper are the total sequencing costs, dividing the amount of genomes per total of individual nucleotidic positions read .

However being the typical read length so short here present an disadvantage, since that in the particular case of our study, we can't still capture, however, one "entire" 16S HV region in just one single read.

One way to deal with this disadvantage, is to use a technique on NGS that allows an improved coverage for regions captured by the sequencers: it is called "paired-end" sequencing. Paired-end because the same fragment is read twice in opposite directions: one in forward (3' $\rightarrow$ 5') and the other in reverse (5' $\rightarrow$ 3'). This way, and if the fragment has size smaller than twice the typical read length for the platform, there is a great probability that the two reads have a common overlap zone at their opposite ends (fig. 2.2).

2.3 The merging process

These paired-end reads come grouped in pairs in which one is identified as the forward read and the other as the reverse read, stored in two FASTQ files, one for each member of the pair.

So, having a overlap is a way to merge the two reads of the pair into a single one. We also can call assembly, or 'merge' the joining of the two paired-end reads: the overlap zone allows their merge given that this overlap possesses sufficient

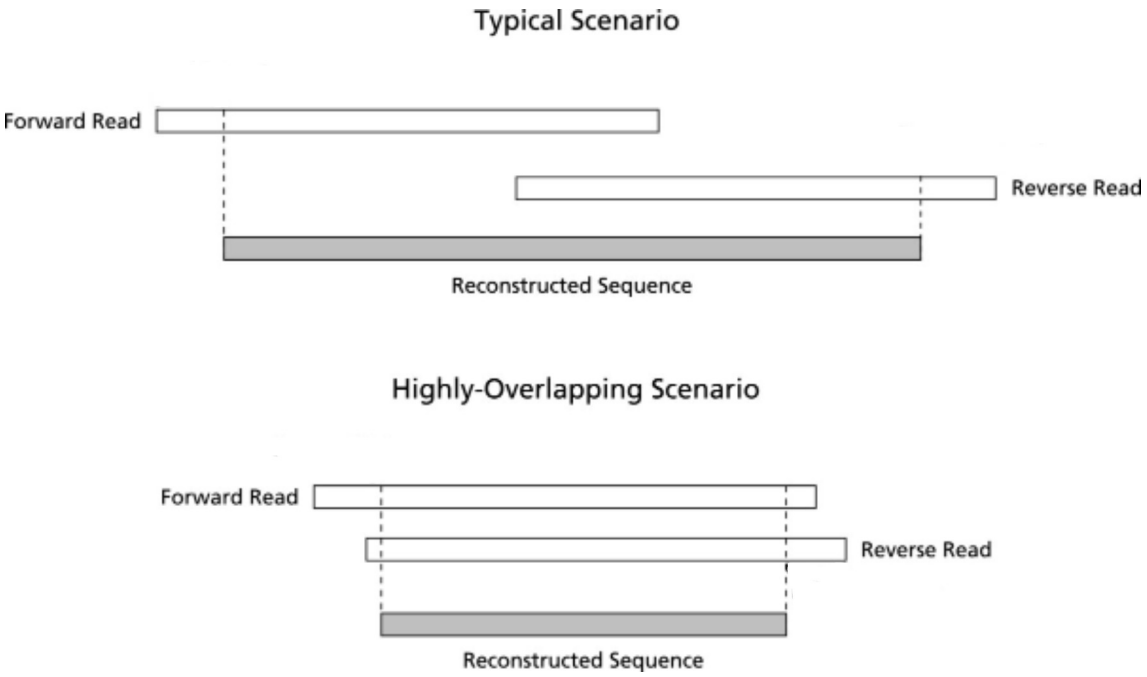

Figure 2.2: Paired-end reads overlap merging (from (Masella *et al.*, 2012))
Two typical scenarios for paired-end sequencing, one with a shorter overlap zone, the other one in which the overlap reaches the read in its entire length

length with minimal confidence for such a 'good' merge to happen (Magoč & Salzberg, 2011).

Different scenarios can, however, arise regarding the overlap, as depicted in the figure 2.3.

Another aspect that affects paired-end merging is that, in Illumina sequencers, the overall quality of each read tends do degrade as we walk towards its end (Fuller *et al.*, 2009), an effect due to the 'Sequencing by Synthesis' principle. Since each DNA molecule acts as a template for the synthesis, and the molecules are read in clusters, as more bases are added to the growing copy on the flow cell, there is a desynchronization between all the growing copies in the cluster and it is difficult to get a steady signal from the entire cluster (see fig.˜2.4).

Another common need that occurs in dealing with amplified DNA is to remove **chimeras** (Haas *et al.*, 2011) which are erroneous copies that came from the DNA polymerase chain reaction (PCR). What happens is that two or more copies from the same fragment can merge together contiguously and form one single sequence.

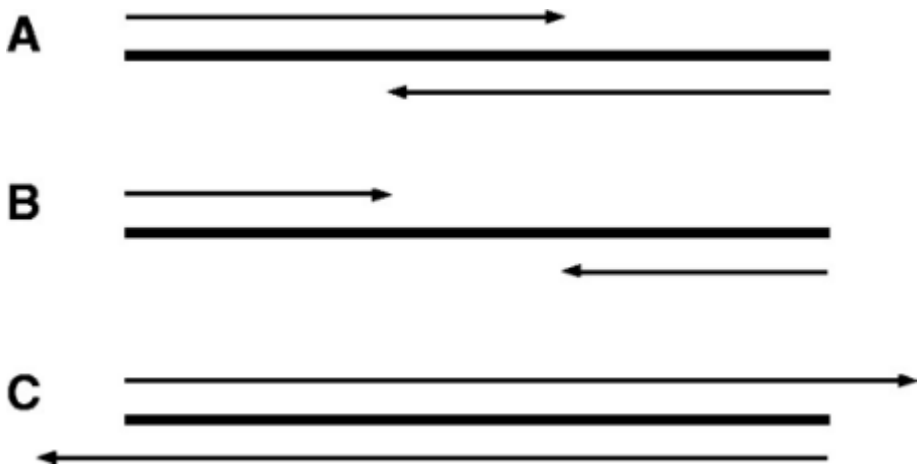

Figure 2.3: Three different scenarios for the overlap on paired-end reads (from (Zhang *et al.*, 2014))

(A) the most common scenario - a good to total overlap , in which we can achieve a favorable merge, (B) no overlap at all, where it is impossible to merge the reads or (C), the merged fragment is shorter than any of its originating sequences: not frequent, but the merge should be dealt with special care

In the downstream processes this can have the undesired effect of some of these chimeras be assigned wrongly to some *taxa*, which in reality are not present in the sample.

2.4 Clustering into Operational Taxonomic Units (OTUs)

After the merge is complete, the merged reads' (*contigs*) dataset are submitted to a 16S pipeline tool which execute the essential step of clustering the reads in OTUs. In order to belong to a OTU, there should be similarity defined in terms of the nucleotide composition above a defined threshold (the most used value is 97 which represent 'genus-level', being 99% the 'species-level'(Huse *et al.*, 2010)) . It is expected that these similarity values correlate with a phylogenetic closeness (Konstantinidis & Tiedje, 2005, Schloss & Westcott (2011)), although multiple copies of the same gene - which can potentially diverge between them - in the same organism can affect the accuracy of such affirmation (Větrovský & Baldrian, 2013). These OTUs can be already known at start, being stored in the 16S gene

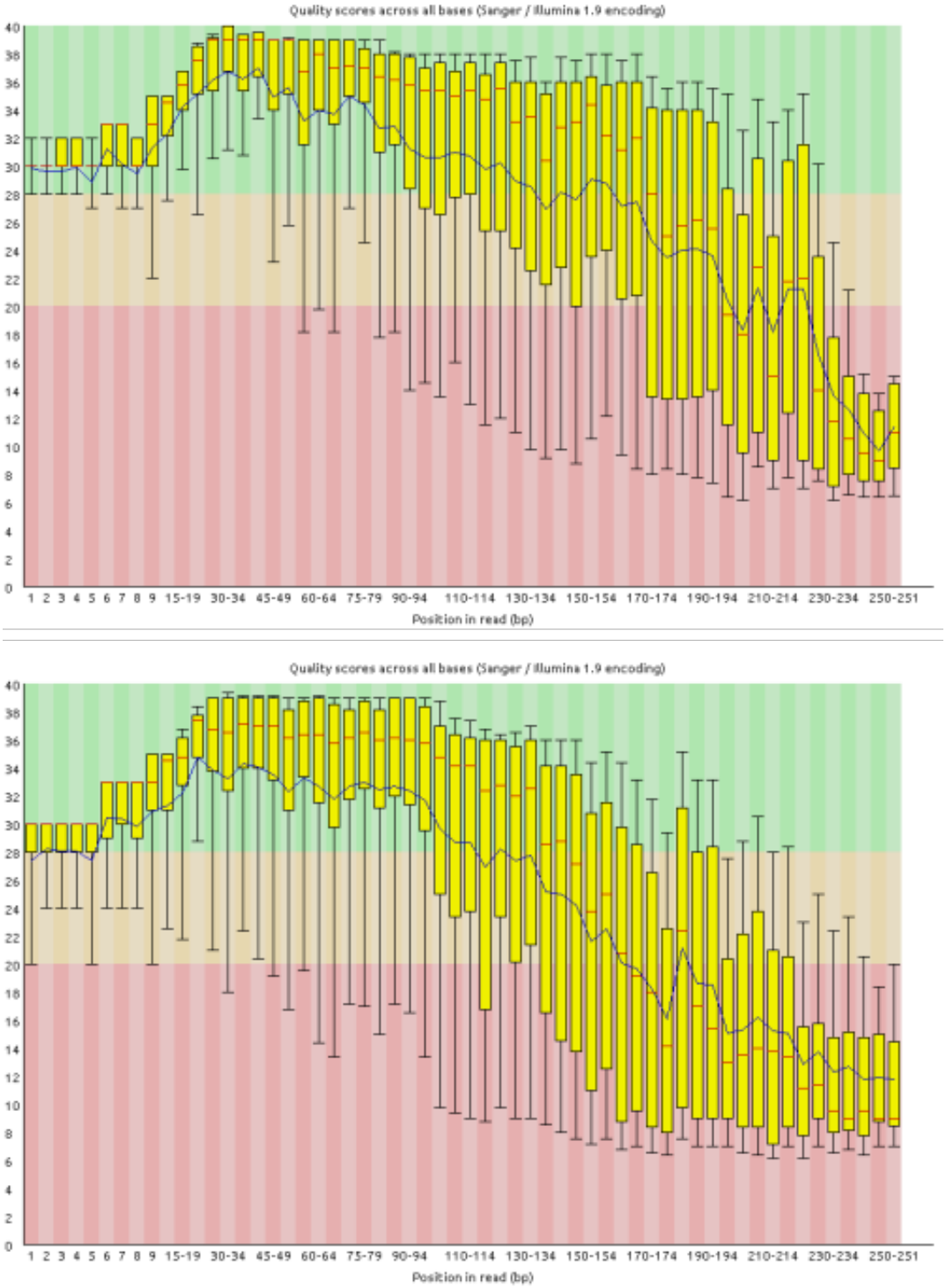

Figure 2.4: An example of quality scores boxplots

For each read pair per sequence position (forward top, reverse below, as generated by fastqc). As can been seen, the quality degrades more quickly in the reverse read than in the forward one

reference databases. For each one, a representative sequence is elected, usually the most frequent (or its multiple alignment template) is used as the reference to be collected into the database.

The OTUs besides being stored in a database, can be created "on the fly", at the moment of the pipelines execution, using a method called *open reference picking* (see Westcott & Schloss (2015),Kopylova *et al.* (2016) and Rideout *et al.* (2014) for a discussion of benefits or disadvantages of each one): while the open method can take more resources than the closed (or 'reference-guided') method, it can be beneficial since new, totally unknown 16S gene sequences can be accounted for the first time. There is still a third method: which is called *de novo* , which uses at start the closed reference method and then uses the clusters which failed to found any closest reference and are then tested in a next step, which will use the open reference.

Chapter 3

State of the art

Here we present in no special order the software tools used in this work, respecting the workflow order, so we will start with the trimming, followed by mergers, pipelines, 16S databases and finally the original Unifrac algorithm.

3.1 Trimming

3.1.1 The algorithm used by `seq-tk`

`seq-tk` was the tool used for the trimming, its algorithm (see section "The effects of trimming" on the next chapter) is called 'Phrap', which consists in a modified Mott algorithm which uses the quality scores information to purge out the low quality positions (Del Fabbro *et al.*, 2013). Just to say that each sequence is processed in an individual manner, independently of the others .

The algorithm consists of the following steps:

- A cutoff value, C , (lets say, 5%) is selected
- The associate error probability associated with the quality score, calculated as $P_{error} = 10^{\frac{Q}{-10}}$, is calculated for every base.
- The result is then subtracted from the cutoff value, $T = C - P_{error}$.
- If $T > 0$, we retain this base position , but if T is negative, we assume $T = 0$, and this base is disregarded.

- The score in the next base position $i + 1$ is then evaluated in the same manner, but this time we calculate $S = T_i - T_{i-1}$.
- If S is then negative, the base position is thrown out .
- The steps above are repeated for every read base until the end of the read.

3.2 Mergers

There are a variety of merging tools available for performing paired-end merging. Here we selected seven of these tools which operate under the same principle: find out the overlap between the paired-end reads, and discover the most probable overlap through sliding window methods by computing a score for each candidate overlap.

3.2.1 BBMerge

`BBMerge` (Bushnell & Rood, 2015) is a merger used to merge paired-end reads. Its authors claim that obtained less false positives than other mergers. In a performance comparison with other mergers, their authors showed that it can scale in CPU cores better than other mergers. The data which was used to demonstrate this result was, however, not from a real dataset, but from a simulated one.

It uses some parameters, while not providing directly ones like the overlap or read length, it has the `-maxloose` option which increases its recall, and modifies the merging rate.

The details beside this tool, were however not published until this moment. On the only information source available, it was not possible to find any details regarding its merging algorithm. The source code were, however, released as open source, and included in the bioinformatics tools package called `BBmap` (Bushnell, 2016).

3.2.2 Fastq-join

`Fastq-join` belongs to a suite of another sequencing utilities called 'ea-utils' (Aronesty, 2013). Its algorithm is described as follows:

1. start assuming that L is the overlap length and d the Hamming distance[1], between the two-paired end reads
2. start with a minimal L, e.g. $L = 10$.
3. we can assign a parameter p, which is the maximum limit for d
4. it is assigned a score $s = (d^2+1)/L$, and respecting $d < p$.
5. repeat from step 2 using a longer L up to the maximum length of both reads, and assign a s to it, this way, all potential overlaps are scored .
6. when the loop (2. - 5.) finishes, it is selected as the best overlap the one that reached the lowest s .

We want to minimize s, so, according to this equation, having more mismatches means however an higher score, and dividing it by the read length, this means that longer overlaps with less mismatches are this way selected over the ones with shortest overlaps but with a lower number of mismatches. There is, however, something to note for: the number of mismatches is squared (d^2), in contrast with L, which is not squared, so these two put as a ratio will penalize the mismatches more than a longer overlap.

As the command line tool, `Fastq-join`, it uses the following parameters, beside others: the minimum overlap length (which is the minimal value for L) and the maximum percentage of allowable mismatches (which is d/L).

3.2.3 FLASH

At the algorithmic level, `FLASH` (Magoč & Salzberg, 2011) uses the ratio of the number of overlap matches and the overlap length as the score in search of the "best" suited merge.

FLASH starts its algorithm assuming that there will not be gaps in the overlap zone. Summarily, the algorithm follows these steps:

- Assumes a minimum value, o (by default, 10), for the admissible overlap.
- Another parameter M, is the maximum overlap (default 65, which is the expected overlap for HiSeq reads).

[1]The *Hamming distance* is a function of two strings of the same length that measures the number of substitutions needed to turn one into the other.

- Align the pair of reads in a way that they overlap totally.
- Repeat the following steps while the overlap zone is longer than o and shorter than M:
  - calculate the alignment length. If a `N` (an 'uncalled' base) is found in the overlap zone, it is ignored;
  - calculates a score for this overlap in the following way: the ratio between the number of mismatches and the length of the overlap (ignoring any uncalled base if it is found).

There is more parameters that can be assigned at the command line: the overlap length standard deviation, the maximum overlap length, the mismatch density, and the fragment size (the complete length of the merged read).

3.2.4 Mothur

`mothur` (Schloss *et al.*, 2009) is an open-source data set processing multi-purpose tool able to analyze 16S sequence datasets . Besides other features, mothur has the command `make.contigs` which allows the merging of paired-end reads. It takes takes in consideration the quality values for each base position, having the following approach:

- it verifies between the two paired-end reads where a potential alignment can be,
- if there is a mismatch, the posterior quality values are assigned in this way:
  - if in one of the positions the difference between the quality values are higher than 6, it "wins" the base that has better quality,
  - if is lower, in that position is placed a `N`.
  - if the quality differences is higher than 25, it assigns a gap to that position.

Unfortunately, regarding the mothur's merge algorithm there is not much information available, all that was possible to found was gathered from its website.

3.2.5 PandaSeq

`PandaSeq` (Masella *et al.*, 2012) is, along the two next two mergers, one of the three mergers that possess a statistical approach. It needs some parameters like final length of the merged read and the minimum overlap. Among all mergers, is one of the few that takes into consideration the quality information stored in every sequence position to calculate the posterior scores, which are the values of quality that will be stored in the merged read.

This way, the posterior probability after merge is computed following these rules:

- the regions outside the overlap are copied to the final read and their quality value is simply the products of that position scores being correctly called;
- in the overlap zone the process is more complex and is separated in two cases:
  - if the bases coincide, the quality is calculated as being the conditional probability of happening a real match based on an observed match,
  - on the contrary, if there is a mismatch, the base with the higher quality is chosen, being the posterior score of the conditional probability, having a mismatch being observed, if a real match occurs.
- in the case if on the overlap region some uncalled bases appear in just one or both reads, it is stipulated that there will always be a match, and it is placed there a `N` .

3.2.6 PEAR

`PEAR` (Zhang *et al.*, 2014) takes into consideration all the possible overlap possibilities in the reads, even ones like the one already depicted in figure 2.3, situation (C), arguing that the other mergers aren't able to cope with that particular scenario. PEAR calculates the qualities in the overlap zone distinguishing four possibilities:

- the bases are known and are the same: the base is inserted in its position and the quality is the product of the initial probabilities since their errors are independent.
- the bases are known but don't coincide: PEAR chooses the base of higher quality using its value as the posterior quality.
- one of the bases are unknown: PEAR chooses the known base and its quality
- both bases unknown: in this position is placed a `N` and used for quality the lowest possible value.

The PEAR special feature is its statistical test, which is executed after the merge, inspecting the statistical significance of the merge process: in proper terms, it measures the probability of the null hypothesis that the two reads are purely independent of one another, or, in other words, it tests if the merge was just a result of chance. By default, the cutoff value is defined as 0.01. Lowering this value will improve the precision, at the price of a lower recall.

3.2.7 USEARCH

`USEARCH` (Edgar & Flyvbjerg, 2015) is also a pipeline, which includes, besides another features, a merger with low False positive rate (FPR). It states that there is an expected value E for errors on each read, being this value useful for filtering reads which can have an error value higher than this E threshold.

USEARCH takes $E=1$ by default. E can be estimated in a simple way by the sum of error probability for all the positions of the read alongside its entire length. An higher value for E can be used, and in this way the precision will be increased at the price on recall decreasing. The probability distribution of error values, specially in the matter of the distribution tails can be of the highest importance in the *taxa* abundance computation, something which is of major importance in metagenomic studies.

3.3 16S Analysis Pipelines

In this section are described the 16S analysis pipelines used in this work. The mergers and pipelines described here were selected based in their popularity.

3.3.1 mothur

`mothur`(Schloss *et al.*, 2009), besides providing the merging of paired-end reads, also provides many other functionalities for manipulating sequencing reads, like dereplicating, aligning to a multiple alignment template, restriction to a segment on that template, and removing chimeras. It advises for the use of Silva as its reference. Its processing times can be severely decreased if there are multiple processor cores. The main purpose of the tool is to reveal OTUs on the merged paired-end reads datasets, classify it and count their abundances. `mothur` will do a `de novo` clustering performing it with three different possibilities: *nearest neighbor*, *furthest neighbor* or *average neighbor* .

3.3.2 QIIME

`QIIME`(Caporaso *et al.*, 2010b) stands for *Quantitative Insights into Microbial Ecology* and is a package of different commands, presenting besides other features, the screening of a dataset, demultiplexing in case the lab samples were multiplexed with barcodes on the sequencer running. It comes with the Greengenes (GG) database as the default 16S reference. The QIIME command "join_paired_ends.py" uses the already described fastq-join merger to perform the merge.

The QIIME staff recommends for the clustering the command `pick_open_reference_otus.py` (which uses the third-party tool UCLUST (Edgar, 2010) for that), which will create OTUs based on the existing sequencing data without any *a priori* knowledge regarding the community being sequenced.

3.3.3 USEARCH

`USEARCH`(Edgar, 2010) , besides its capability as a merger, is able to remove chimeras, dereplicate, cluster, and provide also taxonomic classification. It recommends the use of RDP as the reference claiming that the other tools or references create superfluous OTUs, in number much greater than the one expected in accordance with the control.

3.4 16S Databases

3.4.1 Silva

Silva is the most recent of all 16S sequences databases, created as recently as 2007 (Pruesse *et al.*, 2007), and is the one that possesses more references and the most currently updated. As of 2016, it more than 600 000 sequences for the SSU. It is the recommended reference used by mothur.

It also features sequences for the Ribosomal Long Subunit (LSU) for prokaryotes, keeping also sequences for the homologous ribosomal subunits of eukariotes. The majority of this sequences are retrieved from the European Microbiological Laboratory (EMBL) archive data and collected through the *Arb* software suite (Ludwig *et al.*, 2004). It possesses two versions for the 16S sequences: the *Parc* version, which possesses almost 2 millions of sequences and the *SSU Ref* which at time of its last release in 2015 it stores roughly one million and an half 16S rRNA sequences, with quality filtering (Santamaria *et al.*, 2012). In this work we used the *Ref* version.

3.4.2 Greengenes

Greengenes (GG) provides sequences for Bacteria, Archaea and Fungi. It was created on 2006 (DeSantis *et al.*, 2006). In his last release, on 2013, it encompassed more than one million sequences. Unfortunately, after this last release, it has not known yet any new releases. It is the database used by default by QIIME.

Besides storing rRNA sequences, it also provides its own taxonomy, being open to proposal of new *taxa* to classify new sequences. Existing taxonomies can be imported into the Greengenes taxonomy, and chimeras and other low quality sequences are kept out of the database through tools like ChimeraSlayer and UCHIME.

3.4.3 RDP

RDP stands for *Ribosomal Database Project* and is the oldest of all 16S references, with its first version launched in 2001. As of 2013 release 11.1 there were collected

more than 2 millions sequences from Archaea and Bacteria (Cole *et al.*, 2014).

In the same way of its "competitors", it also provides on-line tools for retrieving and aligning sequences. The most recent release (11,update 4), in 2015 has more than three million 16S rRNA sequences. It is the recommended database to be used with USEARCH.

3.5 Unifrac

Unifrac (Lozupone & Knight, 2005) is a metric created to compare the biodiversity between two communities, a concept that is also called *beta diversity*. It compares the phylogenetic trees from both communities and computes the ratio of branches which are unique to each tree to the total diversity in both (see eq. 1). In this way, it allows for the multiple comparison of many communities since it works like it were a distance (it's positive, it is transitive and it satisfies the triangular inequality).

The **weighted** (or *quantitative*) Unifrac is calculated as:

$$\frac{\sum_{i=1}^{N} l_i |A_i - B_i|}{\sum_{i=1}^{N} l_i \max(A_i, B_i))} \qquad (1)$$

, being A_i and B_i equal to 0 or 1 if some *taxa* and its descendants are absent or present on the tree of the communities A and B, respectively, N the total number of *taxa* present on both communities, and the l_i's the branch lengths of the tree. The abundances are represented by l_i.

If we assume that all l_i's (the branch lengths) are the same, we have the **unweighted** (or *qualitative*) Unifrac, which is synonymous with the Jaccard distance, eq. 2 , being $J(A, B)$ the Jaccard index.

$$d_J(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \qquad (2)$$

Chapter 4

Evaluating Paired-end mergers

In this chapter, the first evaluation step is described in detail. We introduce the two datasets used, the evaluation pipeline used for the mergers and explain the evaluation method used.

4.1 Experimental design

4.1.1 Datasets used

For evaluating the paired-end read mergers two different datasets obtained from mock communities where used, one that we'll call `Mock1` and the other `Mock2`. Both datasets were obtained using an Ilumina MiSeq sequencer.

4.1.1.1 Dataset Mock1

The first community is constituted by 21 species (Kozich *et al.*, 2013), (see phylogenetic tree on fig. 1 on appendix) all in identical concentrations . The reference were adopted from the Human MetaGenome Project (HMP). This mock community was already used in other studies, see (Callahan *et al.*, 2016) , (Mysara *et al.*, 2016) or (Edgar & Flyvbjerg, 2015) .

In the sequencing run that has generated the dataset, `Mock1` achieved a total of 477 675 pairs of paired-end raw reads with 250 nucleotide positions each, covering the hypervariable regions HV4 and HV5.

4.1.1.2 Dataset Mock2

The host institution in which this work has taken place, Instituto Gulbenkian de Ciência (IGC), has created its own mock community from 8 bacterial species (see appendix) with the objective of using it as a positive control. The main difference relatively to the last dataset is that has different concentrations between the species used (see table 4.1 and fig. 2 on appendix page 60) .

This dataset has 14832 pairs of paired-end raw reads, since it was produced from a DNA with the purpose of infer the error rate in a sequencing run, as it was included with other libraries.

Table 4.1: `Mock2` *a priori* known concentrations

Species	Rel. proportions
Dorea longicatena	1
Desulfovibrio vulgaris	10
Escherichia coli MG1655	100
Eubacterium rectale	1,000
Bifidobacterium longum	1,000
Anaerotruncus colihominis	10,000
Roseburia intestinalis	100,000
Anaerostipes caccae	1,000,000

4.1.2 The Mergers Evaluation pipeline

This pipeline is composed by the steps are depicted visually on the fig.~4.1.

Below are presented the steps. Each one is subdivided in substeps, respecting the initial numbering used in the last picture.

- BEFORE MERGE STEPS:
  - 1.1. Cleaning of shorter reads, also with truncation of portions with too much 'uncalled' bases (`N`),
  - 1.2. Removal of replicates of both reads of the same pair in case one of the pair reads was excluded in the previous step

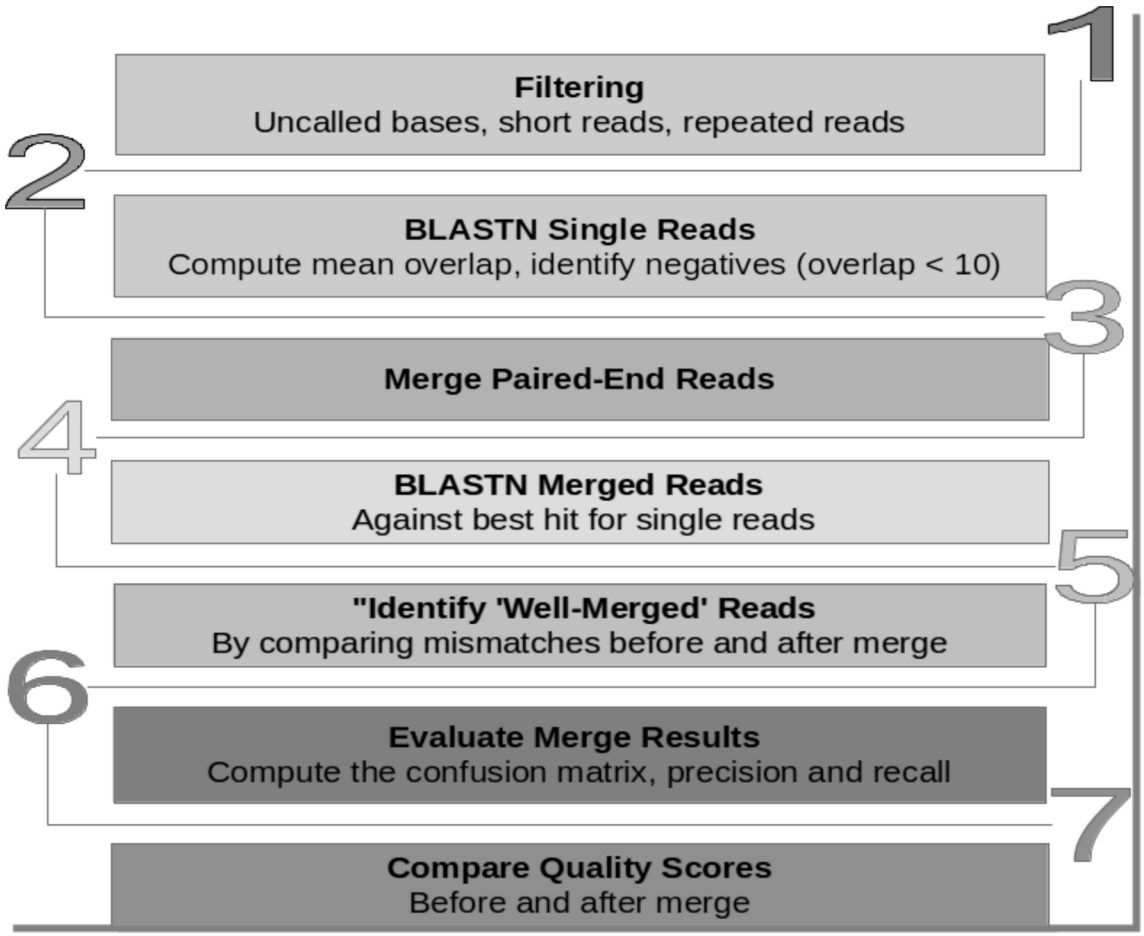

Figure 4.1: Mergers evaluation pipeline

- 2.1. Submit each read from the same pair to a BLAST against the reference
- 2.2. the BLAST results are analyzed and each reads' pair is assigned to an organism that maximizes its identity
- 2.3. the 'surviving' reads' pairs are used to estimate a distribution of potential overlaps
- 2.4. this overlap distribution are used to create two sets, one with potential negatives (reads that *should not* be merged) and another with positive reads
- 2.5. the pairs of reads with potential overlap shorter than 10 are excluded
- 2.6. The sequences on the reference belonging to each identified organism found in the step 2.2 are used to generate BLAST databases which will be used later on the BLAST after merge, in order to speed up processing

• AFTER MERGE STEPS:

- 4. The merged reads are sent for BLAST against the reference only for the 16S gene sequence for the organism which was identified before as the one which maximizes the identity (step 2.2)
- 5. The BLAST results provide the number of mismatches. Comparing the number of mismatches after merge with the ones before merge we can say if the merge was correct or incorrect.
- 6.1. The count of 'good' merged reads (for each merger) is calculated and used to compute true and false positives or negatives
- 6.2. These counts are used to compute the precision and recall
- 7. The quality scores of the sequences before and after the merge are assessed to verify if their values were improved

4.1.2.1 The 'before-merge' steps in detail

The datasets are submitted to a preliminary process which performs a basic cleaning on the dataset:

- detects the replicated reads on the dataset
- truncates the reads possessing 'N' at their extremities
- If the 'surviving' subsequences in the trimmed reads are less than 20 bp in length, these are removed from the dataset.

After this preliminary step, the whole dataset is scanned for reads that were 'orphaned' (i.e., the other read in the pair was removed in the previous step). If found, these 'orphaned' reads are also removed. At the end of this step, there must be no such 'orphaned reads' remaining in the dataset.

Then, and in order to prepare for the BLAST, the reads are stored in one single file, with the paired reads in consecutive order (remember that before the start of the evaluation pipeline the paired-end read datasets are stored in two separated files, one for each reading direction).

Having all the reads stored in the same file allows for the BLAST executable to read all the reads in the same loop in a predictable way, which allows the comparison of the ensuing results in this precise order. In order to exploit the multi-core nature of CPUs a parallelizer were used in order for the BLAST process to use all those CPU cores. This way, the BLAST processing times (which takes the major fraction of entire merge evaluation analysis) were in great matter reduced.

BLAST produces an output file in tabular form which we use for further processing. Each row in this file is a High-scoring Segment Pair (HSP) and for each one of these HSPs, BLAST calculates some parameters like the bitscore, the e-value, but the important parameters are: *subject* (the name of the organism to which belongs the sequence that was aligned to) , *number of mismatches*, *identity percent*, *alignment length*,*start position* and *end position* (we will call these last three parameters the *alignment coordinates*) .

For the BLAST processing, we used as the initial arguments a 95% of sequence identity minimum for the forward read, with a threshold (90%) for the reverse

read. We stored every HSP that presented starting with the maximum identity until the one that presented lower identity value, but above the threshold. Sometimes it could happen that the most favorable HSP for each read of the same pair are from different subjects (which is, from a 16S gene in different organisms). In order to deal with this situation, we formulated the following *algorithm* :

- for every BLAST hit for some paired-end read:
  - store the subject and the bitscore
  - if the subject is already known, add the bitscore value to the variable *sum* for this subject
  - if the subject is not known, create a variable *sum* and assign the bitscore value to it
- at the end of the loop, verify which subject has the higher value of *sum* . The HSP with the maximum value for *sum* is chosen as the most suitable to be used and its alignment coordinates are used to calculate a *putative overlap* as the difference between the forward and reverse *end positions* assigned by BLAST (see fig. 4.2) .

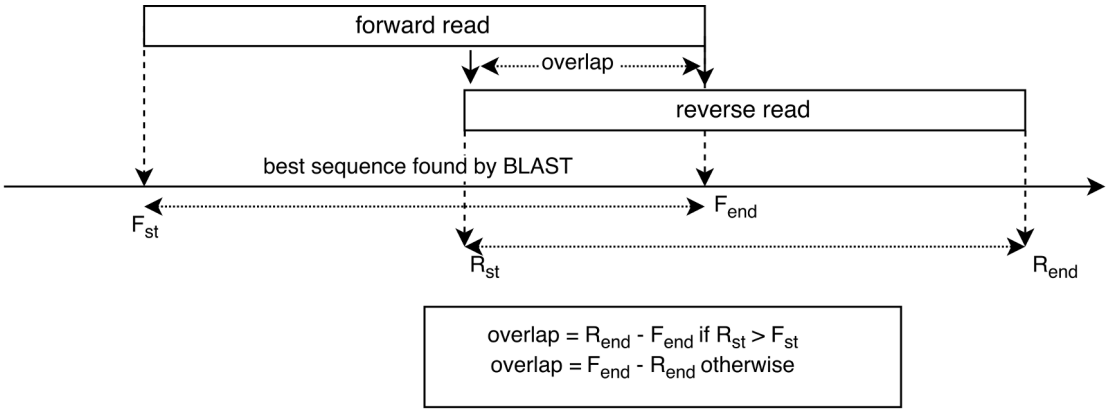

Figure 4.2: Calculation of the putative overlap

Using the alignment coordinates found by BLAST for each one of the reads of the pair we can calculate the putative overlap F_{end} and R_{end} here are the end position coordinates for the forward and reverse reads, respectively. Normally we expect $F_{end} < R_{end}$, but in the case the opposite happens (a "towards-left" alignment), we reverse the order of the values on the subtraction

The *putative overlap* value can assume negative values which means that there isn't any potential overlap in the paired reads. The reads with positive values are assumed to be potential *a priori* 'well-merged' reads. This way, we create two sets of paired-end reads, one for reads with potential *well merged* reads and other one with *not well merged* reads, which doesn't have any potential overlap. As a side note, we also included into the *negatives* set the read pairs which had as its potential overlap less than +10.

After we have concluded filling these sets, we collect the names of all organisms in the reference which had got at least one match in the BLAST. Then we use the 16S sequences from the reference from each one of these organisms to create a subset of the main reference with just the sequences from the matched organisms.

4.1.2.2 Difficulties that arise from the 'before-merge' BLAST

Some read pairs had to be excluded over from the evaluation pipeline during the course of its execution. Those reads were:

- had not got any hits from the BLAST (we call them 'missing' reads)
- one of the reads from the pair got BLAST hits, but not the other one (these are the 'unpaired').
- both reads from the pair got hits, but these ones come from completely different organisms ('mispaired' reads).

The fig. 4.3 shows how these reads are interrelated. In the end of the evaluation pipeline, only the pairs of reads that got a suitable alignment with the reference and got to be identified as belonging to the same organism were considered.

4.1.2.3 Mergers Execution

When the 'before-merge' steps are complete, we are ready to run the mergers. These are invoked in batch, and each one is executed two times, first with the default configurations and the other with adjusted parameters, as explained in a next section.

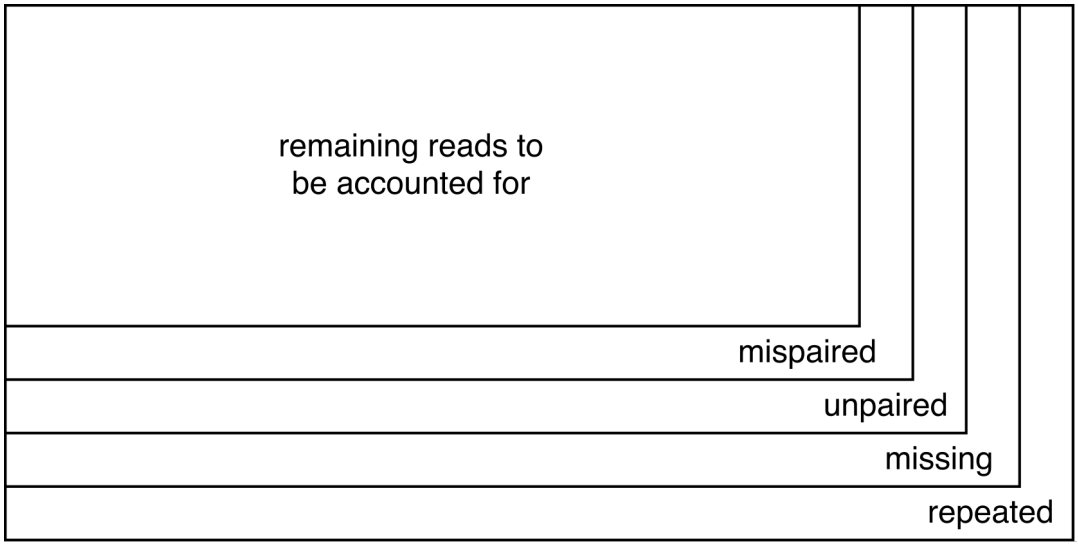

Figure 4.3: How the reads which were excluded from the pipeline are related
Some reads had to be excluded from the final accounting on positives and negatives, due to implications related to BLAST

4.1.2.4 Parameter configurations

The mergers possess many different parameters. The most frequent where the shortest length admissible for the merged read, or the minimum and maximum for the overlap length, what to do with reads with uncalled bases, and the number of processing threads. For each merger, two different combinations of parameters were tested, the parameters' default configuration used by each merger and a different one with parameters adjusted specifically for each dataset. At the end of the before-merge steps on the evaluation pipelines, the maximum overlap is adjusted according to the maximal potential overlap found estimated according to the data provided by the BLAST.

The different parameters tested in each configuration are shown in the table 1 in the appendix.

4.1.2.5 After-merge evaluation steps

After merging, the resulting merged reads are submitted to a new BLAST, this time not using the complete reference to search against, but just with 16S gene sequence of the most probable organism found in the 'before-merge' phase. This

way, using just that sequence as the reference, we not only speedup the BLAST execution but also ensure that we are using the same reference and the number of mismatches that is computed from it, number that we are going to use later.

The same pair of reads will be merged again each time a new merger runs over it. This way we obtain the number of merged reads multiplied by the number of different merger runs we have. For all of these sequences in each one of these merged sequences files we'll have to do a BLAST in the conditions enumerated above.

After finishing the BLASTs, and peering over the HSPs obtained from the BLASTs, we calculate the differences between the mismatches obtained after and before the merge, one for each combination merger/merged read. If the number of mismatches for a particular read decreased, this means that the read were **well** merged, but if the number of mismatches otherwise has risen, this means for that particular merger, that the pair of reads were **wrongly** merged.

We'll now count the paired-end reads that were **well** merged and compare these counts for every merger, and describe these results in terms of precision and recall values, two metrics defined in the next section.

Now we have a set of 'well-merged' pairs, and then we use the set of positives created before and collect the paired reads shared by both sets. Their intersection may be classified as true positives. The other ones are the false positives.

4.1.3 The evaluation metrics: precision and recall

To evaluate the merging process we adopted two metrics: **precision** and **recall** .

Precision (Pr) is the ratio between the true positives and the sum of true positives (TP) and false positives (FP) and mathematically defined as:

$$Pr = \frac{TP}{TP + FP}$$

, and Recall (Rec) is the ratio between the true positives and the sum of true positives and false negatives (FN) as:

$$Rec = \frac{TP}{TP + FN}$$

As the metric that "joins" into one value these two metrics together we use the F-measure (F_1) to give the 'overall' idea.

The F-measure is defined (Zheng, 2015) as the harmonic mean of these two last values:

$$F_1 = 2\frac{Pr * Rec}{Pr + Rec}$$

.

By this equation, we'll have that when Pr and Rec are higher and very close to each other, the F_1 will be higher too, but on the contrary, if they are close but their value is too low (closer to zero) the F_1 will be lower.

Put by other words, the true positives are the reads that were correctly merged by some particular merger, the false positives the reads that were merged incorrectly, the true negatives are the reads that were rejected for merging and the false negatives the reads that were not merged but should have been merged.

These two ratios are used to evaluate the effectiveness of each merger on each dataset.

4.2 Results and discussion

4.2.1 Mock1

Before merging, the overlap length revealed the distribution depicted (see fig. 3 on appendix, page 61). Much potential negatives candidates were found, i.e., paired end reads without any putative overlap (circa 4238 from 455 000 reads on Mock1).

The values of precision and recall obtained for dataset `Mock1` are depicted on the scatterplot of fig. 4.4, with the precision on the xx axis and the recall on the yy axis . The concrete values are shown on table 2 on appendix.

Fastqjoin and FLASH achieved the best results on precision, along with PEAR and Usearch. But these last two state more balance between precision and recall, as we can see through their F_1 values. Mothur merged every read without regard if that merge suitable or not, so its recall is exactly 1.0 . Usearch was, from the mergers with best precision, the one with the recall most closer to it.

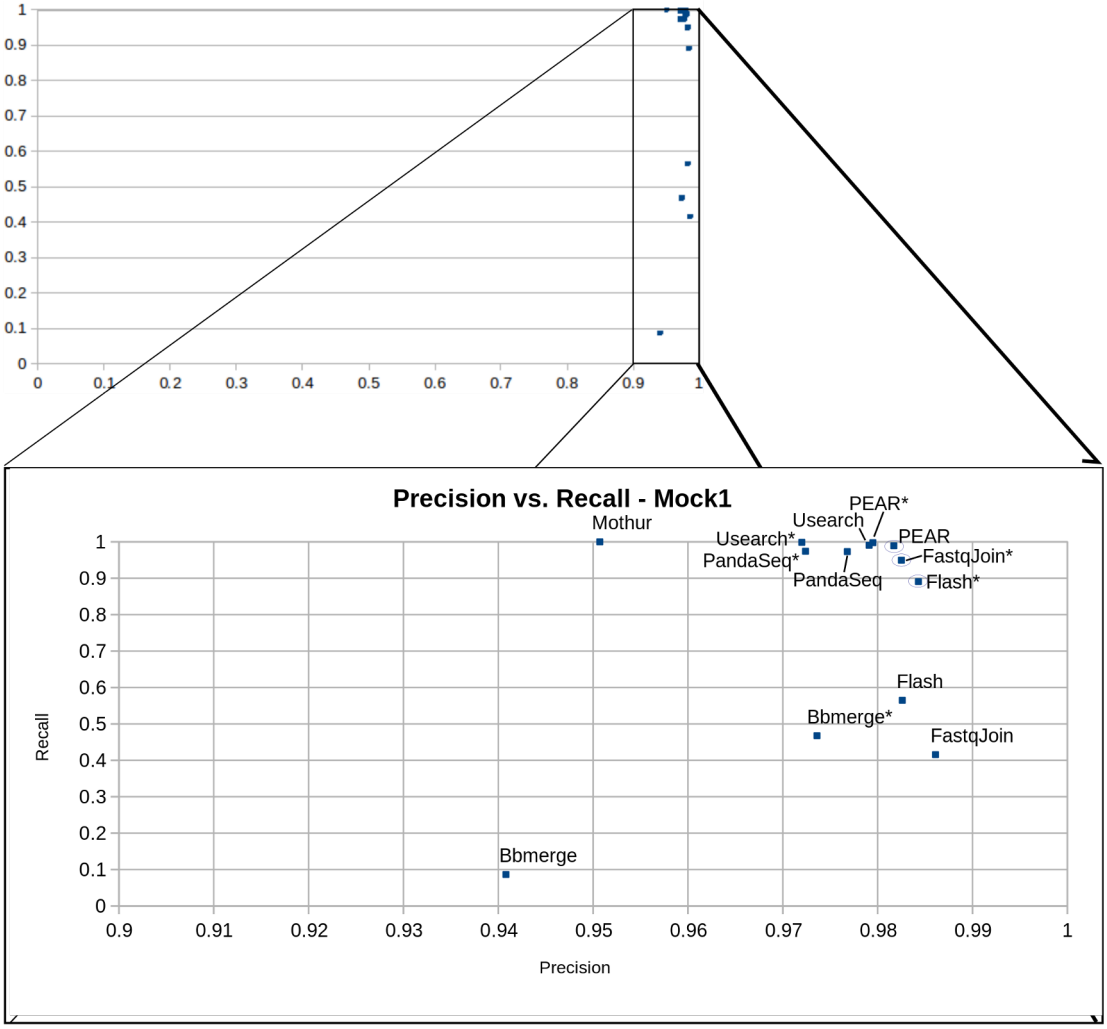

Figure 4.4: Precision-recall scatterplot for Mock1

The precision acquired all its values inside the interval (0.9;1.0) , so a zoom was made on the x axis in that range in order for the points in the scatterplot were sufficiently distanced to be seen as being apart. The merger names marked with an asterisk means an 'optimized' configuration was tested - for the concrete values and parameters used on this configurations see table 1 on appendix . The values of precision and recall are shown on table 2 on appendix.

PEAR has a statistical test that is done by default, disabling it dropped the precision with a slight improvement in the recall.

PandaSeq also achieved good precision, but specifying explicitly parameters namely the expected overlap had a negative impact instead of improving them.

Usearch achieved best precision with its default values, touching in parameters like the minimum overlap had a very small effect on improving the recall at the expense of a precision drop.

BBmerge was disappointing in `Mock1`, even when using the most optimized `-maxloose` command line option. Even in this case, its recall didn't improve that much.

There was an important effect on the recall for FLASH and FastqJoin that came from changing the mergers' parameters regarding the minimum and maximum expected overlap length. That effect didn't reflect however on the other mergers like Usearch, PEAR and PandaSeq.

4.2.2 Mock2

Regarding the dataset Mock2, the putative overlaps are shown on fig. 4. In this case, there were not found any 'negative' reads. All the reads revealed, before merging, that are, in its entirety, to be able to be merged correctly.

Regarding the precision (see fig. 4.5), practically all mergers, with the sole exception of BBmerge, performed very close to each other, all inside the (0.98; 0.99) interval, being FLASH with the custom configuration the best (0.9874), with Usearch and PEAR very close to it (0.9871 and 0.9872). These last two achieved the best balanced scores, belonging to them the highest F_1 (0.9935).

Mothur repeated its status as having the best recall, with Usearch equalizing it in this trait. BBmerge improved slightly its recall when compared with Mock1, having reached 98.3% precision and 72.3% on recall with the `-maxloose` command line option.

In some cases (FastqJoin, PEAR and Usearch) there were not any modification on precision and recall between the two different parameter configurations tested for each merger.

.

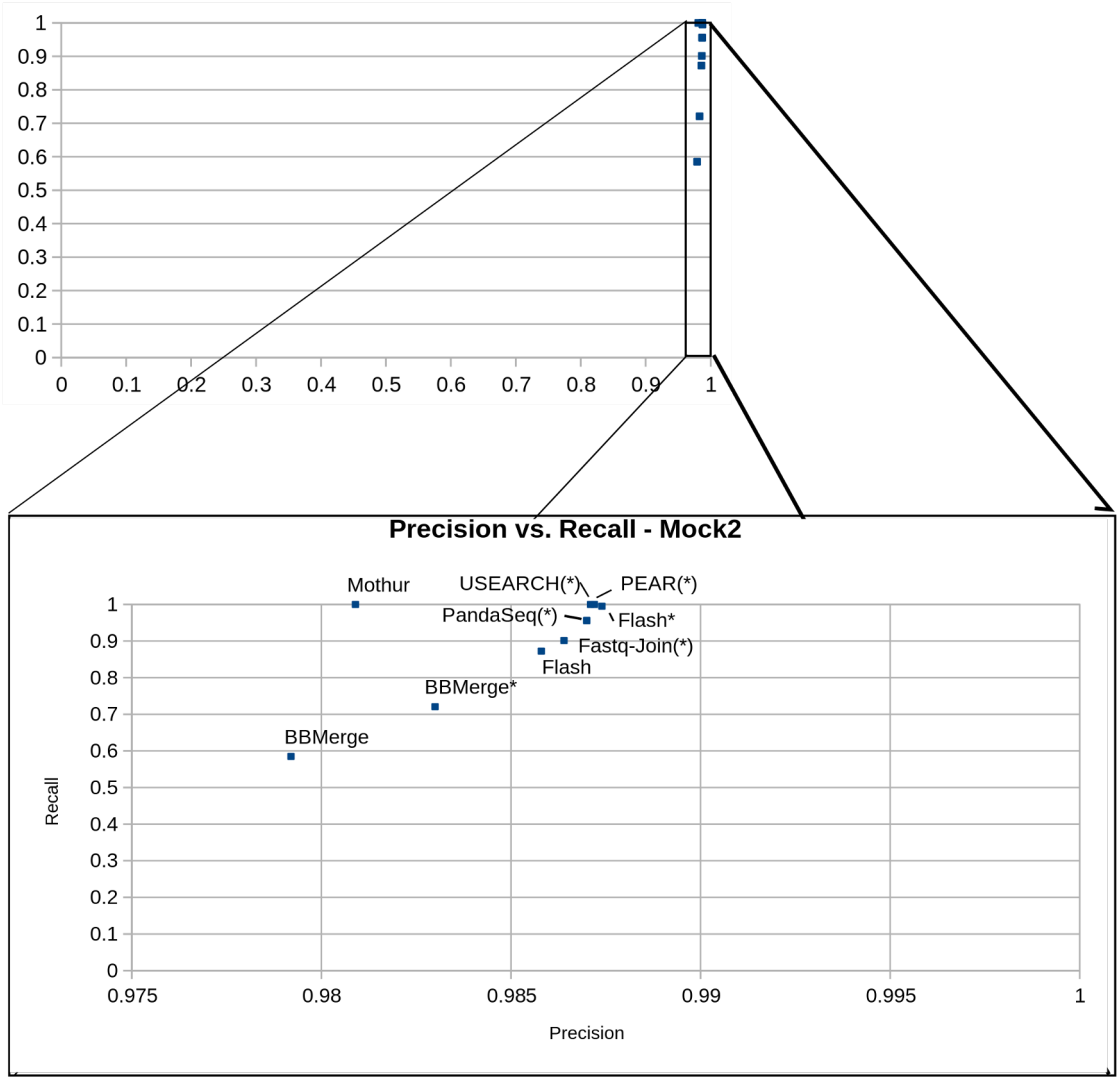

Figure 4.5: Precision-recall scatterplot for Mock2

This picture were zoomed in the interval at precision values in the xx axis in which all mergers got values : (0.975; 0.99). The mergers names marked with (*) mean that there were no relevant difference between the two parameters configurations tested. The exact values of precision and recall are shown on table 3 on appendix

The overall quality scores of each dataset (beside their lengths) are the main factor that explains the differences in these results, as we'll see in the next section.

4.2.3 The quality scores after the merge

Another variable evaluated on our study was of the way each merger computed the posterior quality scores. The results are depicted in the boxplots on figs. 4.6 and 4.7.

The difference on results between the two datasets can be imputed to the average error rates which were revealed to be very distinct between the two datasets, in accordance to their quality scores distributions . Mock1 has an higher mean error rate than Mock2, so the precision values suffer from that.

According to (Edgar & Flyvbjerg, 2015), we can calculate the expected value (E) for the error on individual base positions per read using its quality scores. This value can be calculated as

$$E = \sum_i p_i = \sum_i 10^{-Q_i/10}$$

Being p_i and Q_i the error probability and score value at a concrete position i , respectively. Averaging this value using all the reads in the dataset we'll obtain for dataset Mock1 the value $\bar{E} = 4$ and $\bar{E} = 1.0$ for Mock2. These are the concrete values for the average error per read on each dataset. Mock1 reads have four errors in the read while Mock2 have just one error in the read. So, in overall, the quality scores on Mock2 show that its quality is better than Mock1.

For dataset Mock1, we can see that practically all mergers distinguished themselves in improving the scores values, with the sole exception of PandaSeq, which worsened them. This is due to the way the PandaSeq calculates the posterior scores, which are wrong, according to (Edgar & Flyvbjerg, 2015). Mock2 reveals the same effect.

Although we didn't had any concrete way of understanding how some mergers implement their own way of calculating the posterior scores we could see some curious behavior on BBmerge that puts the median of the scores distribution in both datasets almost to the right side of the boxplot (should we consider the

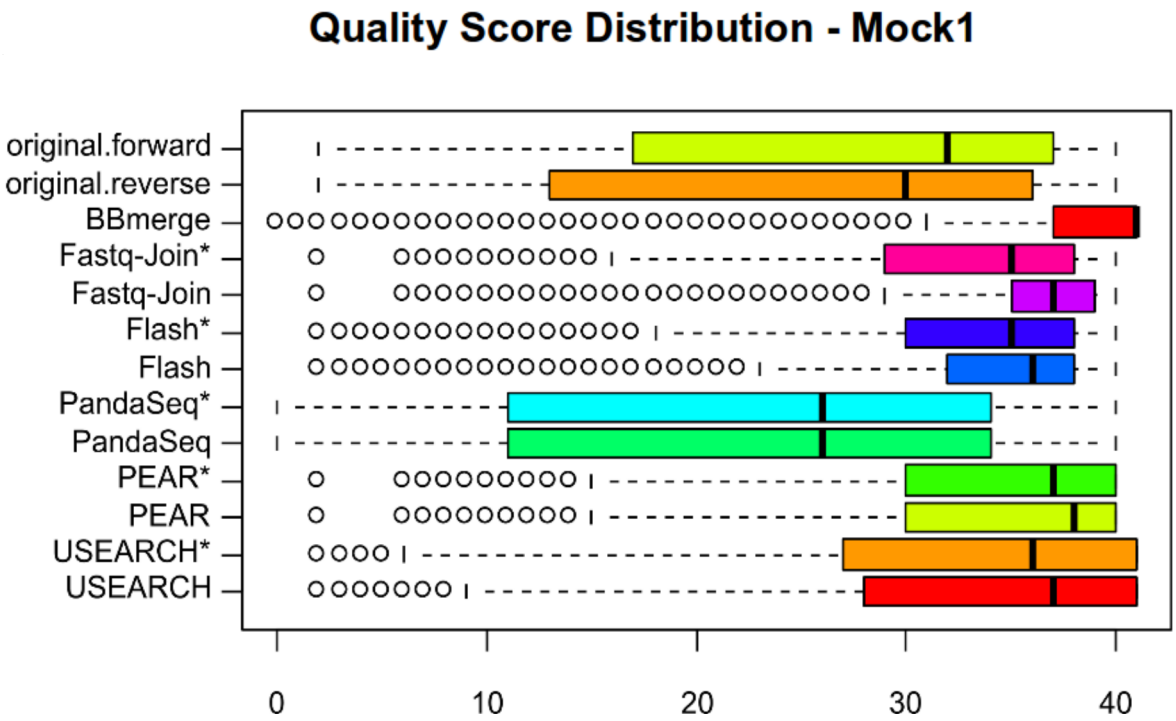

Figure 4.6: Mock1 quality scores distribution boxplot before and after merge
The quality scores distributions before the merge are the two topmost boxes (`original.forward` and `original.reverse`).

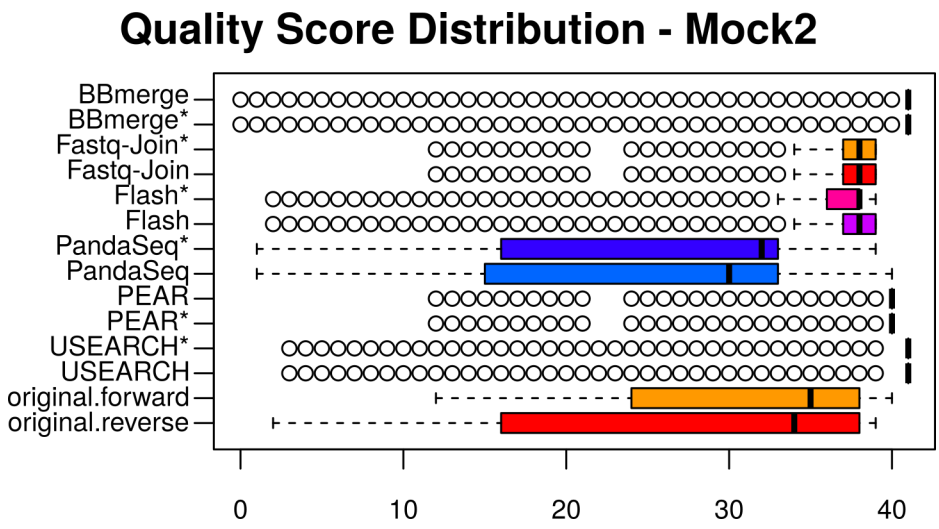

Figure 4.7: Mock2 quality scores distribution boxplot before and after merge
The quality scores distributions before the merge are the two bottommost boxes (`original.forward` and `original.reverse`)

posterior scores computation by BBmerge too optimistic ?). The other mergers were able to improve in the overall the posterior scores in general, the customized merger configurations provided more widened distributions, which is not strange to understand since their recall value was higher than the one revealed by the default configuration (see FLASH, FastqJoin and USEARCH). The effect of outliers appearing in the merged reads distributions is not something bad, but a signal that the merger were able to correctly assign the vast majority of the dataset as a whole - after all, there is no outliers to the right.

If we forget about PandaSeq, Usearch achieved the widest distribution, which by itself is a result of the way its author (Edgar & Flyvbjerg, 2015) puts importance in the tails, since it is in the tails that some scarce *taxa* can be revealed in the downstream steps ahead.

In Mock2, since the overall quality is better than Mock1 we see that some mergers "staggered" the boxplot (concretely, BBmerge, PEAR and USEARCH), which means that 98% of reads after had their scores with the same value.

Just one side note here: mothur box doesn't appear in these two boxplots because we couldn't retrieve the quality scores after merge because mothur at the last version we tested it does not produce FASTQ files as its merge output.

4.3 The effects of trimming before the merge

Due to the very nature of the way the sequencing platform works, there is a greater probability for reading errors at the reads' start and end positions. So trimming out these positions will be advantageous for the further dataset downstream processing.

With this in mind, the tool `seq-tk` was used on the raw reads at two 'trimming' levels: 95 and 99% . So we obtain two 'trimmed' datasets from each one of the raw datasets. After the trimming with `seq-tk`, the trimmed datasets were submitted to the same evaluation steps as the original datasets.

The net results are shown along with the results obtained where there were not used trimming (for comparison purposes) on figs. 4.8 and 4.9 .

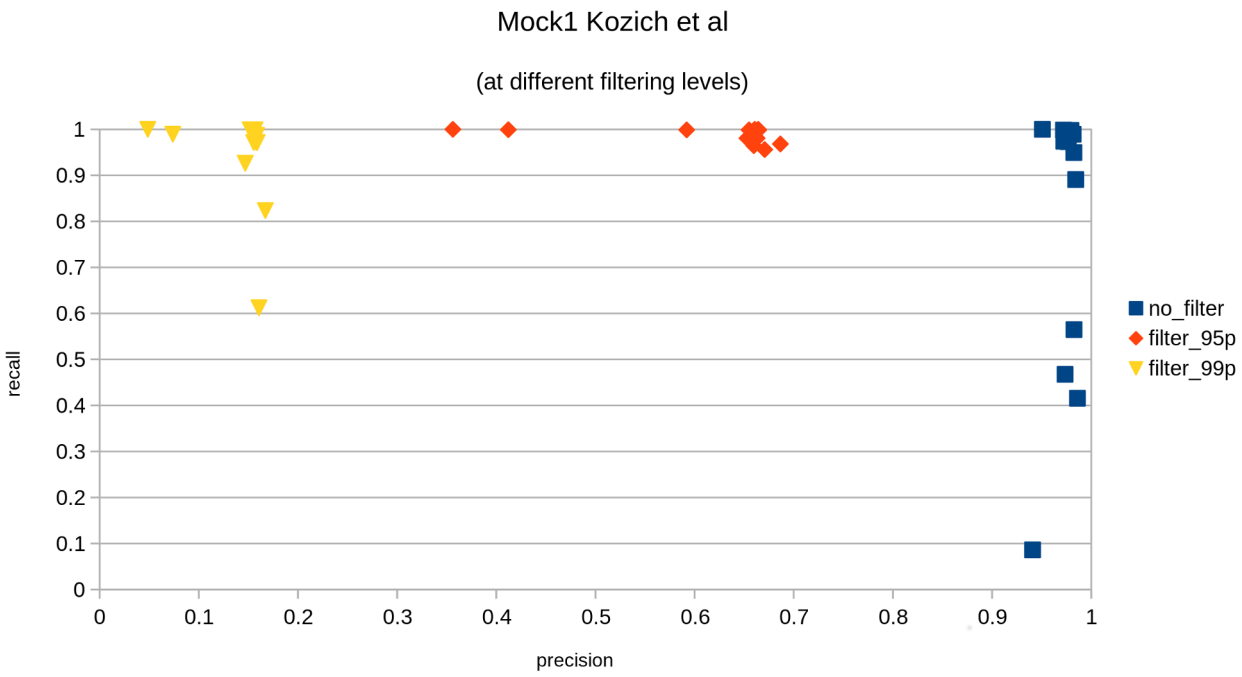

Figure 4.8: The trimming effects on Mock1

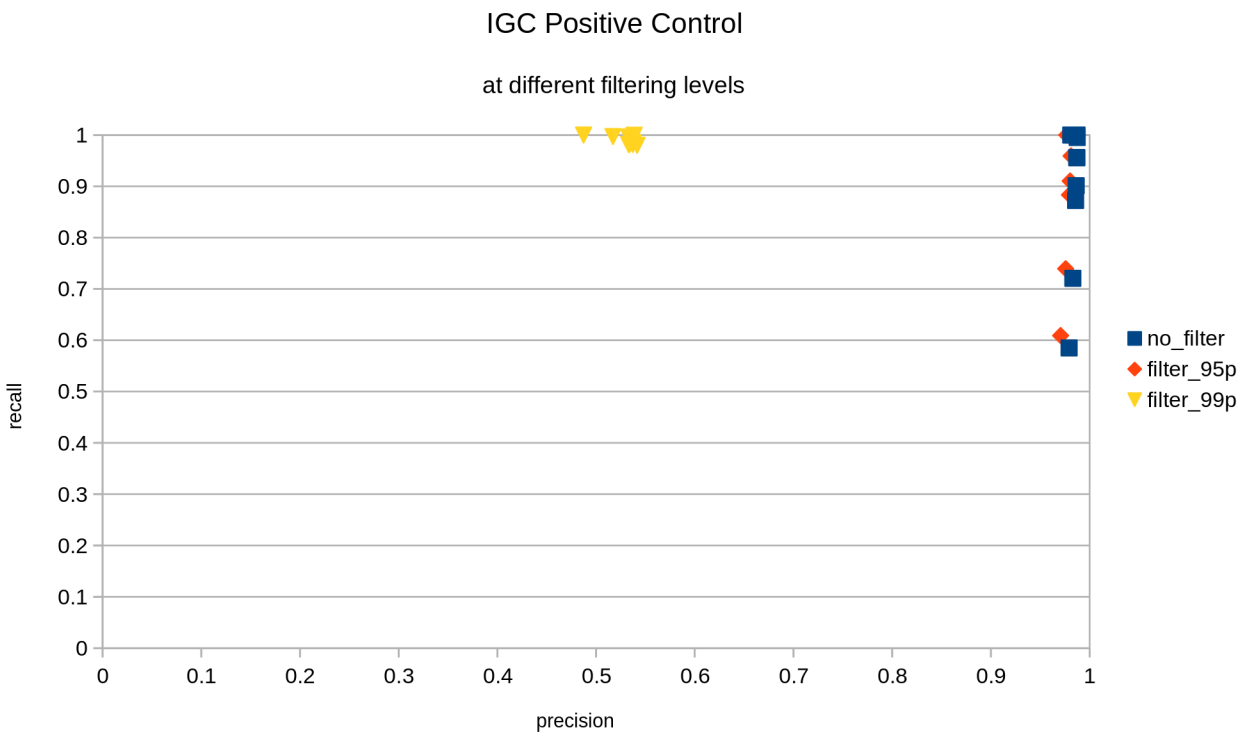

Figure 4.9: The trimming effects on Mock2

For the dataset Mock1, there is an impervious effect on the precision, although the recall improves substantially with it, with practically all the mergers reaching 1.0 . At 99% the precision is greatly reduced, but the recall is not so affected.

For Mock2 the filtering at 95% does almost nothing, but at 99% the precision is greatly reduced, with the recall however again improving almost to 100%.

We can conclude that the trimming has great effect in improving the recall. The better overall quality of the reads on dataset Mock2 is put to the test on the trimming at 95%: there were almost no change in the values of precision/recall. Only at 99% we see the precision being substantially affected by the excessive trimming.

Too much trimming is not good, it lowers the precision too much. What can we achieve with lower precision but high recall ? Even if in the end we can get all our remaining reads merged, what's the point if it is not done correctly !?

4.4 Source code availability

The source code used for the mergers' evaluation is available with the name `pe-avaliator` at `https://bitbucket.org/sviana/pe-avaliator/` .

Chapter 5

Evaluating 16S Analysis pipelines

After the initial merge, the merged datasets were submitted to the 16S analysis pipelines for the downstream steps.

5.1 Experimental design

The pipelines follow these steps: template alignment, replicates removal, chimera removal, clustering and finally taxonomic classification.

The results of clustering are stored in **OTU tables** which contain representative sequences alongside with the counts of the sequences gathered as sufficiently 'similar' with this representative sequence.

These tables, in the form of BIOM[1] files, are then purged from singletons, the *taxa* names extracted, and compared with the phylogeny from the control.

This comparison can be qualitative or quantitative and we implemented our own Unifrac version which we'll call **Unifrac similarity**, in contrast with the conventional Unifrac, which is a measure of difference.

5.1.1 The merged reads' datasets chosen

From the datasets that were the product of the mergers as detailed in the previous chapter, we chose five: three were the ones created by the mergers which had the best balance between precision and recall (PEAR, fastqjoin and FLASH) and the

[1] Biological Observation Matrix.

other two were created by the mothur's merger and Usearch's merger. We added these two datasets because we wanted to know if there was some positive effect of using these pipelines with their own mergers. Fastqjoin is the merger used by default in QIIME, but it was not needed to explicitly add it since it was already selected.

5.1.2 Unifrac similarity

The Unifrac similarity assumes values from 0 to 1, but it is symmetrical to the Original Unifrac (U). We will designate our Unifrac similarity measure by U_{sim}, being calculated as $U_{sim} = 1.0 - U$.

As in the original Unifrac, we also have **weighted** and **unweighted** versions of it, being synonymous with quantitative and qualitative, respectively. The weighted version takes into consideration the *taxa*'s abundances, and the unweighted just if the *taxa* are present or not.

Unweighted U_{sim} values closer to 1 means that the two communities are very similar to each other, disregarding the abundances for each *taxa* in either of the communities. For example, if some species has 100 individuals in the control and 1 individual on the other, a species B one individual in control and 10 in the other, the unweighted U_{sim} will be 1.0.

On the opposite side, for the weighted U_{sim} if there is difference in the population effectives of the same *taxa* between the two communities being sampled, it will be much more difficult for U_{sim} to assume exactly the value 1.0, since it is highly improbable that the two communities will share exactly the same number of individuals for the same *taxa* in them .

5.1.3 The 16S databases

As stated in the state of the art, three 16S gene sequences databases where used as reference for the three distinct pipelines we selected for our study. This resulted in distinct combinations to compare on our analysis for each dataset.

From all databases, RDP had less sequences but encompassing more high-level *taxa* . Silva has sequences that come from much more biodiversity, since it also stores Archaea 16S sequences in it. The fact that each database has its

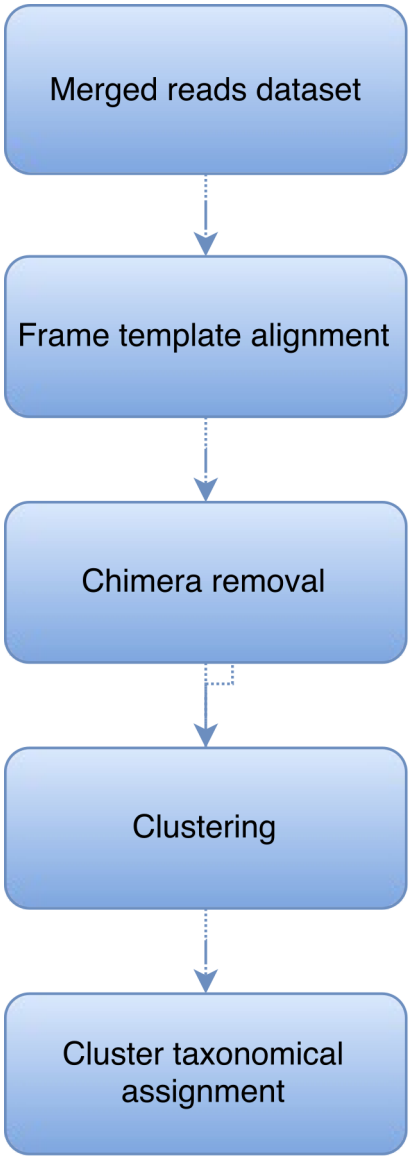

Figure 5.1: Main steps implemented by the 16s Analysis Frameworks

own taxonomy presented an additional difficulty, but if the database contained at least a mapping file that indexed each sequence with the NCBI taxonomy, the problem was solved, which were the case for all databases.

As we will see, curiously each framework presented better results when the non default database was used instead of the default one.

5.1.4 Some difficulties posed by each analysis pipeline

Regarding the user experience with each pipeline, QIIME gives you the most 'user-friendly' results. It provides graphics and charts, has a learning curve less steep, and it comes with its own reference.

mothur demands more knowledge from the user in order to give a more precise control. To obtain what was intended from it - the OTU tables -, we had to peruse through long documentation. We tried to follow, as close as possible the Mothur MiSeq Standard Operating Procedure (SOP), through all its steps.

Usearch is straightforward, it reuses the algorithms for clustering and chimera removal already conceived by its author (UCHIME and UCLUST). The merger and OTU taxonomic assignment (UTAX) algorithms were until now however not published. Only recent versions (starting with the 6.xx series) started to write OTU tables in the BIOM format, and this way it was possible to process those BIOM files in the same way as the other pipelines.

Regarding the difficulties presented by each pipeline and the solutions found to solve them, those are provided on the "Materials and methods" section A on the appendices.

5.1.5 The pos-pipelines processing

In this step, we want to process the OTU tables stored in the BIOM files. In order to calculate the unweighted and weighted Unifrac values, we must proceed to extract the *taxa* names for the first case and the names along with their abundances for the second case. But before any of these steps, we must purge the singletons out of the original BIOM files.

Singletons are those OTUs that just only got 1 read that matched them, or in other words, it's a taxa that was present with only one read on the dataset. So,

there's a high probability that this singleton was generated from a read that was a chimera or was generated 'in silico' somewhere during the pipelines workflow. From all the pipelines tested, the singleton removal was automatically done by QIIME and Usearch. Only with the BIOM files created by mothur the singletons were not already removed by the pipeline itself, so the need to explicitly remove the singletons from the OTU tables generated by mothur .

After the singletons removal, the *taxa* names are extracted and compared with the ones on the control to verify if there were matches between them. The qualitative Unifrac is then calculated in this comparison. For the weighted version, beside the names, the abundances are also needed.

Along with the U_{sim} values, we also calculated precision and recall for all the combinations. Using precision we can know how much of the *taxa* that were already present in the control were also identified as existing by the pipelines, excluding eventual *taxa* that were mistakenly identified as existing in the mock community. The Unifrac has the disadvantage that considers these spurious *taxa* in its calculation, so it is more sensitive to identification errors than the precision. But the precision does not take into consideration the inherent taxonomy, so it is a measure that is not taxonomically informed.

5.2 Results

As stated above, we submitted five merged reads datasets to each pipeline and 16S databases. The results go on the appendix. The combinations paired-end merger/pipeline/16S database which achieved the best numerical values on each dataset and Unifrac type are shown on table 5.1.

5.2.1 Mock1

For this dataset, the best unweighted Unifrac values were obtained using mothur as the merger and QIIME as pipeline, and independently of the 16S reference used. Mothur was also the merger associated with the best quantitative values, with mothur as a pipeline, but using Greengenes (GG) as the reference.

The Unifrac unweighted values for this dataset are shown on the table 5.2.

Table 5.1: The configurations that achieved the best Unifrac values per Unifrac type and dataset

Dataset	qualtitative Unifrac similarity	quantitative Unifrac similarity
Mock1	mothur →QIIME(SILVA):61%	mothur→mothur(GG):78%
Mock2	mothur→mothur(RDP):67%	mothur→USEARCH(SILVA): 68%

Table 5.2: Unweighted Unifrac similarity values calculated for dataset Mock1

Unweighted Unifrac Mock1		pipeline		
merger	16S reference	mothur	QIIME	USEARCH
FastqJoin	GG	53,57%	51,97%	50,00%
	RDP	41,89%	52,00%	47,89%
	SILVA	46,48%	49,68%	48,82%
FLASH	GG	53,96%	52,67%	52,31%
	RDP	41,89%	52,32%	50,37%
	SILVA	47,83%	50,00%	49,18%
mothur	GG	52,38%	58,91%	53,03%
	RDP	46,72%	57,69%	50,37%
	SILVA	47,59%	61,11%	50,82%
PEAR	GG	53,19%	52,67%	47,89%
	RDP	41,89%	51,97%	46,90%
	SILVA	46,15%	49,69%	48,44%
USEARCH	GG	52,48%	48,77%	47,55%
	RDP	41,06%	54,86%	45,03%
	SILVA	45,83%	51,97%	45,71%

For the weighted values, and a table with all the used metrics for this dataset, including precision and recall, please see table 4 on the appendix.

We also calculated precision and recall values to know if some OTUs were dropped on the final results and the values were well assigned. In the Mock1 case, the best precision achieved were 70%, again with mothur as the merger and QIIME as the pipeline. The higher recall achieved was 85%, using QIIME as the pipeline and Usearch as the merger.

5.2.2 Mock2

For the Mock2 dataset (see table 5.3 for the unweighted Unifrac values), mothur as the 16S analysis pipeline also achieved the best results for the unweighted values, no matter what was the paired-end merger used. The range of values achieved for this case was between 49 and 67 %.

The best result was 67% which was achieved by mothur using either Greengenes or RDP as the references. In this case, mothur performed better with references different from the recommended (which is Silva). For the weighted version, Usearch as the pipeline achieved the best results with also either Greengenes or Silva as the references.

It was with mothur both as merger and pipeline, that this dataset has achieved 100% precision, with either Greengenes or RDP as the references. In every combination tested, QIIME was responsible for the best results for recall in `Mock1`.

5.2.3 Discussion

The fact that the merger with the best recall, mothur - in combination with any pipeline used - obtained the best Unifrac similarity values either in weighted or unweighted versions, appears to be a contradiction since this means that it is more important to have more merged reads available, not being important if they were correctly merged or not. If there are reads that were not merged correctly, and which are to be the input for the downstream pipelines, that should have a negative impact on the taxonomic identification executed by the pipelines, but the results do not agree with that.

In the mergers' evaluation using Mock2, mothur achieved a good precision value (above 0.98), which means that if the remaining 2% are false positives, its

Table 5.3: Unweighted Unifrac similarity values calculated for dataset Mock2

Unweighted Unifrac Mock2		pipeline		
merger	16S reference	mothur	QIIME	USEARCH
FastqJoin	GG	65,12%	30,38%	40,00%
	RDP	65,00%	30,38%	37,33%
	SILVA	58,54%	30,38%	31,08%
FLASH	GG	54,90%	29,76%	40,00%
	RDP	60,47%	29,76%	37,33%
	SILVA	48,98%	29,76%	30,67%
mothur	GG	66,67%	30,38%	47,06%
	RDP	66,67%	30,38%	41,18%
	SILVA	61,54%	30,38%	33,33%
PEAR	GG	56,00%	29,76%	23,30%
	RDP	60,47%	29,76%	15,64%
	SILVA	48,98%	29,76%	16,91%
USEARCH	GG	57,14%	29,76%	16,55%
	RDP	60,47%	29,76%	13,08%
	SILVA	48,98%	32,14%	14,02%

For the weighted values, and a table with all the used metrics for this dataset, including precision and recall, please see table 5 on the appendix.

impact on the qualitative Unifrac values shall be lower, since these were numerically higher than the ones in Mock1. Also the better overall quality of Mock2 should play here an effect.

5.3 Source code availability

- The main evaluation pipeline is written in Python and available with the name `sub16s` at `https://bitbucket.org/sviana/sub16s`.
- The unifrac similarity calculator was implemented in Java, and reused code authored by my supervisor Daniel Faria, and it's available with the name `unifracsim` at `https://bitbucket.org/sviana/unifracsim.git`.
- The mothur-wrapper is implemented in Python and available at `https://github.com/digfish/mothur-wrapper` .

Chapter 6

Conclusion

In the light of what was exposed in the two last chapters, we can conclude that:

- the mergers analysis prove that the mergers with statistical background achieved the best balance between precision and recall;
- since the majority of the mergers achieved good precision, the way to break the tie was to select the one with the best recall;
- the mergers with good recall also can be useful in the downstream analysis, as was revealed by mothur;
- trimming the datasets has no special effects on the mergers effectiveness, but degrades the precision and slightly improves the recall;
- the effect of the overall quality scores in the results obtained were demonstrated through their influence in the mergers ability to merge in a qualitative or quantitative way;
- the results of the analysis pipelines when Mock1 is used are not very conclusive, only QIIME using the merged reads by mothur achieves preponderant values. Moreover, QIIME achieves the most balanced results, independently of the database or merger used;
- in overall, in the downstream steps of analysis, after the merge, mothur revealed to be the best 16S analysis pipeline tool, as testified by the results obtained on dataset Mock2.

Regarding which can be the better choices for the 16S pipeline in each situation, QIIME can be one of them if we take into consideration that achieved the

best values for the dataset with most reads, and also because it uses FastqJoin, one of the best mergers . It is quicker and faster than the other pipelines and also has the less steep learning curve.

mothur as the pipeline of choice for datasets with less reads and better overall quality can be the best candidate, since it requires greater computational resources namely memory to work properly.

Just two datasets were used in this evaluation. More could have been used, but there is properly not much abundance of publicly available mock community datasets. Bigger datasets (in the order of the millions of reads) could have been also put to the test: but BLAST consumed most of the time of the evaluation workflow and the hardware (an eight-cores Intel Xeon 3 GHz with 20 GB RAM) provided for the execution also has its limitations. For each new merger added we have to execute one extra BLAST to analyze the results it produced.

Besides the range of the different tools tested, either being mergers, pipelines or databases, the evaluation pipeline developed in this work are ready to include more tools to be tested, with no much implementation effort.

In either way, it remains open the possibility to research which effect the dataset size can have over the pipelines performance. Another avenues of exploration can also be considered: with different mock communities, in equal or dissimilar concentrations, and with more *taxa* to add more diversity, test the ability that each pipeline has in the OTU resolution, or, in other words, the capacity that each pipeline has in distinguishing two closer *taxon* in a threshold similarity of 99%.

Furthermore research the possibility of the effects that each different clustering methods, others than the defaults, can have over on the final number of OTUs.

As a final remark, the decision on which framework must be selected depends on our goals: if we want just want to know the *taxa* present in the community, or, besides that, we pretend also to know their abundances. And if we have already an *a priori* knowledge regarding the composition and the dataset size these can also be an important factor for that decision.

Appendices

A Materials and Methods

A.1 Technical details regarding the pipelines

A.1.1 QIIME

By default, QIIME expects as is input the dataset in raw form, properly in the form of multiplexed samples, which has in many DNA samples that were joined together to be sequenced in just one run. Each sample is identified by a specific barcode which is a short oligonucleotide portion that is added to the ends of the DNA fragments to be sequenced.

So, there was a need to have to write down a "map file" that lists, one per line, the barcode that identifies each sample on the multiplexed dataset. As our mock communities datasets were already demultiplexed, so this was an unneeded step.

But QIIME only way to accept FASTQ files as input was to use the command "split_libraries_fastq.py" which needs the map file for demultiplexing, and since we had an already demultiplexed dataset, there was a need to create a script that emulated a 'dummy sample' in order to use that particular command. This lone command is in reality a script that invokes other QIIME commands, doing the cleaning and trimming through the quality scores that goes in those FASTQ files. The user can customize these steps. Even if this one step does essentially nothing in terms of what a 16S pipeline is supposed to do, it is anyway important for the further steps, since it converts the reads from the FASTQ in the suited format for the further downstream steps to be executed by QIIME.

The next command that comes after, "pick_open_reference_otus.py" calls a chain of other scripts on the background, but its essential steps are: it uses the configured reference to compare the existing reads with some sequence identity threshold. Stores the ones that have a identity above that threshold and forget the other ones. These forgotten sequences which couldn't find any identical ones on the 16S reference are then used for a *de novo* clustering. They could be potential sequences from not yet known organisms. The problem with this *de novo* clustering is that it takes a lot of resources, so it can be computationally

infeasible, so this method is applied over just a small subsample instead of the whole dataset, one in which couldn't be found any already known sequences for.

Before the clustering there was still a round of alignment with a consensus template, which can be useful to restrict the length of the alignment template to just a small portion of it - which could extend to some dozens of thousands of nucleotides long if this step was not executed, for the sake of reducing computational resources. This concrete step is performed by the `filter_alignment.py` command. Behind the scenes, the tool `pyNAST` (Caporaso *et al.*, 2010a) is called to execute it.

Having then the representative sequences collected by both means (close and open reference), QIIME is now ready to create the OTU tables, being each sequence a centroid in the whole space of possible sequences. A distances matrix is calculated between all the 'remaining' sequences in the dataset and these centroids. We can imagine these centroids as some kind of bins in which each sequence will fall into if it is at least similar above some threshold value (*eg* 97% for the genus level). In the end, we'll count all the sequences that fell into each one of these 'so-called' bins. The OTU table is saved as a BIOM file. For the pos-processing steps regarding what to do with these BIOM files see section The Pos-pipelines processing.

A.1.2 Mothur

`mothur`, taken as a complete 16S analysis tool, needs a more closer control than QIIME. The `mothur` executable is just one single binary with all its dependencies statically compiled into it. It can be an advantage since we won't go into potential version conflicts but either way the executable can achieve some huge size in file length.

With all this functionality built in commands as in the likeness of QIIME there is no, on the contrary, a 'do-it all' command. The entire operation procedure is done trough a command-line Interface (CLI), where the commands are entered in and executed. The CLI looks much like the R CLI.

We felt a need to write a simple wrapper script in Python around the mothur executable, in order for not to have to type the same commands again at the

start of execution. `mothur` by itself supports scripting, but it does not allow to pass argument values directly from the operating system default interpreter. The mothur wrapper script simplified this particular feature.

`mothur` works with FASTA/FASTQ files, and also reads alignment templates and references in FASTA too. Besides the alignment, it can filter out repeated sequences, and remove chimeras. The clustering is done in a two-step process: first the distance matrix between all the unique sequences is computed and is with this matrix that the clustering is done. By default is used the average neighbor method. The default cluster command is also able to do the clustering from a more thick to the more coarse level.

At these different levels, the representative sequences for each cluster are taxonomically classified and the OTU tables built. By default, the earlier versions didn't support writing the OTU table in the BIOM format. Later versions were able to do that. It involved creating a group file which identifies which group each sequence belongs to.

A.1.3 Usearch

`Usearch` as opposed of the last two pipelines, is not open source: you have to download the compiled static binary after a registration step. The available version is a 32-bit one, which is able to use until 2 GB of memory. Since the majority of datasets are in an order of magnitude of size bigger, and for example, the processing in the clustering step will need to store a distance matrix with which big datasets will easily exceed that size limit. So, this 32-bit version was just for demonstration purposes. Being the bigger of the two datasets used in our study of roughly 450000 paired-end reads each one 255 bases long, this not posed a problem luckily.

Usearch uses for the taxonomic assignment its own native format of databases, which is called UTAX. So the obvious need to convert the 16S references to this format. That had to be done both for Silva and Greengenes. For the RDP, binaries in the UTAX format were already available for download on the Usearch website. This conversion step are prolongated and divided in several substeps which have to be accomplished before proceeding to the next ones.

B Supplementary figures and tables

Table 1: Parameter configurations tested for the mergers

configuration	acronym	command line
bbmerge_default	B	`bbmerge.sh in1=%forward in2=%reverse out=%out`
bbmerge_maxloose	B*	`bbmerge.sh in1=%forward in2=%reverse out=%out maxloose`
fastq-join_custom	J*	`fastq-join %forward %reverse -o  %out -m10`
fastq-join_default	J	`fastq-join %forward %reverse -o  %out`
flash_custom	F*	`flash -t$cpus %forward %reverse -m10 -M$max_overlap -c > %out`
flash_default	F	`flash -t$cpus %forward %reverse -c > %out`
mothur	M	`mothur "#make.contigs(ffastq=%forward,rfastq=%reverse, Processors=$cpus)"`
pandaseq_custom	P*	`pandaseq -f %forward -r %reverse -F -T $cpus -w %out  -o 10 -O $max_overlap  -g lixo.txt`
pandaseq_default	P	`pandaseq -f %forward -r %reverse -F -T $cpus -w %out -g lixo.txt`
pear_default	E	`pear -f %forward -r %reverse -o %out -j$cpus`
pear_statsmode2	E*	`pear  -j $cpus -g2 -f %forward -r %reverse -o %out`
usearch	U	`usearch -fastq_mergepairs %forward -reverse %reverse -fastqout %out`
usearch_custom	U*	`usearch -fastq_mergepairs %forward -reverse %reverse -fastqout %out -fastq_minovlen 10 -minhsp 10`

parameters passed for mergers script batch	
%forward	forward reads fastq
%reverse	reverse reads fastq
%out	merged reads output fastq
$cpus	number of CPU cores
$max_overlap	max overlap found on pre-merge blast of paired-end reads

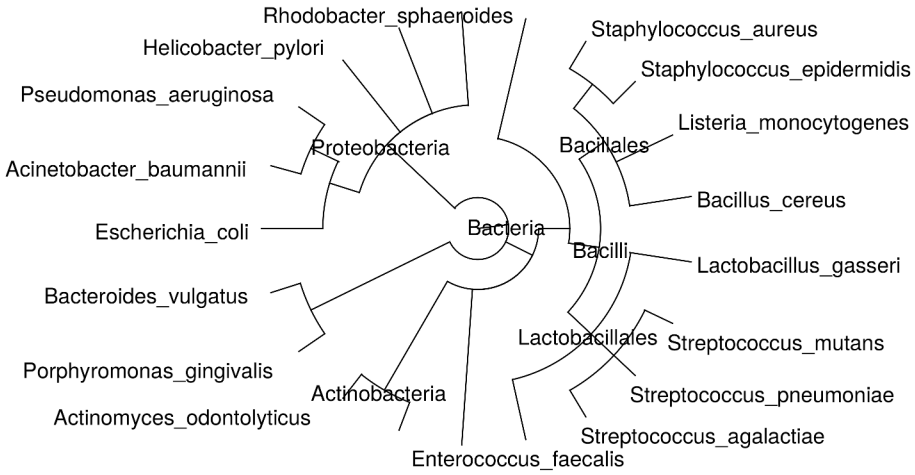

Figure 1: Mock1 phylogenetic tree

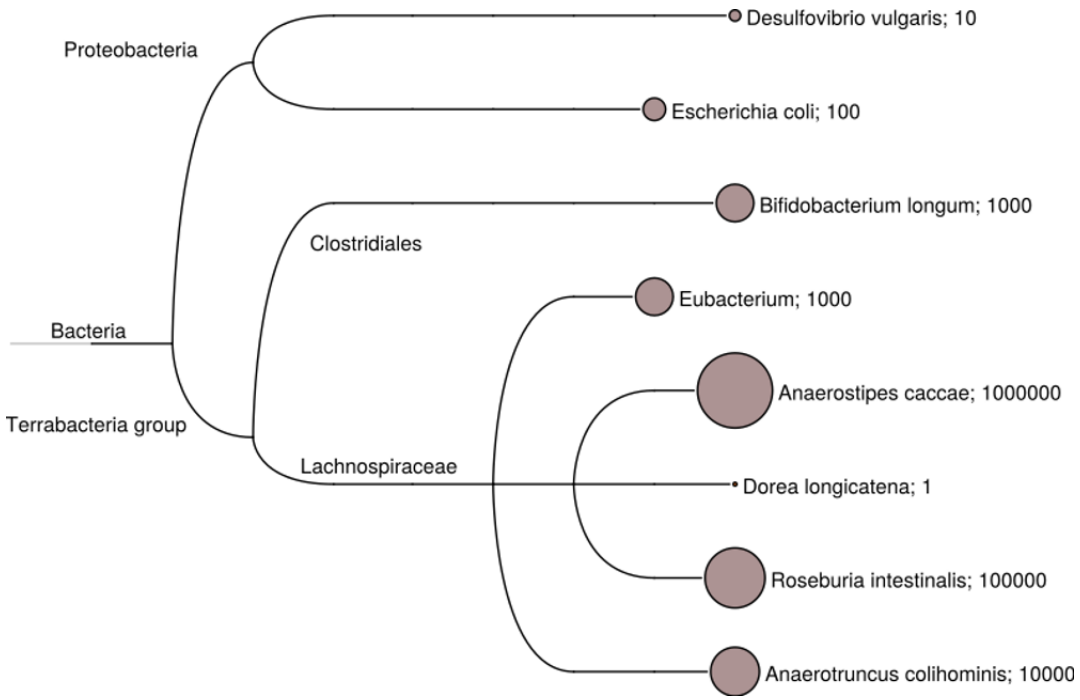

Figure 2: Mock2 phylogenetic tree
The relative abundances are depicted in proportion to the circles radius (logarithmic scale)

Table 2: Precision, Recall and F-Measure on Mock1

merger		precision	recall	f_measure
bbmerge_default	B	0.9408	0.0863	0.1582
bbmerge_maxloose	B*	0.9736	0.4676	0.6317
fastq-join_default	J*	0.9861	0.4156	0.5847
fastq-join_custom	J	0.9825	0.9495	0.9657
flash_default	F*	0.9826	0.5649	0.7174
flash_custom	F	0.9843	0.8908	0.9352
mothur	M	0.9507	1.0000	0.9747
pandaseq_custom	P*	0.9724	0.9741	0.9732
pandaseq_default	P	0.9768	0.9731	0.975
pear_default	E	0.9817	0.9892	0.9855
pear_statsmode2	E*	0.9795	0.9976	0.9885
usearch	U	0.9791	0.9904	0.9847
usearch_custom	U*	0.972	0.9984	0.985

Table 3: Precision, Recall and F-Measure on Mock2

merger		precision	recall	f_measure
bbmerge_default	B	0.9792	0.585	0.7324
bbmerge_maxloose	B*	0.983	0.7207	0.8317
fastq-join_custom	J*	0.9864	0.9012	0.9419
fastq-join_default	J	0.9864	0.9012	0.9419
flash_custom	F*	0.9874	0.9951	0.9913
flash_default	F	0.9858	0.8724	0.9257
mothur	M	0.9809	1.0000	0.9904
pandaseq_custom	P*	0.987	0.9555	0.971
pandaseq_default	P	0.987	0.9562	0.9714
pear_default	E	0.9872	0.9999	0.9935
pear_statsmode2	E*	0.9872	0.9999	0.9935
usearch	U	0.9871	1.0000	0.9935
usearch_custom	U*	0.9871	1.0000	0.9935

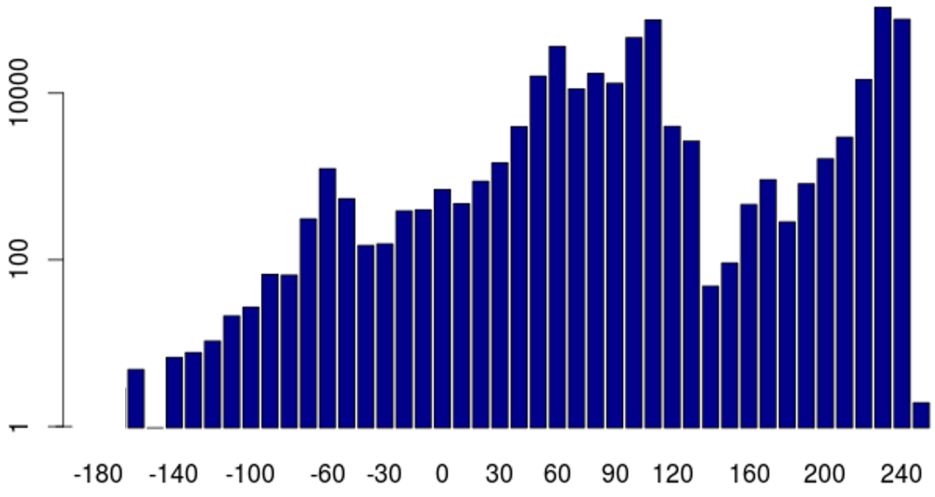

Figure 3: Mock1 putative overlaps distribution (logarithmic scale)

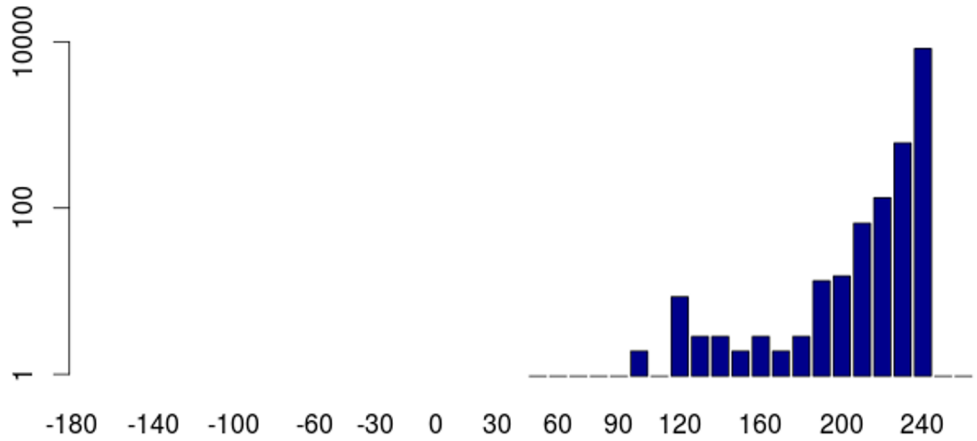

Figure 4: Mock2 putative overlaps distribution (logarithmic scale)

Table 4: Mock1 pipelines evaluation results

ordered by unweighted Unifrac (U_{sim}), along with weighted (WU_{sim}) Unifrac, precision and recall (all values are in percent)

Merger	Pipeline	BD	U_{sim}	WU_{sim}	precision	recall
mothur	QIIME	SILVA	61	68	70	83
mothur	QIIME	GG	59	68	68	82
mothur	QIIME	RDP	58	68	67	81
USEARCH	QIIME	RDP	55	65	61	85
FLASH	mothur	GG	54	78	62	81
FastqJoin	mothur	GG	54	78	61	81
PEAR	mothur	GG	53	78	61	81
mothur	USEARCH	GG	53	75	64	75
FLASH	QIIME	GG	53	66	58	85
PEAR	QIIME	GG	53	66	58	85
USEARCH	mothur	GG	52	78	61	80
mothur	mothur	GG	52	78	59	83
FLASH	QIIME	RDP	52	65	58	85
FLASH	USEARCH	GG	52	75	65	73
FastqJoin	QIIME	RDP	52	65	58	84
USEARCH	QIIME	SILVA	52	70	57	85
FastqJoin	QIIME	GG	52	66	57	85
PEAR	QIIME	RDP	52	65	57	85
mothur	USEARCH	SILVA	51	68	68	67
FLASH	USEARCH	RDP	50	75	62	73
mothur	USEARCH	RDP	50	75	62	73
FLASH	QIIME	SILVA	50	70	55	85
FastqJoin	USEARCH	GG	50	74	61	73
PEAR	QIIME	SILVA	50	70	54	85
FastqJoin	QIIME	SILVA	50	70	55	84
FLASH	USEARCH	SILVA	49	68	67	65
FastqJoin	USEARCH	SILVA	49	68	65	67
USEARCH	QIIME	GG	49	65	53	85
PEAR	USEARCH	SILVA	48	68	64	67
FastqJoin	USEARCH	RDP	48	75	58	73
PEAR	USEARCH	GG	48	74	58	73
FLASH	mothur	SILVA	48	71	59	71
mothur	mothur	SILVA	48	77	57	74
USEARCH	USEARCH	GG	48	75	58	73
PEAR	USEARCH	RDP	47	75	57	73
mothur	mothur	RDP	47	76	59	69
FastqJoin	mothur	SILVA	46	71	57	71
PEAR	mothur	SILVA	46	70	57	71
USEARCH	mothur	SILVA	46	70	56	71
USEARCH	USEARCH	SILVA	46	68	58	69
USEARCH	USEARCH	RDP	45	75	54	73
PEAR	mothur	RDP	42	75	53	67
FastqJoin	mothur	RDP	42	75	53	67
FLASH	mothur	RDP	42	75	53	67
USEARCH	mothur	RDP	41	75	52	67

Table 5: Mock2 pipelines evaluation results

ordered by unweighted Unifrac (U_{sim}), along with weighted (WU_{sim}) Unifrac, precision and recall (all values are in percent)

Merger	Pipeline	BD	U_{sim}	WU_{sim}	precision	recall
mothur	mothur	GG	67	58	90	72
mothur	mothur	RDP	67	59	100	67
FastqJoin	mothur	GG	65	58	88	72
FastqJoin	mothur	RDP	65	59	96	67
mothur	mothur	SILVA	62	59	100	62
FLASH	mothur	RDP	60	59	87	67
PEAR	mothur	RDP	60	59	87	67
USEARCH	mothur	RDP	60	59	87	67
FastqJoin	mothur	SILVA	59	58	92	62
USEARCH	mothur	GG	57	58	74	72
PEAR	mothur	GG	56	58	72	72
FLASH	mothur	GG	55	58	70	72
FLASH	mothur	SILVA	49	59	71	62
PEAR	mothur	SILVA	49	59	71	62
USEARCH	mothur	SILVA	49	59	71	62
mothur	USEARCH	GG	47	67	67	62
mothur	USEARCH	RDP	41	63	49	72
FastqJoin	USEARCH	GG	40	65	53	62
FLASH	USEARCH	GG	40	65	53	62
FastqJoin	USEARCH	RDP	37	62	44	72
FLASH	USEARCH	RDP	37	62	44	72
mothur	USEARCH	SILVA	33	68	43	59
USEARCH	QIIME	SILVA	32	62	38	69
FastqJoin	USEARCH	SILVA	31	67	40	59
FLASH	USEARCH	SILVA	31	66	39	59
FastqJoin	QIIME	GG	30	62	38	62
FastqJoin	QIIME	RDP	30	62	38	62
FastqJoin	QIIME	SILVA	30	62	38	62
mothur	QIIME	GG	30	64	38	62
mothur	QIIME	RDP	30	63	38	62
mothur	QIIME	SILVA	30	64	38	62
FLASH	QIIME	GG	30	62	36	64
FLASH	QIIME	RDP	30	62	36	64
FLASH	QIIME	SILVA	30	62	36	64
PEAR	QIIME	GG	30	62	36	64
PEAR	QIIME	RDP	30	62	36	64
PEAR	QIIME	SILVA	30	62	36	64
USEARCH	QIIME	GG	30	62	36	64
USEARCH	QIIME	RDP	30	62	36	64
PEAR	USEARCH	GG	23	65	27	62
PEAR	USEARCH	SILVA	17	66	19	59
USEARCH	USEARCH	GG	17	58	18	62
PEAR	USEARCH	RDP	16	61	17	72
USEARCH	USEARCH	SILVA	14	62	16	59
USEARCH	USEARCH	RDP	13	56	14	72

Table 6: Weighted Unifrac similarity values calculated for dataset Mock1

Weighted Unifrac Mock1		pipeline		
merger	16S reference	mothur	QIIME	USEARCH
FastqJoin	GG	77,89%	66,10%	74,49%
	RDP	74,90%	64,73%	74,76%
	SILVA	70,50%	69,86%	67,51%
FLASH	GG	77,89%	66,35%	74,96%
	RDP	74,81%	64,72%	75,03%
	SILVA	70,65%	69,58%	67,89%
mothur	GG	78,37%	68,44%	74,60%
	RDP	76,24%	68,12%	74,73%
	SILVA	76,83%	68,04%	67,80%
PEAR	GG	77,99%	66,23%	74,50%
	RDP	75,00%	65,12%	74,87%
	SILVA	70,50%	69,76%	67,54%
USEARCH	GG	77,63%	65,48%	74,52%
	RDP	74,82%	65,28%	74,83%
	SILVA	70,32%	69,54%	67,56%

Table 7: Weighted Unifrac similarity values calculated for dataset Mock2

Weighted Unifrac Mock2		pipeline		
merger	16S reference	mothur	QIIME	USEARCH
FastqJoin	GG	57,63%	61,97%	65,18%
	RDP	58,71%	61,90%	61,86%
	SILVA	58,44%	62,02%	66,59%
FLASH	GG	57,65%	61,85%	65,02%
	RDP	59,00%	61,77%	61,71%
	SILVA	58,72%	61,89%	66,38%
mothur	GG	57,56%	63,88%	66,72%
	RDP	58,78%	63,44%	63,16%
	SILVA	58,76%	63,53%	68,01%
PEAR	GG	57,67%	61,84%	64,79%
	RDP	58,98%	61,76%	61,13%
	SILVA	58,68%	61,86%	65,79%
USEARCH	GG	58,20%	62,47%	58,03%
	RDP	59,28%	61,78%	56,08%
	SILVA	59,03%	61,86%	61,79%

References

Aronesty, E. (2013). Comparison of Sequencing Utility Programs. *The Open Bioinformatics Journal*, **7**, 1–8. 12

BBMap - Bushnell B. - sourceforge.net/projects/bbmap/. 12

Bushnell, B. & Rood, J. (2015). BBMerge_poster_v2.pptx. 12

Callahan, B.J., Mcmurdie, P.J., Rosen, M.J., Han, A.W., Johnson, A.J.A. & Holmes, S.P. (2016). dada2: high-resolution sample inference from illumina amplicon data. *Nature Methods*. 21

Caporaso, J.G., Bittinger, K., Bushman, F.D., DeSantis, T.Z., Andersen, G.L. & Knight, R. (2010a). PyNAST: a flexible tool for aligning sequences to a template alignment. *Bioinformatics*, **26**, 266–267. 56

Caporaso, J.G., Kuczynski, J., Stombaugh, J., Bittinger, K., Bushman, F.D., Costello, E.K., Fierer, N., Pena, A.G., Goodrich, J.K., Gordon, J.I. & Others (2010b). QIIME allows analysis of high-throughput community sequencing data. *Nature methods*, **7**, 335–336. 17

Caporaso, J.G., Lauber, C.L., Walters, W.A., Berg-Lyons, D., Lozupone, C.A., Turnbaugh, P.J., Fierer, N. & Knight, R. (2011). Global patterns of 16S rRNA diversity at a depth of millions of sequences per sample. *Proceedings of the National Academy of Sciences*, **108**, 4516–4522. 6

Chakravorty, S., Helb, D., Burday, M., Connell, N. & Alland, D. (2007). A detailed analysis of 16S ribosomal RNA gene segments for the diagnosis of pathogenic bacteria. *Journal of microbiological methods*, **69**, 330–339. 5

REFERENCES

Cole, J.R., Wang, Q., Fish, J.A., Chai, B., McGarrell, D.M., Sun, Y., Brown, C.T., Porras-Alfaro, A., Kuske, C.R. & Tiedje, J.M. (2014). Ribosomal Database Project: Data and tools for high throughput rRNA analysis. *Nucleic Acids Research*, **42**, D633–D642. 19

D'Amore, R., Ijaz, U.Z., Schirmer, M., Kenny, J.G., Gregory, R., Darby, A.C., Shakya, M., Podar, M., Quince, C. & Hall, N. (2016). A comprehensive benchmarking study of protocols and sequencing platforms for 16S rRNA community profiling. *BMC Genomics*, **17**, 55. 1

Del Fabbro, C., Scalabrin, S., Morgante, M. & Giorgi, F.M. (2013). An Extensive Evaluation of Read Trimming Effects on Illumina NGS Data Analysis. *PLoS ONE*, **8**, e85024. 11

DeSantis, T.Z., Hugenholtz, P., Larsen, N., Rojas, M., Brodie, E.L., Keller, K., Huber, T., Dalevi, D., Hu, P. & Andersen, G.L. (2006). Greengenes, a Chimera-Checked 16S rRNA Gene Database and Workbench Compatible with ARB. *Applied and Environmental Microbiology*, **72**, 5069–5072. 18

Edgar, R.C. (2010). Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, **26**, 2460–2461. 17

Edgar, R.C. & Flyvbjerg, H. (2015). Error filtering, pair assembly and error correction for next-generation sequencing reads. *Bioinformatics*, **31**, 3476–3482. 16, 21, 34, 36

Fuller, C.W., Middendorf, L.R., Benner, S.A., Church, G.M., Harris, T., Huang, X., Jovanovich, S.B., Nelson, J.R., Schloss, J.A., Schwartz, D.C. & Vezenov, D.V. (2009). The challenges of sequencing by synthesis. *Nature biotechnology*, **27**, 1013–23. 7

Haas, B.J., Gevers, D., Earl, A.M., Feldgarden, M., Ward, D.V., Giannoukos, G., Ciulla, D., Tabbaa, D., Highlander, S.K., Sodergren, E., Methé, B., DeSantis, T.Z., Petrosino, J.F., Knight, R.

& Birren, B.W. (2011). Chimeric 16S rRNA sequence formation and detection in Sanger and 454-pyrosequenced PCR amplicons. *Genome Research*, **21**, 494–504. 7

Huse, S.M., Welch, D.M., Morrison, H.G. & Sogin, M.L. (2010). Ironing out the wrinkles in the rare biosphere through improved OTU clustering. *Environmental Microbiology*, **12**, 1889–1898. 8

Konstantinidis, K.T. & Tiedje, J.M. (2005). Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences of the United States of America*, **102**, 2567–72. 8

Kopylova, E., Navas-Molina, J.A., Mercier, C., Xu, Z.Z., Mahé, F., He, Y., Zhou, H.W., Rognes, T., Caporaso, J.G. & Knight, R. (2016). Open-Source Sequence Clustering Methods Improve the State Of the Art. *mSystems*, **1**, e00003–15. 1, 10

Kozich, J.J., Westcott, S.L., Baxter, N.T., Highlander, S.K. & Schloss, P.D. (2013). Development of a dual-index sequencing strategy and curation pipeline for analyzing amplicon sequence data on the miseq illumina sequencing platform. *Applied and Environmental Microbiology*, **79**, 5112–5120. 21

Lozupone, C. & Knight, R. (2005). UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and environmental microbiology*, **71**, 8228–35. 19

Ludwig, W., Strunk, O., Westram, R., Richter, L., Meier, H., Yadhukumar, A., Buchner, A., Lai, T., Steppi, S., Jacob, G., Förster, W., Brettske, I., Gerber, S., Ginhart, A.W., Gross, O., Grumann, S., Hermann, S., Jost, R., König, A., Liss, T., Lübmann, R., May, M., Nonhoff, B., Reichel, B., Strehlow, R., Stamatakis, A., Stuckmann, N., Vilbig, A., Lenke, M., Ludwig, T., Bode, A. & Schleifer, K.H. (2004). ARB: A software environment for sequence data. *Nucleic Acids Research*, **32**, 1363–1371. 18

Magoč, T. & Salzberg, S.L. (2011). FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, **27**, 2957–2963. 7, 13

Masella, A.P., Bartram, A.K., Truszkowski, J.M., Brown, D.G. & Neufeld, J.D. (2012). PANDAseq: paired-end assembler for illumina sequences. *BMC Bioinformatics*, **13**, 31. xix, 7, 15

Mysara, M., Leys, N., Raes, J. & Monsieurs, P. (2016). IPED: a highly efficient denoising tool for Illumina MiSeq Paired-end 16S rRNA gene amplicon sequencing data. *BMC Bioinformatics*, **17**, 192. 21

Pruesse, E., Quast, C., Knittel, K., Fuchs, B.M., Ludwig, W., Peplies, J. & Glöckner, F.O. (2007). SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucleic Acids Research*, **35**, 7188–7196. 18

Rideout, J.R., He, Y., Navas-Molina, J.a., Walters, W.a., Ursell, L.K., Gibbons, S.M., Chase, J., McDonald, D., Gonzalez, A., Robbins-Pianka, A., Clemente, J.C., Gilbert, J.a., Huse, S.M., Zhou, H.W., Knight, R. & Caporaso, J.G. (2014). Subsampled open-reference clustering creates consistent, comprehensive OTU definitions and scales to billions of sequences. *PeerJ*. 10

Santamaria, M., Fosso, B., Consiglio, A., De caro, G., Grillo, G., Licciulli, F., Liuni, S., Marzano, M., Alonso-alemany, D., Valiente, G. & Pesole, G. (2012). Reference databases for taxonomic assignment in metagenomics. *Briefings in Bioinformatics*, **13**, 682–695. 18

Schloss, P.D. & Westcott, S.L. (2011). Assessing and improving methods used in operational taxonomic unit-based approaches for 16S rRNA gene sequence analysis. *Applied and Environmental Microbiology*, **77**, 3219–3226. 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., Sahl, J.W., Stres, B., Thallinger, G.G., Van Horn,

D.J. & Weber, C.F. (2009). Introducing mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Applied and Environmental Microbiology*, **75**, 7537–7541. 14, 17

Soergel, D.A.W., Dey, N., Knight, R. & Brenner, S.E. (2012). Selection of primers for optimal taxonomic classification of environmental 16S rRNA gene sequences. *The ISME journal*, **6**, 1440–4. 5

Větrovský, T. & Baldrian, P. (2013). The Variability of the 16S rRNA Gene in Bacterial Genomes and Its Consequences for Bacterial Community Analyses. *PLoS ONE*, **8**, e57923. 8

Westcott, S.L. & Schloss, P.D. (2015). De novo clustering methods outperform reference-based methods for assigning 16S rRNA gene sequences to operational taxonomic units. *PeerJ*, **3**, e1487. 10

Woese, C.R., Kandler, O. & Wheelis, M.L. (1990). Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. *Proceedings of the National Academy of Sciences*, **87**, 4576–4579. 5

Yang, B., Wang, Y. & Qian, P.Y. (2016). Sensitivity and correlation of hypervariable regions in 16S rRNA genes in phylogenetic analysis. *BMC Bioinformatics*, **17**, 135. xix, 6

Zhang, J., Kobert, K., Flouri, T. & Stamatakis, A. (2014). PEAR: a fast and accurate Illumina Paired-End reAd mergeR. *Bioinformatics*, **30**, 614–620. xix, 8, 15

Zheng, A. (2015). *Evaluating Machine Learning Algorithms*. O'Reilly. 30